

# O que é MVC?

MVC significa **Model - View - Controller** (Modelo - Visão - Controlador) e é um modelo da arquitetura de software que tem a função de separar front-end (que o usuário vê) do back-end (que é o motor da aplicação).

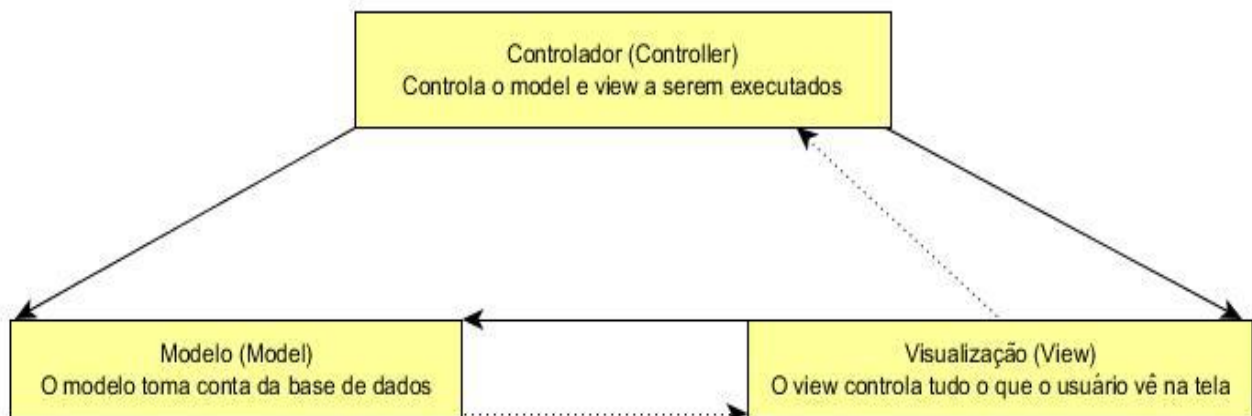
A estrutura MVC funciona da seguinte maneira:

**Model** (modelo) - O Model é responsável por tratar de tudo que é relacionado com os dados, como criar, ler, atualizar e excluir valores da base de dados ([CRUD](#)), tratar das regras de negócios, da lógica e das funções. Apesar de fazer isso tudo, o Model não apresenta nada na tela e não executa nada por si. Normalmente, um View requisita que determinado Model execute uma ação e a mesma é executada dentro do View.

**View** (Visão) - O View é a parte que o usuário vê na tela, como HTML, JavaScript, CSS, Imagens e assim por diante. O View não tem nenhuma ação, mas requisita que o Model execute qualquer ação e mostra os valores retornados para o usuário. É importante ressaltar que um View não depende de nenhum Model, por exemplo, se você vai apenas exibir dados HTML na tela, e não vai precisar de base de dados, talvez um Model não seja necessário.

**Controller** (Controlador) - O Controller é responsável por resolver se um Model e/ou um View é necessário. Caso positivo; ele incluirá os arquivos e funções necessárias para o sistema funcionar adequadamente.

Veja uma imagem representando como o modelo **MVC** funciona:



## Model-view-controller (MVC) 1

Na imagem acima, as linhas sólidas exemplificam partes que têm ligações diretas; as linhas tracejadas mostram ligações indiretas. Isso é mais ágil e pode variar dependendo da sua aplicação e ação que está sendo executada.

## Nosso projeto

Nosso projeto será criar um sistema de notícias com área administrativa, portanto, é obrigatório que tenhamos o seguinte:

- Sistema de login para os administradores;

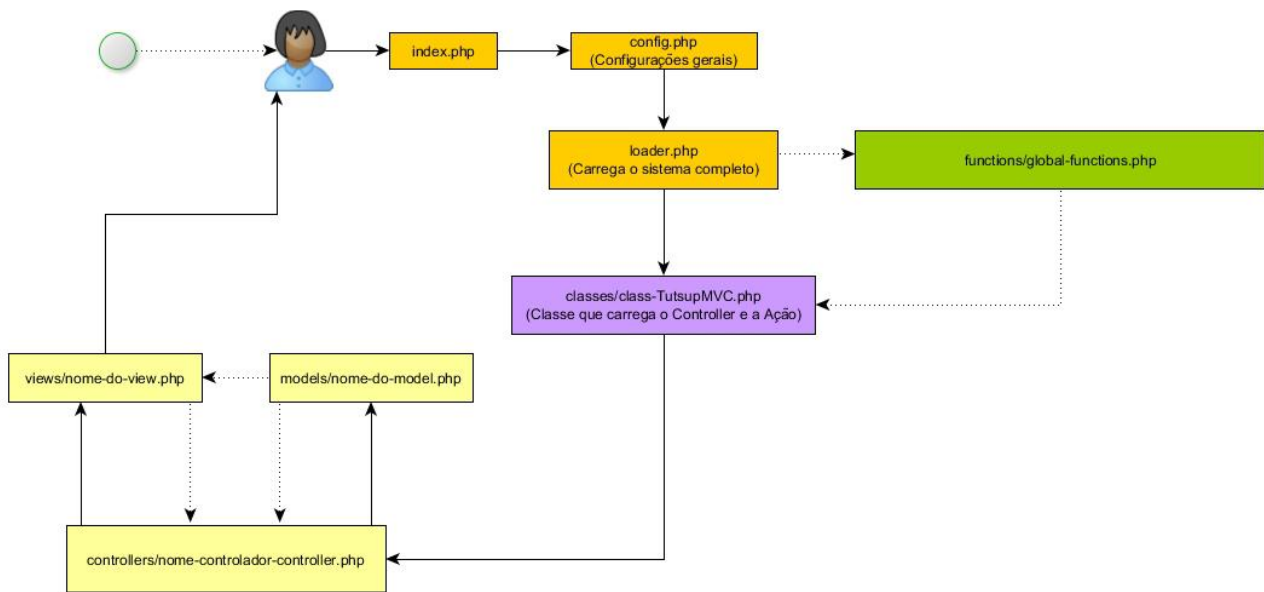
- Sistema de registro de usuários (CRUD);

- Sistema de permissões;

- Sistema de cadastro de notícias (CRUD);

Faremos tudo no modelo MVC, mas para atingir nosso objetivo teremos que criar várias outras pastas e arquivos.

Basicamente, nossa estrutura ficar á como na imagem abaixo:



## Fluxograma da nossa aplicação MVC em PHP 1

Na imagem acima temos uma apresentação de como a informação vai passar pelo nosso sistema. Veja uma descrição:

1. O usuário acessa o site;
2. O arquivo `index.php` apenas inclui o arquivo `config.php`;
3. O arquivo `config.php` é responsável por registrar nossas configurações e carregar o arquivo `loader.php`;
4. O arquivo `loader.php` carrega o arquivo `global-functions.php`, que é responsável por manter todas as funções globais. Na verdade, a função mais importante que temos ali é a `_autoload`, para carregar classes automaticamente. Ele também é responsável por instanciar a classe “`TutsupMVC`” que vai controlar todo o início da aplicação.

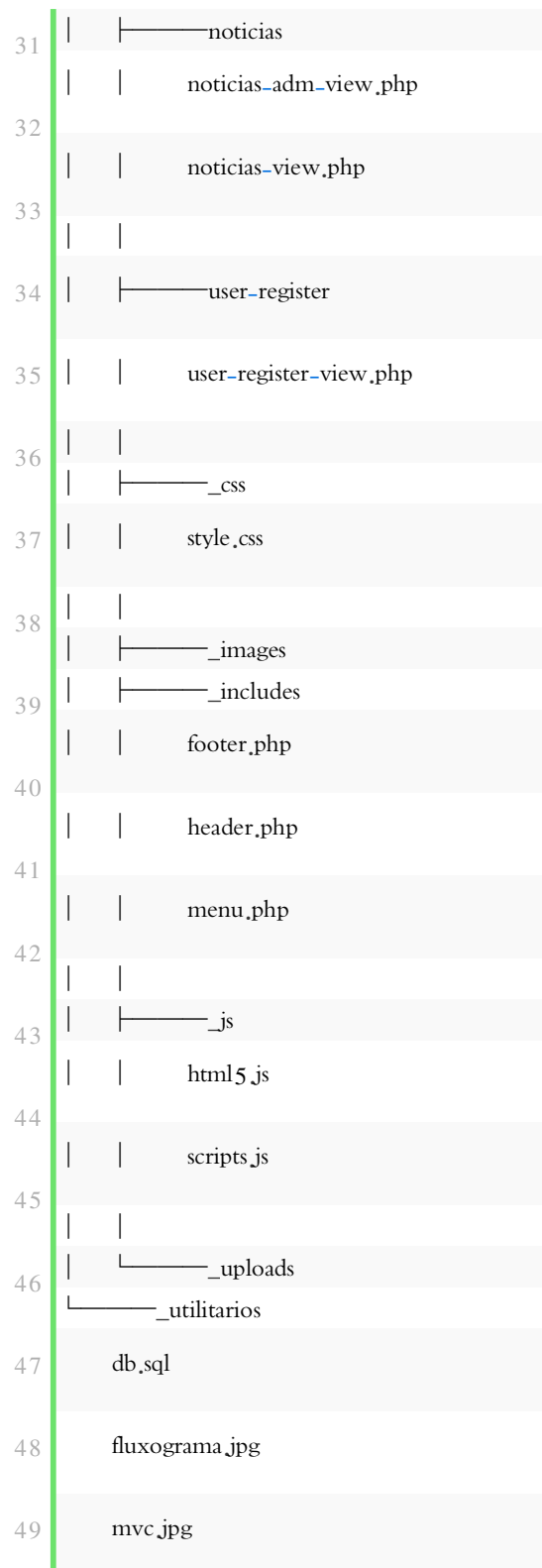
5. A classe “**TutsupMVC**” vai verificar se um controlador foi requisitado (pela URL) e incluir o mesmo. Ela também vai verificar se alguma ação do controlador foi requisitada (ainda pela URL). Caso contrário, a ação “index” do controlador será executada. Sendo assim, todo controlador tem que ter pelo menos uma ação, chamada de “**index**”.
6. O **arquivo controlador** (controller) é responsável por ter todas as ações daquela sessão. Cada ação irá diferenciar os *Models* e/ou *Views* que forem requisitados. Às vezes uma ação pode utilizar um *View* e vários *Models*, ou vice-versa;
7. O **arquivo modelo** (model) terá todos os métodos necessários para executar as ações do View. Este arquivo não é obrigatório;
8. O **arquivo de visão** (view) irá simplesmente mostrar tudo ao usuário que requisitou a ação.

## Estrutura de pastas

Nossas pastas ficarão da seguinte maneira:

```
1 | .htaccess
2 | config.php
3 | index.php
4 | loader.php
5 |
6 | └── classes
7 |     ├── class-MainController.php
8 |     ├── class-MainModel.php
   |     └── class-PasswordHash.php
```

9		class-TutsupDB.php
10		class-TutsupMVC.php
11		class-UserLogin.php
12		
		└── controllers
13		home-controller.php
14		login-controller.php
15		noticias-controller.php
16		user-register-controller.php
17		
		└── functions
18		global-functions.php
19		
		└── includes
20		404.php
21		
		└── models
22		└── noticias
		└── noticias-adm-model.php
23		
		└── user-register
24		
25		user-register-model.php
26		
		└── views
27		└── home
		└── home-view.php
28		
		└── login
29		└── login-view.php
30		



As pastas que têm um *sublinhado* antes do nome, são pastas que incluem arquivos que os views utilizam, mas que não são views. Normalmente são arquivos que não são acessados diretamente, como [header.php](#), [footer.php](#) e outros. Eles são incluídos nos arquivos que precisamos.

Ao decorrer desse curso, pode ser que eu adicione ou remova recursos, mas não se preocupe, vou lhe dizer quando algo for alterado.

## Modelo da URL e parâmetros

Vamos obter todos os nossos parâmetros por [HTTP GET](#) no seguinte formato:

<http://www.exemplo.com/index.php?url=controlador/ação/parametro1/parametro2/etc...>

Se você perceber, temos apenas um parâmetro na URL acima, o `%_GET['url']`. Isso porque vamos utilizar o arquivo `.htaccess` do Apache para reescrever a URL.

A mesma URL acima ficará assim:

<http://www.exemplo.com/controlador/ação/parametro1/parametro2/etc...>

Nossa classe [TutsupMVC](#) vai tratar de separar o controlador, a ação, e enviar o restante dos parâmetros para os métodos dos controladores.

## Base de dados do modelo MVC em PHP

Precisaremos de uma base de dados chamada [Tutsup](#):

Base de dados

```
1 CREATE DATABASE IF NOT EXISTS `tutsup` CHARACTER SET utf8;
```

Uma tabela chamada [users](#):

Tabela users

```
1 CREATE TABLE IF NOT EXISTS `tutsup`.`users` (  
2 `user_id` INT(11) NOT NULL AUTO_INCREMENT,
```

```

3 `user` VARCHAR(255) COLLATE utf8_bin NOT NULL,
4 `user_password` VARCHAR(255) COLLATE utf8_bin NOT NULL,
5 `user_name` VARCHAR(255) COLLATE utf8_bin DEFAULT NULL,
6 `user_session_id` VARCHAR(255) COLLATE utf8_bin DEFAULT NULL,
7 `user_permissions` LONGTEXT COLLATE utf8_bin,
8 PRIMARY KEY (`user_id`)
9 ) ENGINE=MYISAM AUTO_INCREMENT=0 DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

```

Uma tabela chamada **noticias**:

Tabela noticias

```

1 CREATE TABLE IF NOT EXISTS `tutsup`.`noticias` (
2 `noticia_id` INT(11) NOT NULL AUTO_INCREMENT,
3 `noticia_data` DATETIME DEFAULT '0000-00-00 00:00:00',
4 `noticia_autor` VARCHAR(255),
5 `noticia_titulo` VARCHAR(255),
6 `noticia_texto` TEXT,
7 `noticia_imagem` VARCHAR(255),
8 PRIMARY KEY (`noticia_id`)
9 ) ENGINE = MYISAM CHARSET = utf8 ;

```

Precisaremos inserir um usuário chamado “**Admin**” na base de dados.

Os dados do usuário inicial serão:

Usuário: **Admin**

Senha: **admin**

Nosso sistema faz distinção entre letras maiúsculas e minúsculas.



Inserindo usuário Admin na tabela users

```
1 INSERT INTO `tutsup`.`users` (  
2   `user_id`,  
3   `user`,  
4   `user_password`,  
5   `user_name`,  
6   `user_session_id`,  
7   `user_permissions`  
8 )  
9 VALUES  
10 (  
11   NULL,  
12   '% 2a% 08% 2sGQinTFe3GF/YqAYQ66auL9o6HeFCQryHdqUDvuEVN0J1vdhimii',  
13   'Admin',  
14   'ljfp99gvqm2hg2bj6jpu4ol64',  
15   'a:2:{i:0;s:13:"user-register";i:1;s:18:"gerenciar-noticias";}'  
16 );
```

Se você quiser o script completo, basta criar um arquivo com a extensão “.sql”, com o seguinte texto:

Script para criar a base de dados e as tabelas

```
1 CREATE DATABASE IF NOT EXISTS `tutsup` CHARACTER SET utf8;  
2  
3 CREATE TABLE IF NOT EXISTS `tutsup`.`noticias` (  
4
```

```
4 `noticia_id` INT (11) NOT NULL AUTO_INCREMENT,
5 `noticia_data` DATETIME DEFAULT '0000-00-00 00:00:00',
6 `noticia_autor` VARCHAR (255),
7 `noticia_titulo` VARCHAR (255),
8 `noticia_texto` TEXT,
9 `noticia_imagem` VARCHAR (255),
10 PRIMARY KEY (`noticia_id`)
11 ) ENGINE = MYISAM CHARSET = utf8 ;
12
13 CREATE TABLE IF NOT EXISTS `tutsup`.`users` (
14     `user_id` INT(11) NOT NULL AUTO_INCREMENT,
15     `user` VARCHAR(255) COLLATE utf8_bin NOT NULL,
16     `user_password` VARCHAR(255) COLLATE utf8_bin NOT NULL,
17     `user_name` VARCHAR(255) COLLATE utf8_bin DEFAULT NULL,
18     `user_session_id` VARCHAR(255) COLLATE utf8_bin DEFAULT NULL,
19     `user_permissions` LONGTEXT COLLATE utf8_bin,
20     PRIMARY KEY (`user_id`)
21 ) ENGINE=MYISAM AUTO_INCREMENT=0 DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
22
23 INSERT INTO `tutsup`.`users` (
24     `user_id`,
25     `user`,
26     `user_password`,
```

```

26 `user_name`,
27 `user_session_id`,
28 `user_permissions`
29 )
30 VALUES
31 (
32     NULL,
33     'Admin',
34     '% 2a% 08% 2sGQinTFe3GF/YqAYQ66auL9o6HeFCQryHdqUDvuEVN0J1vdhimii',
35     'Admin',
36     'ljfp99gvqm2hg2bj6jpu4ol64',
37     'a:2:{i:0;s:13:'user-register';i:1;s:18:'gerenciar-noticias';}'
38 );

```

Utilize qualquer programa de gerenciamento de base de dados MySQL para importar este script.

Se quiser utilizar o *phpMyAdmin*, já criar um tutorial pra você :

[Crie tabelas e bases de dados no phpMyAdmin - Aula 27](#)

Veja uma imagem da minha base de dados já pronta:

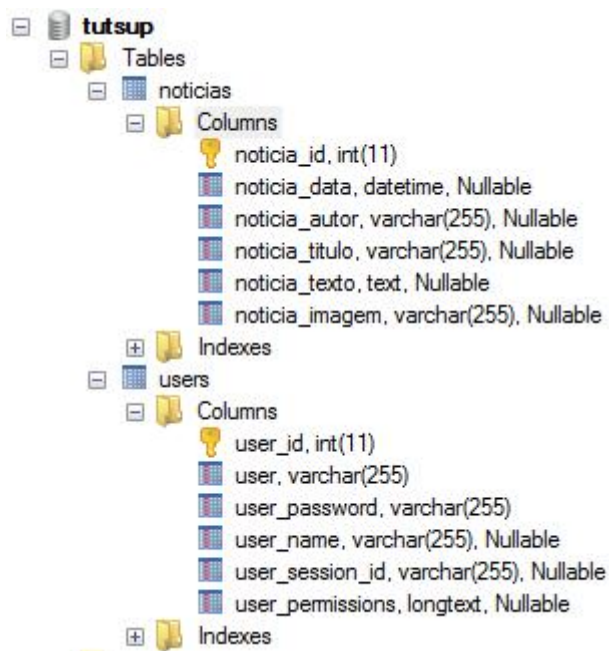


Tabela para nossa aplicação 1

## Criando o arquivo .htaccess

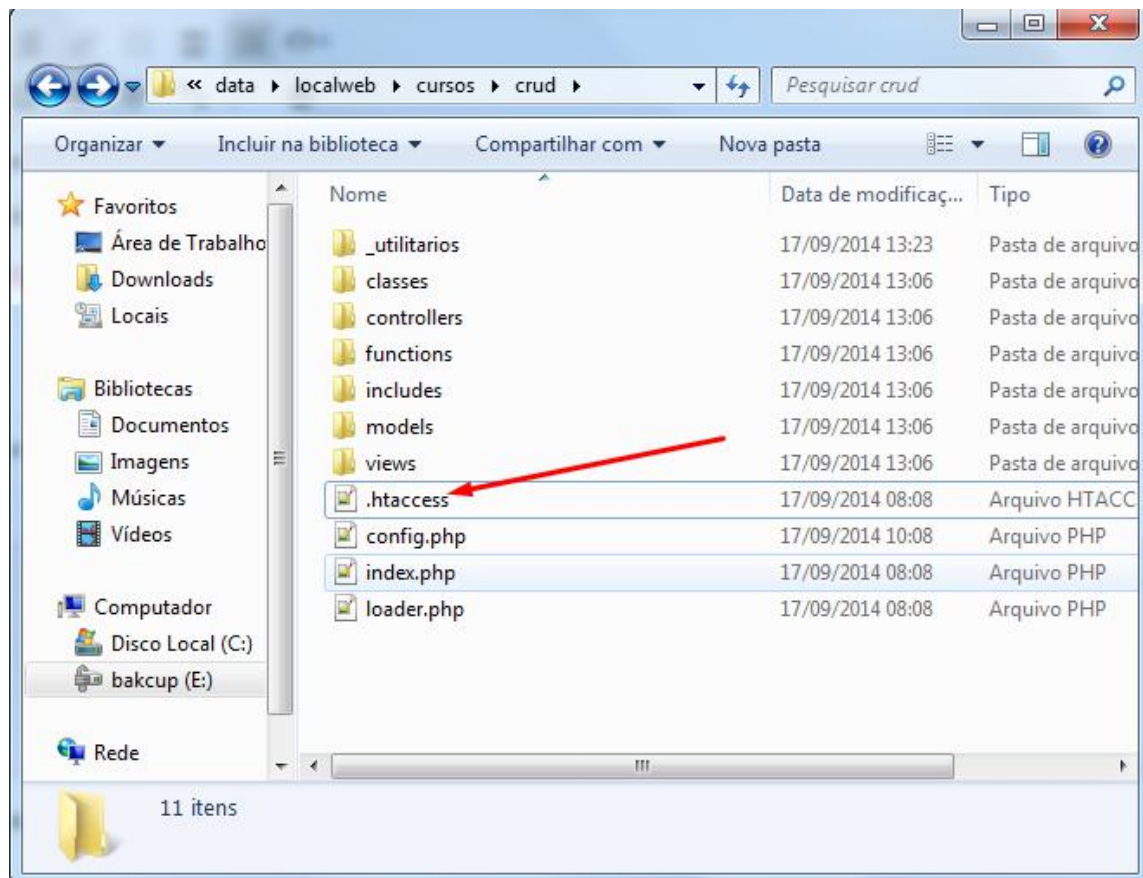
Crie uma pasta com o nome do nosso aplicativo (no meu caso “**crud**”), em seguida abra seu editor de textos preferido e crie um arquivo chamado **.htaccess**:

Nele adicione o seguinte:

```
.htaccess
1 RewriteEngine On
  RewriteCond %{REQUEST_FILENAME} !-d
2
  RewriteCond %{REQUEST_FILENAME} !-f
3
  RewriteCond %{REQUEST_FILENAME} !-l
4
  RewriteRule ^(.+)% index.php ?path=%1 [QSA,L]
```

Este arquivo deverá ficar na pasta principal da nossa aplicação. Veja um exemplo do meu sistema

(já pronto):



.htaccess 1

Este arquivo vai permitir que nossas URLs sejam escritas dessa maneira:

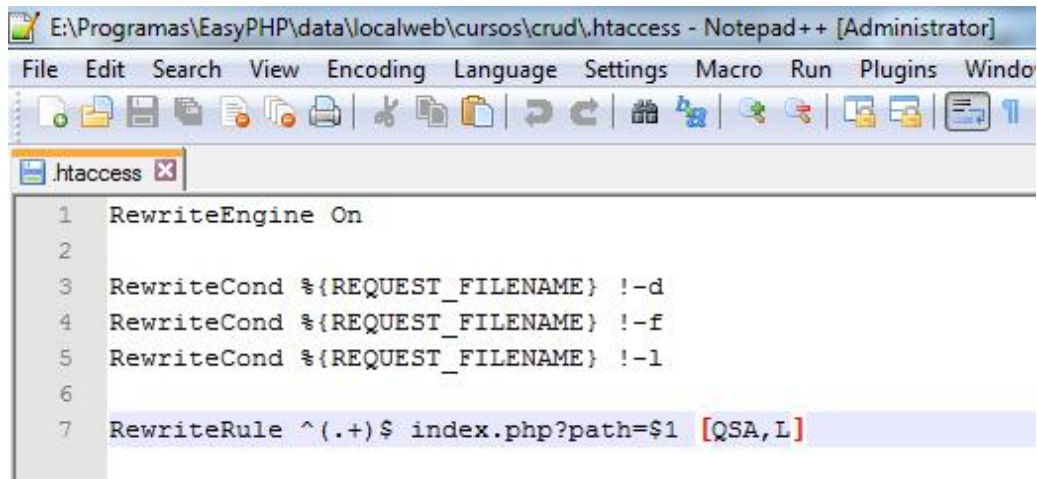
<http://www.exemplo.com/index.php?url=controlador/ação/parametro1/parametro2/etc...>

Para:

<http://www.exemplo.com/controlador/ação/parametro1/parametro2/etc...>

Se você não quiser utilizar o recurso do Apache, basta seguir o primeiro exemplo da URL.

Veja uma imagem do arquivo [.htaccess](#):

A screenshot of the Notepad++ application window. The title bar reads "E:\Programas\EasyPHP\data\localweb\cursos\crud\.htaccess - Notepad++ [Administrator]". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, and Window. The toolbar contains various icons for file operations and editing. The main text area shows the content of the .htaccess file with line numbers 1 through 7. The text is as follows:

```
1 RewriteEngine On
2
3 RewriteCond %{REQUEST_FILENAME} !-d
4 RewriteCond %{REQUEST_FILENAME} !-f
5 RewriteCond %{REQUEST_FILENAME} !-l
6
7 RewriteRule ^(.+)$ index.php?path=$1 [QSA,L]
```

.htaccess 2

## index.php

Nosso arquivo principal, o [index.php](#), ter á apenas o seguinte:

index.php

```
1 <?php
2 // Config
3 require_once 'config.php';
4 ?|
```

Mais nada...

## config.php

Nosso arquivo [config.php](#) ter á as configurações que voc ê pode alterar para cada um de seus

projetos, como configurações de URL, base de dados, debug, e assim por diante, veja:

config.php

```
1 <?php
2 /**
```

```
3  * Configuração geral
4  */
5
6  // Caminho para a raiz
7
8  define( 'ABSPATH', dirname( __FILE__ ) );
9
10 // Caminho para a pasta de uploads
11
12 define( 'UP_ABSPATH', ABSPATH . '/views/_uploads' );
13
14 // URL da home
15
16 define( 'HOME_URI', 'http://127.0.0.1/Cursos/crud' );
17
18 // Nome do host da base de dados
19
20 define( 'HOSTNAME', 'localhost' );
21
22 // Nome do DB
23
24 define( 'DB_NAME', 'tutsup' );
25
26 // Usuário do DB
27
28 define( 'DB_USER', 'root' );
29
30 // Senha do DB
31
32 define( 'DB_PASSWORD', '' );
33
34 // Charset da conexão PDO
35
36 define( 'DB_CHARSET', 'utf8' );
```

```

25 // Se voc   estiver desenvolvendo, modifique o valor para true
26
27 define( 'DEBUG', true );
28
29 /**
30  * N o edite daqui em diante
31 */
32
33 // Carrega o loader, que vai carregar a aplica  o inteira
34 require_once ABSPATH . '/loader.php';
35
36 ?!

```

Perceba que este arquivo tamb   m carrega o arquivo [loader.php](#) (veremos seu conte   do abaixo).

## loader.php

O arquivo [loader.php](#) inicia a sess o, configura os erros (dependendo da constante DEBUG) e inclui um arquivo com fun  es globais.

Veja:

loader.php

```

1 < ?php
2 // Evita que usu   rios acesse este arquivo diretamente
3 if ( ! defined( 'ABSPATH' )) exit;
4 // Inicia a sess o
5 session_start();

```



```

6 // Verifica o modo para debugar
7 if ( ! defined('DEBUG') || DEBUG === false ) {
8
9 // Esconde todos os erros
10 error_reporting(0);
11 ini_set('display_errors', 0);
12 } else {
13 // Mostra todos os erros
14 error_reporting(E_ALL);
15 ini_set('display_errors', 1);
16 }
17
18 // Funções globais
19 require_once ABSPATH . '/functions/global-functions.php';
20
21 // Carrega a aplicação
22 % tutsup_mvc = new TutsupMVC();

```

Veja que este arquivo também inicia a classe “**TutsupMVC**”, ela vai procurar o controlador e a ação. Você vai ver seu conteúdo na próxima aula.

O **loader.php** carrega o arquivo **/functions/global-functions.php**, vamos ver seu conteúdo.

## functions/global-functions.php

Este arquivo carrega duas funções muito importantes para nossa aplicação, veja:

functions/global-functions.php

```

1 < ?php
2 /**
3  * Verifica chaves de arrays
4  *
5  * Verifica se a chave existe no array e se ela tem algum valor.
6  * Obs.: Essa função está no escopo global, pois, vamos precisar muito da mesma.
7  *
8  * @param array $array O array
9  * @param string $key A chave do array
10 * @return string|null O valor da chave do array ou nulo
11 */
12 function chk_array ( $array, $key ) {
13 // Verifica se a chave existe no array
14 if ( isset( $array[ $key ] ) && !empty( $array[ $key ] ) ) {
15 // Retorna o valor da chave
16 return $array[ $key ];
17 }
18 // Retorna nulo por padrão
19 return null;
20 } // chk_array
21 /**
22 * Função para carregar automaticamente todas as classes padrão

```

```

23 * Ver: http://php.net/manual/pt\_BR/function.autoload.php.
24 * Nossas classes estão na pasta classes/.
25 * O nome do arquivo deverá ser class-NomeDaClasse.php.
26 * Por exemplo: para a classe TutsupMVC, o arquivo vai chamar class-TutsupMVC.php
27 */
28 function __autoload(% class_name) {
29     % file = ABSPATH . '/classes/class-' . % class_name . '.php';
30     if ( ! file_exists( % file ) ) {
31         require_once ABSPATH . '/includes/404.php';
32         return;
33     }
34     // Inclui o arquivo da classe
35     require_once % file;
36 } // __autoload

```

As ações das funções estão descritas no código acima.

## Utilizando estrutura MVC em PHP - Parte 2

Continuando com o desenvolvimento da nossa aplicação com estrutura MVC em PHP, hoje você vai entender como funciona a classe [TutsupMVC](#), que está presente na pasta [classes](#), em um arquivo chamado [class-TutsupMVC.php](#).

É uma classe relativamente pequena, por é muito importante. Ela carrega o controlador e executa a ação que irá incluir o model e view, formando assim a estrutura MVC. Se ela não funcionar corretamente, toda nossa aplicação irá ter problemas, por isso é muito importante que você saiba como ela funciona.

Além disso, também vamos falar sobre todas as classes que [TutsupMVC](#) carrega, gerando um desencadeamento de classes que serão carregadas até gerar um view para o usuário.

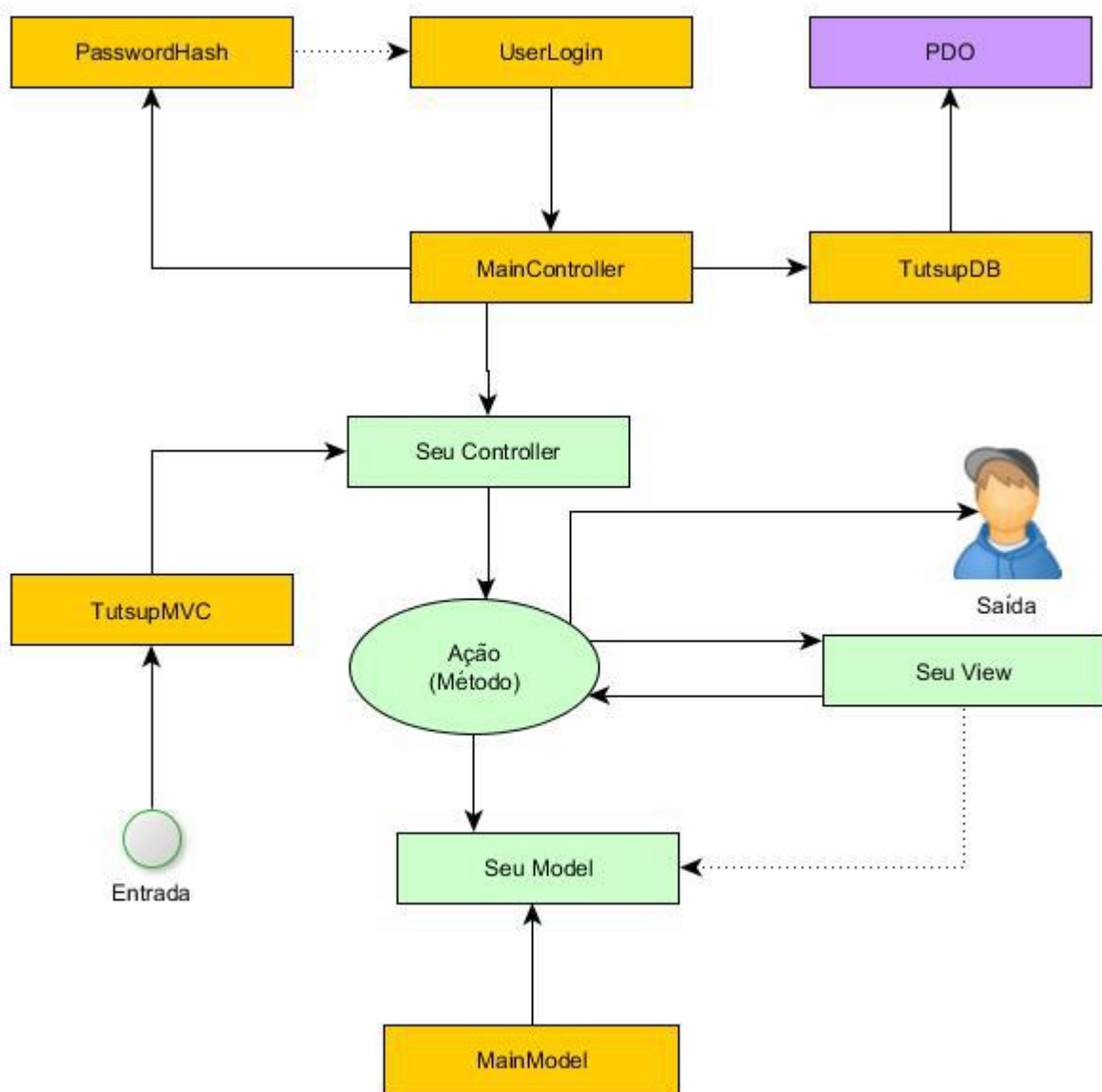
Lembre-se que este artigo é parte de uma série de artigos sobre [MVC em PHP](#), não deixe de ler a aula anterior:

[Utilizando estrutura MVC em PHP - Parte 1](#)

Então vamos lá !

## Como a classe TutsupMVC funciona ?

Antes de vermos códigos, vamos analisar um desenho simples mostrando como a classe funciona:



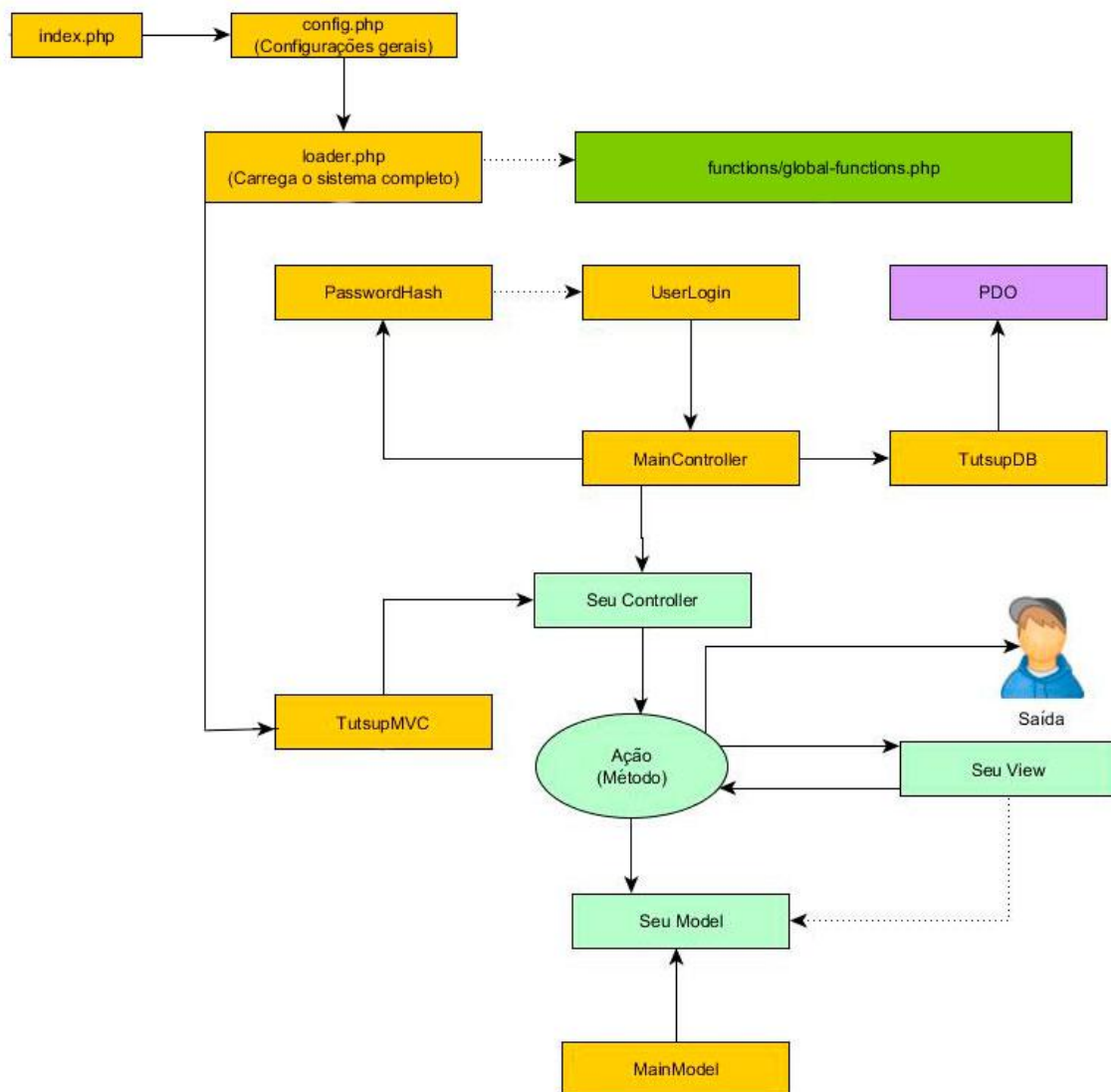
## Classe TutsupMVC e o fluxo da informação 1

Na imagem acima, as classes que estão em amarelo são parte da estrutura. A

classe **TutsupMVC** desencadeia uma série de outras classes que fazem parte de um quebra-cabeça para fazer a aplicação funcionar conforme pretendemos.

O controlador personalizado terá todos os métodos e propriedades presentes em todas essas classes, bem como as funcionalidades, como verificação se o usuário está ou não logado, permissões, conexão PDO, métodos para criar, apagar, atualizar e editar dados da base de dados, e qualquer coisa que criarmos posteriormente.

Lembra da imagem que apresentei no [último artigo sobre MVC em PHP](#) ? Veja agora uma junção de ambas:



## Modelo MVC em PHP 1

Com os modelos acima em mente, veja alguns detalhes importantes sobre as classes:

1. TutsupMVC só vai tentar carregar seu controller se isso for especificado na URL. Por exemplo: *exemplo.com/seu-controller/*. Caso contrário, o controlador “home” será incluído e a ação `index()` será executada;

2. Se nenhuma ação for especificada na URL (exemplo.com/seu-controller/acao/), TutsupMVC vai tentar carregar a ação index(); se a ação index() não existir, a página de erro 404.php é incluída;
3. Seu Controller deverá estender a classe “MainController” ;
4. Seu Model deverá estender a classe “MainModel” ;
5. Seu Model estará disponível dentro do seu view por um objeto que você mesmo vai criar;

Agora vamos ver um pouco de código PHP.

## classes/class-TutsupMVC.php

Então vamos ver o que realmente interessa, o código.

Caso queira baixar o que já criamos até agora para acompanhar, segue o link:

Download [crud-1-0.zip](#)

A classe encontra-se em: [classes/class-TutsupMVC.php](#) e contém o seguinte:

```
<?php
```

```
/**
```

```
 * TutsupMVC - Gerencia Models, Controllers e Views
```

```
 *
```

```
 * @package TutsupMVC
```

```
 * @since 0.1
```

```
 */
```

```
class TutsupMVC
```

```
{
```

```
/**
```

```
* % controlador
```

```
*
```

```
* Receber á o valor do controlador (Vindo da URL).
```

```
* exemplo.com/controlador/
```

```
*
```

```
* @access private
```

```
*/
```

```
private % controlador;
```

```
/**
```

```
* % acao
```

```
*
```

```
* Receber á o valor da ação (Também vem da URL):
```

```
* exemplo.com/controlador/acao
```

```
*
```

```
* @access private
```

```
*/
```

```
private % acao;
```

```
/**
```

```
* % parametros
```

```
*
```

```
* Receber á um array dos parâmetros (Também vem da URL):
```



```
* exemplo.com/controlador/acao/param1/param2/param50
```

```
*
```

```
* @access private
```

```
*/
```

```
private % parametros;
```

```
/**
```

```
* % not_found
```

```
*
```

```
* Caminho da página não encontrada
```

```
*
```

```
* @access private
```

```
*/
```

```
private % not_found = '/includes/404.php';
```

```
/**
```

```
* Construtor para essa classe
```

```
*
```

```
* Obtém os valores do controlador, ação e parâmetros. Configura
```

```
* o controlado e a ação (m é todo).
```

```
*/
```

```
public function __construct () {
```

```
// Obtém os valores do controlador, ação e parâmetros da URL.
```

```
// E configura as propriedades da classe.
```

```
% this->get_url_data();
```

```
/**
```

\* Verifica se o controlador existe. Caso contrário, adiciona o

\* controlador padrão (controllers/home-controller.php) e chama o método index().

```
*/
```

```
if ( ! $this->controlador ) {
```

```
// Adiciona o controlador padrão
```

```
require_once ABSPATH . 'controllers/home-controller.php';
```

```
// Cria o objeto do controlador 'home-controller.php'
```

```
// Este controlador deverá ter uma classe chamada HomeController
```

```
% this->controlador = new HomeController();
```

```
// Executa o método index()
```

```
% this->controlador->index();
```

```
// FIM :)
```

```
return;
```

```
}
```

```
// Se o arquivo do controlador não existir, não faremos nada
```

```
if ( ! file_exists( ABSPATH . 'controllers/' . $this->controlador . '.php' ) ) {
```

```
// Página não encontrada
```

```
require_once ABSPATH . $this->not_found;
```

```
// FIM :)
```

```
return;
```

```
}
```

```
// Inclui o arquivo do controlador
```

```
require_once ABSPATH . '/controllers/' . % this-lcontrolador . '.php';
```

```
// Remove caracteres inválidos do nome do controlador para gerar o nome
```

```
// da classe. Se o arquivo chamar 'news-controller.php', a classe deverá
```

```
// se chamar NewsController.
```

```
% this-lcontrolador = preg_replace( '/[a-zA-Z]/i', '', % this-lcontrolador );
```

```
// Se a classe do controlador indicado não existir, não faremos nada
```

```
if ( ! class_exists( % this-lcontrolador ) ) {
```

```
// Página não encontrada
```

```
require_once ABSPATH . % this-lnot_found;
```

```
// FIM ;)
```

```
return;
```

```
} // class_exists
```

```
// Cria o objeto da classe do controlador e envia os parâmetros
```

```
% this-lcontrolador = new % this-lcontrolador( % this-lparametros );
```

```
// Se o método indicado existir, executa o método e envia os parâmetros
```

```
if ( method_exists( % this-lcontrolador, % this-lacao ) ) {
```

```
% this-lcontrolador-l{% this-lacao }( % this-lparametros );
```

```
// FIM ;)
```

```
return;
```

```
} // method_exists
```

```
// Sem ação, chamamos o m é todo index
```

```
if ( ! % this-lacao && method_exists( % this-controlador, 'index' ) ) {
```

```
% this-controlador-index( % this-parametros );
```

```
// FIM ;)
```

```
return;
```

```
} // ! % this-lacao
```

```
// P á gina não encontrada
```

```
require_once ABSPATH . % this-not_found;
```

```
// FIM ;)
```

```
return;
```

```
} // __construct
```

```
/**
```

```
* Obt é m parâmetros de % _GET['path']
```

```
*
```

```
* Obt é m os parâmetros de % _GET['path'] e configura as propriedades
```

```
* % this-controlador, % this-lacao e % this-parametros
```

```
*
```

```
* A URL dever á ter o seguinte formato:
```

```
* http://www.example.com/controlador/acao/parametro1/parametro2/etc...
```

```
*/
```

```
public function get_url_data ( ) {
```

```
// Verifica se o parâmetro path foi enviado
```

```
if ( isset( $_GET['path'] ) ) {
```

```
// Captura o valor de $_GET['path']
```

```
$_path = $_GET['path'];
```

```
// Limpa os dados
```

```
$_path = rtrim($_path, '/');
```

```
$_path = filter_var($_path, FILTER_SANITIZE_URL);
```

```
// Cria um array de parâmetros
```

```
$_path = explode('/', $_path);
```

```
// Configura as propriedades
```

```
$_this->controlador = chk_array( $_path, 0 );
```

```
$_this->controlador .= '-controller';
```

```
$_this->acao = chk_array( $_path, 1 );
```

```
// Configura os parâmetros
```

```
if ( chk_array( $_path, 2 ) ) {
```

```
unset( $_path[0] );
```

```
unset( $_path[1] );
```

```
// Os parâmetros sempre virão após a ação
```

```
$_this->parametros = array_values( $_path );
```

```
}
```

```
// DEBUG
```

```
//
```

```
// echo $_this->controlador . '<br|';
```

```
// echo % this-lacao . '<br|';

// echo '<pre|';

// print_r( % this-lparametros );

// echo '</pre|';
}

} // get_url_data

} // class TutsupMVC
```

É bastante c ó d i g o , mas vamos ver o que cada parte faz:

1. O construtor da classe carrega o m é todo `get_url_data`;
2. `get_url_data` obt é m os parâmetros necess á rios da URL;
3. O primeiro parâmetro é o controlador que voc ê dever á criar. Os controladores “home” , “login” , “user-register” e “noticias” j á existem e v ê m embutidos no pacote que voc ê baixou. Eles servem apenas como exemplo, j á que voc ê poder á criar seus pr ó prios models, controllers e views; Se voc ê não indicar nenhum controlador, a ação `index()` do controlador “home” ser á executada;
4. O segundo parâmetro é a ação. Por ação, me refiro a um m é todo que existe dentro do controller para carregar o(s) model(s) e view(s). Se voc ê não indicar uma ação, a classe ir á buscar a ação `index()` dentro do seu controlador; Se essa ação não existir, a p á gina de erro 404 é inclu í da (p á gina não encontrada). Voc ê pode personalizar essa p á gina conforme preferir;
5. Do terceiro parâmetro em diante, a classe ir á considerar os valores como parâmetros, e enviar á os dados para o seu controlador. Voc ê pode manipular os dados conforme preferir;

6. Tudo isso é executado no [construtor](#) da nossa classe.

Vamos ver um exemplo do controlador “home-controller.php”, que está dentro da pasta [controllers](#).

## controllers/home-controller.php

Este é o controlador padrão que é carregado quando nenhum controller é enviado para a URL.

Por padrão, ele não tem nenhum model, apenas views.

Veja seu código:

```
<?php
```

```
/**
```

```
 * home - Controller de exemplo
```

```
 *
```

```
 * @package TutsupMVC
```

```
 * @since 0.1
```

```
 */
```

```
class HomeController extends MainController
```

```
{
```

```
/**
```

```
 * Carrega a página ‘/views/home/home-view.php’
```

```
 */
```

```
    public function index() {
```

```
        // Título da página
```

```

% this->title = 'Home';

// Parametros da função

% parametros = ( func_num_args() != 1 ) ? func_get_arg(0) : array();

// Essa página não precisa de modelo (model)

/** Carrega os arquivos do view */

// /views/_includes/header.php

    require ABSPATH . '/views/_includes/header.php';

// /views/_includes/menu.php

    require ABSPATH . '/views/_includes/menu.php';

// /views/home/home-view.php

    require ABSPATH . '/views/home/home-view.php';

// /views/_includes/footer.php

    require ABSPATH . '/views/_includes/footer.php';

} // index

} // class HomeController

```

Perceba que ele tem apenas uma ação (um m é todo), chamado de `index()`.

Este m é todo apenas inclui os views que preciso para exibir o conteúdo na tela, como:

/views/\_includes/header.php - Cabeçalho HTML

/views/\_includes/menu.php - Menu HTML

/views/home/home-view.php - Conteúdo HTML

/views/\_includes/footer.php - Rodapé HTML



Você também pode incluir esses arquivos diretamente no seu view, neste caso você só precisaria incluir o view padrão (home-view.php) e o restante seria feito tudo dentro desse arquivo.

Existem mais coisas que você pode configurar, por exemplo:

```
// Título da página

% this->title = 'Home';

// Parametros da função

% $parametros = ( func_num_args() != 1 ) ? func_get_arg(0) : array();
```

No trecho acima, estamos configurando o título da página e recebendo os parâmetros que a classe TutsupMVC nos enviou.

Ainda vamos falar sobre controllers mais avançados, com permissão de usuário, e coisas do tipo, mas vamos deixar assim por enquanto.

Como a nossa classe HomeController estende a classe MainController, vamos ver o conteúdo dessa classe.

## classes/class-MainController.php

Este é nosso controller padrão, todos os outros controllers irão estender essa classe. Veja seu código:

```
<?php

/**

 * MainController - Todos os controllers deverão estender essa classe

 */
```

```
* @package TutsupMVC
```

```
* @since 0.1
```

```
*/
```

```
class MainController extends UserLogin
```

```
{
```

```
/**
```

```
* % db
```

```
*
```

```
* Nossa conexão com a base de dados. Manter á o objeto PDO
```

```
*
```

```
* @access public
```

```
*/
```

```
public % db;
```

```
/**
```

```
* % phpass
```

```
*
```

```
* Classe phpass
```

```
*
```

```
* @see http://www.openwall.com/phpass/
```

```
* @access public
```

```
*/
```

```
public % phpass;
```

```
/**
```

```
 * % title
```

```
 *
```

```
 * T í tulo das p á ginas
```

```
 *
```

```
 * @access public
```

```
 */
```

```
public % title;
```

```
/**
```

```
 * % login_required
```

```
 *
```

```
 * Se a p á gina precisa de login
```

```
 *
```

```
 * @access public
```

```
 */
```

```
public % login_required = false;
```

```
/**
```

```
 * % permission_required
```

```
 *
```

```
 * Permissão necess á ria
```

```
 *
```

```
* @access public
```

```
*/
```

```
public % permission_required = 'any';
```

```
/**
```

```
* % parametros
```

```
*
```

```
* @access public
```

```
*/
```

```
public % parametros = array();
```

```
/**
```

```
* Construtor da classe
```

```
*
```

```
* Configura as propriedades e m é todos da classe.
```

```
*
```

```
* @since 0.1
```

```
* @access public
```

```
*/
```

```
public function __construct ( % parametros = array() ) {
```

```
// Instancia do DB
```

```
% this->db = new TutsupDB();
```

```
// Phpass
```

```
% this->phpass = new PasswordHash(8, false);
```

```
// Parâmetros
```

```
% this->parametros = % parametros;
```

```
// Verifica o login
```

```
% this->check_userlogin();
```

```
} // __construct
```

```
/**
```

```
* Load model
```

```
*
```

```
* Carrega os modelos presentes na pasta /models/.
```

```
*
```

```
* @since 0.1
```

```
* @access public
```

```
*/
```

```
public function load_model( % model_name = false ) {
```

```
// Um arquivo deverá ser enviado
```

```
if ( ! % model_name ) return;
```

```
// Garante que o nome do modelo tenha letras minúsculas
```

```
% model_name = strtolower( % model_name );
```

```
// Inclui o arquivo
```

```
% model_path = ABSPATH . '/models/' . % model_name . '.php';
```

```
// Verifica se o arquivo existe
```

```
if ( file_exists( % model_path ) ) {
```

```

// Inclui o arquivo

require_once %model_path;

// Remove os caminhos do arquivo (se tiver algum)

%model_name = explode('/', %model_name);

// Pega só o nome final do caminho

%model_name = end( %model_name );

// Remove caracteres inválidos do nome do arquivo

%model_name = preg_replace( '/[a-zA-Z0-9]/is', '', %model_name );

// Verifica se a classe existe

if ( class_exists( %model_name ) ) {

// Retorna um objeto da classe

return new %model_name( %this->db, %this );

}

// The end :)

return;

} // load_model

} // load_model

} // class MainController

```

Nosso controlador principal configura tudo o que precisamos em seu construtor, veja essa parte do código:

```

// Instancia do DB

%this->db = new TutsupDB();

```

```
// Phpass

% this->phpass = new PasswordHash(8, false);

// Parâmetros

% this->parametros = % parametros;

// Verifica o login

% this->check_userlogin();
```

Aqui configuramos nossa conexão com a base de dados, criamos o objeto da classe Phpass, configuramos os parâmetros e acionamos o método que verifica se o usuário estão ou não logado. Além disso, temos um outro método que carrega os modelos. Vamos falar dele posteriormente. Perceba que nossa classe MainController estende outra classe chamada de UserLogin, que é responsável por verificar tudo do usuário, como fazer login, logout, verificar se ele está logado e coisas do tipo.

Vamos ver o conteúdo da classe [UserLogin](#).

## classes/class-UserLogin.php

A classe UserLogin é responsável por tudo o que é relacionado ao usuário em si, veja:

```
<?php

/**

 * UserLogin - Manipula os dados de usuários

 *

 * Manipula os dados de usuários, faz login e logout, verifica permissões e
```

```
* redireciona p á gina para usu á rios logados.
```

```
*
```

```
* @package TutsupMVC
```

```
* @since 0.1
```

```
*/
```

```
class UserLogin
```

```
{
```

```
/**
```

```
* Usu á rio logado ou não
```

```
*
```

```
* Verdadeiro se ele estiver logado.
```

```
*
```

```
* @public
```

```
* @access public
```

```
* @var bol
```

```
*/
```

```
public % logged_in;
```

```
/**
```

```
* Dados do usu á rio
```

```
*
```

```
* @public
```

```
* @access public
```

```
* @var array
```



```
*/
```

```
public % userdata;
```

```
/**
```

```
* Mensagem de erro para o formulário de login
```

```
*
```

```
* @public
```

```
* @access public
```

```
* @var string
```

```
*/
```

```
public % login_error;
```

```
/**
```

```
* Verifica o login
```

```
*
```

```
* Configura as propriedades % logged_in e % login_error. Também
```

```
* configura o array do usuário em % userdata
```

```
*/
```

```
public function check_userlogin () {
```

```
// Verifica se existe uma sessão com a chave userdata
```

```
// Tem que ser um array e não pode ser HTTP POST
```

```
if ( isset( $_SESSION['userdata'] ) )
```

```
&& ! empty( $_SESSION['userdata'] )
```

```
&& is_array( $_SESSION['userdata'] )
```

```
&& !isset( $_POST['userdata'] )
```

```
) {
```

```
// Configura os dados do usuário
```

```
$_SESSION['userdata'];
```

```
// Garante que não é HTTP POST
```

```
$_SESSION['post'] = false;
```

```
}
```

```
// Verifica se existe um $_POST com a chave userdata
```

```
// Tem que ser um array
```

```
if ( isset( $_POST['userdata'] )
```

```
&& !empty( $_POST['userdata'] )
```

```
&& is_array( $_POST['userdata'] )
```

```
) {
```

```
// Configura os dados do usuário
```

```
$_SESSION['userdata'];
```

```
// Garante que é HTTP POST
```

```
$_SESSION['post'] = true;
```

```
}
```

```
// Verifica se existe algum dado de usuário para conferir
```

```
if ( !isset( $_SESSION ) || !is_array( $_SESSION ) {
```

```
// Remove qualquer sessão que possa existir sobre o usuário
```

```
$_SESSION = null;
```

```
return;
```

```
}
```

```
// Passa os dados do post para uma variável
```

```
if ( % userdata['post'] === true ) {
```

```
% post = true;
```

```
} else {
```

```
% post = false;
```

```
}
```

```
// Remove a chave post do array userdata
```

```
unset( % userdata['post'] );
```

```
// Verifica se existe algo a conferir
```

```
if ( empty( % userdata ) ) {
```

```
% this->logged_in = false;
```

```
% this->login_error = null;
```

```
// Remove qualquer sessão que possa existir sobre o usuário
```

```
% this->logout();
```

```
return;
```

```
}
```

```
// Extraí variáveis dos dados do usuário
```

```
extract( % userdata );
```

```
// Verifica se existe um usuário e senha
```

```
if ( ! isset( % user ) || ! isset( % user_password ) ) {
```

```
% this->logged_in = false;
```

```
% this->login_error = null;
```

```
// Remove qualquer sessão que possa existir sobre o usuário
```

```
% this->logout();
```

```
return;
```

```
}
```

```
// Verifica se o usuário existe na base de dados
```

```
% query = % this->db->query(
```

```
'SELECT * FROM users WHERE user = ? LIMIT 1',
```

```
array( % user )
```

```
);
```

```
// Verifica a consulta
```

```
if ( ! % query ) {
```

```
% this->logged_in = false;
```

```
% this->login_error = 'Internal error.';
```

```
// Remove qualquer sessão que possa existir sobre o usuário
```

```
% this->logout();
```

```
return;
```

```
}
```

```
// Obtém os dados da base de usuário
```

```
% fetch = % query->fetch(PDO::FETCH_ASSOC);
```

```
// Obtém o ID do usuário
```

```
% user_id = (int) % fetch['user_id'];
```

```
// Verifica se o ID existe
```

```
if ( empty( % user_id ) ){
```

```
% this->logged_in = false;
```

```
% this->login_error = 'User do not exists.';
```

```
// Remove qualquer sessão que possa existir sobre o usuário
```

```
% this->logout();
```

```
return;
```

```
}
```

```
// Confere se a senha enviada pelo usuário bate com o hash do BD
```

```
if ( % this->phpass->CheckPassword( % user_password, % fetch['user_password'] ) ) {
```

```
// Se for uma sessão, verifica se a sessão bate com a sessão do BD
```

```
if ( session_id() != % fetch['user_session_id'] && ! % post ) {
```

```
% this->logged_in = false;
```

```
% this->login_error = 'Wrong session ID.';
```

```
// Remove qualquer sessão que possa existir sobre o usuário
```

```
% this->logout();
```

```
return;
```

```
}
```

```
// Se for um post
```

```
if ( % post ) {
```

```
// Recria o ID da sessão
```

```
session_regenerate_id();
```

```
% session_id = session_id();
```

```
// Envia os dados de usuário para a sessão
```

```
% $_SESSION['userdata'] = % fetch;
```

```
// Atualiza a senha
```

```
% _SESSION['userdata']['user_password'] = % user_password;
```

```
// Atualiza o ID da sessão
```

```
% _SESSION['userdata']['user_session_id'] = % session_id;
```

```
// Atualiza o ID da sessão na base de dados
```

```
% query = % this->db->query(
```

```
'UPDATE users SET user_session_id = ? WHERE user_id = ?',
```

```
array( % session_id, % user_id )
```

```
);
```

```
}
```

```
// Obtém um array com as permissões de usuário
```

```
% _SESSION['userdata']['user_permissions'] = unserialize( % fetch['user_permissions'] );
```

```
// Configura a propriedade dizendo que o usuário está logado
```

```
% this->logged_in = true;
```

```
// Configura os dados do usuário para % this->userdata
```

```
% this->userdata = % _SESSION['userdata'];
```

```
// Verifica se existe uma URL para redirecionar o usuário
```

```
if ( isset( % _SESSION['goto_url'] ) ) {
```

```
// Passa a URL para uma variável
```

```
% goto_url = urldecode( % _SESSION['goto_url'] );
```

```
// Remove a sessão com a URL
```

```
unset( % _SESSION['goto_url'] );
```

```
// Redireciona para a página
```

```
echo '<meta http-equiv='Refresh' content='0; url=' . % goto_url . '*'>';
```

```
echo '<script type='text/javascript' |window.location.href = '' . % goto_url . *;</script>';
```

```
//header( 'location: ' . % goto_url );
```

```
}
```

```
return;
```

```
} else {
```

```
// O usuário não está logado
```

```
% this->logged_in = false;
```

```
// A senha não bateu
```

```
% this->login_error = 'Password does not match.';
```

```
// Remove tudo
```

```
% this->logout();
```

```
return;
```

```
}
```

```
}
```

```
/**
```

```
* Logout
```

```
*
```

```
* Remove tudo do usuário.
```

```
*
```

```
* @param bool % redirect Se verdadeiro, redireciona para a página de login
```

```
* @final
```

```
*/
```

```
protected function logout( % redirect = false ) {
```

```
// Remove all data from % _SESSION['userdata']
```

```
% _SESSION['userdata'] = array();
```

```
// Only to make sure (it isn't really needed)
```

```
unset( % _SESSION['userdata'] );
```

```
// Regenerates the session ID
```

```
session_regenerate_id();
```

```
if ( % redirect === true ) {
```

```
// Send the user to the login page
```

```
% this->goto_login();
```

```
}
```

```
}
```

```
/**
```

```
* Vai para a página de login
```

```
*/
```

```
protected function goto_login() {
```

```
// Verifica se a URL da HOME está configurada
```

```
if ( defined( 'HOME_URI' ) ) {
```

```
// Configura a URL de login
```

```
% login_uri = HOME_URI . '/login/';
```

```
// A página em que o usuário estava
```

```
% _SESSION['goto_url'] = urlencode( % _SERVER['REQUEST_URI'] );
```

```
// Redireciona
```

```
echo '<meta http-equiv="Refresh" content="0; url=' . % login_uri . '"';
```



```
echo '<script type="text/javascript"|window.location.href = "" . % login_uri . *;</script|';
```

```
// header( 'location: ' . % login_uri );
```

```
}
```

```
return;
```

```
}
```

```
/**
```

```
* Envia para uma página qualquer
```

```
*
```

```
* @final
```

```
*/
```

```
final protected function goto_page( % page_uri = null ) {
```

```
if ( isset( % _GET['url'] ) && ! empty( % _GET['url'] ) && ! % page_uri ) {
```

```
// Configura a URL
```

```
% page_uri = urldecode( % _GET['url'] );
```

```
}
```

```
if ( % page_uri ) {
```

```
// Redireciona
```

```
echo '<meta http-equiv="Refresh" content="0; url=" . % page_uri . *|';
```

```
echo '<script type="text/javascript"|window.location.href = "" . % page_uri . *;</script|';
```

```
//header( 'location: ' . % page_uri );
```

```
return;
```

```
}
```

```
}
```

```
/**
```

```
* Verifica permissões
```

```

*

* @param string % required A permissão requerida

* @param array % user_permissions As permissões do usuário

* @final

*/

final protected function check_permissions(

% required = 'any',

% user_permissions = array('any')

) {

if ( !is_array( % user_permissions ) ) {

return;

}

// Se o usuário não tiver permissão

if ( !in_array( % required, % user_permissions ) ) {

// Retorna falso

return false;

} else {

return true;

}

}

}

}

```

O arquivo [global-functions.php](#) que está dentro da pasta [functions](#) tem a função de [\\_\\_autoload](#), que carregará todas as nossas classes automaticamente.

Para isso, a classe deverá estar dentro da pasta [classes](#) ter o nome [class-NomeDaClasse.php](#) (Com letras maiúsculas e minúsculas).

## Utilizando estrutura MVC em PHP – Parte 3

Outra parte muito importante da nossa aplicação com estrutura MVC em PHP é a base de dados.

A maioria dos sistemas que iremos criar terá acesso ao banco de dados, tanto para salvar dados relevantes como para manipular os dados de usuários.

Para nosso tutorial, criei uma classe que gerencia tudo o que é relacionado à base de dados, como conexão e manipulação de dados.

A classe responsável por manter a base de dados é a “TutsupDB”, que está na pasta “classes”, com o nome “class-TutsupDB.php”.

Mais adiante neste artigo vamos analisar seu código, bem como demais classes que formam nosso formato de [MVC em PHP](#).

## Download

Como estamos criando uma aplicação inteira, são vários arquivos que se encaixam uns nos outros.

Se você quiser baixar os dados e ir acompanhando com eles em seu computador, segue o link abaixo:

Download [crud-1-0.zip](#)

Caso queira contribuir com o projeto, acesso no Github:

[luizomf/TutsupMVC](#)

No download acima, a aplicação está praticamente completa. Pode ser que algo seja alterado conforme nossa necessidade, mas vou detalhar tudo no artigo.

## classes/class-TutsupDB.php

A classe TutsupDB faz a conexão PDO e gerencia dados da base de dados, veja seu código:

```
<?php

/**
 * TutsupDB - Classe para gerenciamento da base de dados
 *
 * @package TutsupMVC
 * @since 0.1
 */

class TutsupDB
{
    /** DB properties */

    public $host      = 'localhost', // Host da base de dados

    $db_name  = 'tutsup', // Nome do banco de dados

    $password = '',      // Senha do usuário da base de dados

    $user     = 'root',  // Usuário da base de dados

    $charset  = 'utf8',  // Charset da base de dados

    $pdo      = null,    // Nossa conexão com o BD

    $error     = null,   // Configura o erro

    $debug     = false,  // Mostra todos os erros
```

```
% last_id = null; // Último ID inserido
```

```
/**
```

```
* Construtor da classe
```

```
*
```

```
* @since 0.1
```

```
* @access public
```

```
* @param string % host
```

```
* @param string % db_name
```

```
* @param string % password
```

```
* @param string % user
```

```
* @param string % charset
```

```
* @param string % debug
```

```
*/
```

```
public function __construct(
```

```
% host = null,
```

```
% db_name = null,
```

```
% password = null,
```

```
% user = null,
```

```
% charset = null,
```

```
% debug = null
```

```
) {
```

```
// Configura as propriedades novamente.
```

// Se voc ê fez isso no in í cio dessa classe, as constantes não serão

// necess á rias. Voc ê escolhe...

```
% this->host = defined( 'HOSTNAME' ) ? HOSTNAME : % this->host;
```

```
% this->db_name = defined( 'DB_NAME' ) ? DB_NAME : % this->db_name;
```

```
% this->password = defined( 'DB_PASSWORD' ) ? DB_PASSWORD : % this->password;
```

```
% this->user = defined( 'DB_USER' ) ? DB_USER : % this->user;
```

```
% this->charset = defined( 'DB_CHARSET' ) ? DB_CHARSET : % this->charset;
```

```
% this->debug = defined( 'DEBUG' ) ? DEBUG : % this->debug;
```

// Conecta

```
% this->connect();
```

```
} // __construct
```

```
/**
```

```
* Cria a conexão PDO
```

```
*
```

```
* @since 0.1
```

```
* @final
```

```
* @access protected
```

```
*/
```

```
final protected function connect() {
```

```
/* Os detalhes da nossa conexão PDO */
```

```
% pdo_details = 'mysql:host={% this->host}';
```

```
% pdo_details .= 'dbname={% this->db_name}';
```

```
% pdo_details .= 'charset={% this->charset}';
```

```
// Tenta conectar
```

```
try {
```

```
% this->pdo = new PDO(% pdo_details, % this->user, % this->password);
```

```
// Verifica se devemos debugar
```

```
if ( % this->debug === true ) {
```

```
// Configura o PDO ERROR MODE
```

```
% this->pdo->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING );
```

```
}
```

```
// Não precisamos mais dessas propriedades
```

```
unset( % this->host );
```

```
unset( % this->db_name );
```

```
unset( % this->password );
```

```
unset( % this->user );
```

```
unset( % this->charset );
```

```
} catch (PDOException % e) {
```

```
// Verifica se devemos debugar
```

```
if ( % this->debug === true ) {
```

```
// Mostra a mensagem de erro
```

```
echo 'Erro: ' . % e->getMessage();
```

```
}
```

```
// Kills the script
```

```
die();
```

```
} // catch
```

```
} // connect
```

```
/**
```

```
* query – Consulta PDO
```

```
*
```

```
* @since 0.1
```

```
* @access public
```

```
* @return object|bool Retorna a consulta ou falso
```

```
*/
```

```
public function query( % stmt, % data_array = null ) {
```

```
// Prepara e executa
```

```
% query    = % this->pdo->prepare( % stmt );
```

```
% check_exec = % query->execute( % data_array );
```

```
// Verifica se a consulta aconteceu
```

```
if ( % check_exec ) {
```

```
// Retorna a consulta
```

```
return % query;
```

```
} else {
```

```
// Configura o erro
```

```
% error    = % query->errorInfo();
```

```
% this->error = % error[2];
```

```
// Retorna falso
```

```
return false;
```

```
}
```



```
}
```

```
/**
```

```
* insert - Insere valores
```

```
*
```

```
* Insere os valores e tenta retornar o último id enviado
```

```
*
```

```
* @since 0.1
```

```
* @access public
```

```
* @param string %table O nome da tabela
```

```
* @param array ... Ilimitado número de arrays com chaves e valores
```

```
* @return object|bool Retorna a consulta ou falso
```

```
*/
```

```
public function insert( %table ) {
```

```
// Configura o array de colunas
```

```
%cols = array();
```

```
// Configura o valor inicial do modelo
```

```
%placeholders = ' ';
```

```
// Configura o array de valores
```

```
%values = array();
```

```
// O %j will assegurar que colunas serão configuradas apenas uma vez
```

```
%j = 1;
```

```
// Obtém os argumentos enviados
```

```
%data = func_get_args();
```

```
// É preciso enviar pelo menos um array de chaves e valores
```

```
if ( !isset( %data[1] ) || !is_array( %data[1] ) ) {
```

```
return;
```

```
}
```

```
// Faz um laço nos argumentos
```

```
for ( %i = 1; %i < count( %data ); %i++ ) {
```

```
// Obtém as chaves como colunas e valores como valores
```

```
foreach ( %data[%i] as %col = %val ) {
```

```
// A primeira volta do laço configura as colunas
```

```
if ( %i == 1 ) {
```

```
%cols[] = '%col';
```

```
}
```

```
if ( %j < %i ) {
```

```
// Configura os divisores
```

```
%placeholders .= '), (';
```

```
}
```

```
// Configura os place holders do PDO
```

```
%placeholders .= '?, ';
```

```
// Configura os valores que vamos enviar
```

```
%values[] = %val;
```

```
%j = %i;
```

```
}
```

```
// Remove os caracteres extra dos place holders
```

```
%placeholders = substr( %placeholders, 0, strlen( %placeholders ) - 2 );
```

```

}

// Separa as colunas por v í rgula

% cols = implode( ', ', % cols );

// Cria a declaração para enviar ao PDO

% stmt = 'INSERT INTO ` % table ` ( % cols ) VALUES % place_holders ' ;

// Inere os valores

% insert = % this->query( % stmt, % values );

// Verifica se a consulta foi realizada com sucesso

if ( % insert ) {

// Verifica se temos o ú ltimo ID enviado

if ( method_exists( % this->pdo, 'lastInsertId' )

&& % this->pdo->lastInsertId( )

) {

// Configura o ú ltimo ID

% this->last_id = % this->pdo->lastInsertId( );

}

// Retorna a consulta

return % insert;

}

// The end :)

return;

} // insert

/**

* Update simples

```

```
*
```

```
* Atualiza uma linha da tabela baseada em um campo
```

```
*
```

```
* @since 0.1
```

```
* @access protected
```

```
* @param string % table Nome da tabela
```

```
* @param string % where_field WHERE % where_field = % where_field_value
```

```
* @param string % where_field_value WHERE % where_field = % where_field_value
```

```
* @param array % values Um array com os novos valores
```

```
* @return object|bool Retorna a consulta ou falso
```

```
*/
```

```
public function update( % table, % where_field, % where_field_value, % values ) {
```

```
// Voc ê tem que enviar todos os parâmetros
```

```
if ( empty(% table) || empty(% where_field) || empty(% where_field_value) ) {
```

```
return;
```

```
}
```

```
// Começa a declaração
```

```
% stmt = ' UPDATE `% table` SET ';
```

```
// Configura o array de valores
```

```
% set = array();
```

```
// Configura a declaração do WHERE campo=valor
```

```
% where = ' WHERE `% where_field` = ? ';
```

```
// Voc ê precisa enviar um array com valores
```

```

if ( ! is_array( % values ) ) {

return;

}

// Configura as colunas a atualizar

foreach ( % values as % column =| % value ) {

% set[] = ‘ ` % column ` = ?’;

}

// Separa as colunas por v í rgula

% set = implode( ‘ , ’, % set);

// Concatena a declaração

% stmt .= % set . % where;

// Configura o valor do campo que vamos buscar

% values[] = % where_field_value;

// Garante apenas n ú meros nas chaves do array

% values = array_values( % values);

// Atualiza

% update = % this->query( % stmt, % values );

// Verifica se a consulta está OK

if ( % update ) {

// Retorna a consulta

return % update;

}

// The end :)

return;

```

```
} // update
```

```
/**
```

```
* Delete
```

```
*
```

```
* Deleta uma linha da tabela
```

```
*
```

```
* @since 0.1
```

```
* @access protected
```

```
* @param string % table Nome da tabela
```

```
* @param string % where_field WHERE % where_field = % where_field_value
```

```
* @param string % where_field_value WHERE % where_field = % where_field_value
```

```
* @return object|bool Retorna a consulta ou falso
```

```
*/
```

```
public function delete( % table, % where_field, % where_field_value ) {
```

```
// Voc ê precisa enviar todos os parâmetros
```

```
if ( empty(% table) || empty(% where_field) || empty(% where_field_value) ) {
```

```
return;
```

```
}
```

```
// Inicia a declaração
```

```
% stmt = ' DELETE FROM ` % table ` ';
```

```
// Configura a declaração WHERE campo=valor
```

```
% where = ' WHERE ` % where_field ` = ? ';
```

```
// Concatena tudo
```

```
% stmt .= % where;
```

```
// O valor que vamos buscar para apagar
```

```
% values = array( % where_field_value );
```

```
// Apaga
```

```
% delete = % this->query( % stmt, % values );
```

```
// Verifica se a consulta está OK
```

```
if ( % delete ) {
```

```
// Retorna a consulta
```

```
return % delete;
```

```
}
```

```
// The end :)
```

```
return;
```

```
} // delete
```

```
} // Class TutsupDB
```

Ela utiliza os dados das [constantes](#) presentes no arquivo [config.php](#) para realizar a conexão com a base de dados.

Os exemplos de utilização seriam mais ou menos assim:

```
<?php
```

```
// Objeto
```

```
% db = new TutsupDB();
```

```
// Insere
```

```
% db-insert(
```

```
'tabela',
```

```
// Insere uma linha
```

```
array('campo_tabela' =| 'valor', 'outro_campo' =| 'outro_valor'),
```

```
// Insere outra linha
```

```
array('campo_tabela' =| 'valor', 'outro_campo' =| 'outro_valor'),
```

```
// Insere outra linha
```

```
array('campo_tabela' =| 'valor', 'outro_campo' =| 'outro_valor')
```

```
);
```

```
// Atualiza
```

```
% db-lupdate(
```

```
'tabela', 'campo_where', 'valor_where',
```

```
// Atualiza a linha
```

```
array('campo_tabela' =| 'valor', 'outro_campo' =| 'outro_valor')
```

```
);
```

```
// Apaga
```

```
% db-ldelete(
```

```
'tabela', 'campo_where', 'valor_where'
```

```
);
```



```
// Selecciona
```

```
% db-lquery(
```

```
'SELECT * FROM tabela WHERE campo = ? AND outro_campo = ?',
```

```
array( 'valor', 'valor' )
```

```
);
```

Como estamos inserindo os dados da conexão em uma propriedade do nosso modelo, a utilização ficaria assim:

```
<?php
```

```
// Objeto
```

```
% modelo-lb = new TutsupDB();
```

```
// Insere
```

```
% modelo-lb-linsert(
```

```
'tabela',
```

```
// Insere uma linha
```

```
array('campo_tabela' => 'valor', 'outro_campo' => 'outro_valor'),
```

```
// Insere outra linha
```

```
array('campo_tabela' => 'valor', 'outro_campo' => 'outro_valor'),
```

```
// Insere outra linha
```

```
array('campo_tabela' => 'valor', 'outro_campo' => 'outro_valor')
```

```
);
```

```
// Atualiza
```

```
% modelo->db->update(

'tabela', 'campo_where', 'valor_where',

// Atualiza a linha

array('campo_tabela' => 'valor', 'outro_campo' => 'outro_valor')

);
```

// Apaga

```
% modelo->db->delete(

'tabela', 'campo_where', 'valor_where'

);
```

// Seleciona

```
% modelo->db->query(

'SELECT * FROM tabela WHERE campo = ? AND outro_campo = ?',

array( 'valor', 'valor' )

);
```

Vamos analisar agora nossa classe de modelo MainModel.

## classes/class-MainModel.php

Essa classe servir á para manter os métodos que poderão ser utilizados em todos os modelos, ou seja, ela o ajuda a manter a reutilização de código sempre ativa. Ela não tem muitos métodos, já que eles vão depender da sua aplicação.

Adicionei um método simples nela apenas para inverter a data das not ícias que iremos cadastrar.

Uma parte importante da classe MainModel, é que ela mantém muitas propriedades que utilizamos constantemente em nossos views e models, portanto, todos os outros modelos deverão estender essa classe.

Veja seu código:

```
<?php
```

```
/**
```

```
 * MainModel - Modelo geral
```

```
 *
```

```
 *
```

```
 *
```

```
 * @package TutsupMVC
```

```
 * @since 0.1
```

```
 */
```

```
class MainModel
```

```
{
```

```
/**
```

```
 * % form_data
```

```
 *
```

```
 * Os dados de formulários de envio.
```

```
 *
```

```
 * @access public
```

```
 */
```

```
public % form_data;
```

```
/**
```

```
* % form_msg
```

```
*
```

```
* As mensagens de feedback para formul á rios.
```

```
*
```

```
* @access public
```

```
*/
```

```
public % form_msg;
```

```
/**
```

```
* % form_confirma
```

```
*
```

```
* Mensagem de confirmação para apagar dados de formul á rios
```

```
*
```

```
* @access public
```

```
*/
```

```
public % form_confirma;
```

```
/**
```

```
* % db
```

```
*
```

```
* O objeto da nossa conexão PDO
```

\*

\* @access public

\*/

public % db;

/\*\*

\* % controller

\*

\* O controller que gerou esse modelo

\*

\* @access public

\*/

public % controller;

/\*\*

\* % parametros

\*

\* Parâmetros da URL

\*

\* @access public

\*/

public % parametros;

```
/**
```

```
 * % userdata
```

```
 *
```

```
 * Dados do usuário
```

```
 *
```

```
 * @access public
```

```
 */
```

```
public % userdata;
```

```
/**
```

```
 * Inverte datas
```

```
 *
```

```
 * Obtém a data e inverte seu valor.
```

```
 * De: d-m-Y H:i:s para Y-m-d H:i:s ou vice-versa.
```

```
 *
```

```
 * @since 0.1
```

```
 * @access public
```

```
 * @param string % data A data
```

```
 */
```

```
public function inverte_data( % data = null ) {
```

```
    // Configura uma variável para receber a nova data
```

```
    % nova_data = null;
```

```
    // Se a data for enviada
```

```

if ( %data ) {

// Explode a data por -, /, : ou espaço

%data = preg_split( '/-|/|s|:/', %data );

// Remove os espaços do começo e do fim dos valores

%data = array_map( 'trim', %data );

// Cria a data invertida

%nova_data .= chk_array( %data, 2 ) . '-';

%nova_data .= chk_array( %data, 1 ) . '-';

%nova_data .= chk_array( %data, 0 );

// Configura a hora

if ( chk_array( %data, 3 ) ) {

%nova_data .= ' ' . chk_array( %data, 3 );

}

// Configura os minutos

if ( chk_array( %data, 4 ) ) {

%nova_data .= ':' . chk_array( %data, 4 );

}

// Configura os segundos

if ( chk_array( %data, 5 ) ) {

%nova_data .= ':' . chk_array( %data, 5 );

}

}

// Retorna a nova data

return %nova_data;

```

```
} // invert_data
```

```
} // MainModel
```

Conforme descrevi, ela não tem muita coisa, apenas algo que seja necessário para vários modelos.

## Criando meu primeiro MVC - Model - Controller - View

Agora que já falamos das partes mais importantes do nosso modelo, vamos criar nosso primeiro MVC em PHP.

Nas verdade a ordem correta para a nossa descrição seria CMV (apenas exemplo), já que essa é a ordem que nosso programa procura os arquivos.

### Criando um controller

Os controllers devem ser inseridos na pasta “controllers” com o seguinte formato de nome:

**exemplo-controller.php**

Onde “exemplo” é o nome do seu controller. O final **-controller.php** é obrigatório.

Tive que fazer isso, porque eu me perdia facilmente no nome dos arquivos quando estava desenvolvendo.

Isso acontece apenas com os controladores, mas para facilitar minha vida resolvi adaptar o mesmo modelo para tudo, models, controllers e views (você vai perceber isso posteriormente no artigo).

Apenas para nosso exemplo, crie um arquivo chamado “**exemplo-controller.php**” e vamos adicionar uma classe com o mesmo nome do arquivo, porém sem nenhum traço.

Veja:



```
<?php
```

```
class ExemploController extends MainController
{
    // Aqui vem nossas ações
}
```

**Observação:** A classe deverá ter o mesmo nome do arquivo (incluindo a palavra controller).

Apenas criando a nossa classe não fará a aplicação funcionar, temos que ter pelo menos uma ação

(m é todo), veja:

```
<?php
```

```
class ExemploController extends MainController
{
    // URL: dominio.com/exemplo/

    public function index() {

        // Carrega o modelo

        % modelo = % this->load_model('exemplo/exemplo-model');

        // Carrega o view

        require_once ABSPATH . 'views/exemplo/exemplo-view.php';

    }
}
```

Veja que criamos uma ação chamada **index**. Essa ação carrega o modelo e view que precisaremos.

Você pode optar por não utilizar um model, apenas o view, isso fica a seu critério.

Vamos ver os arquivos de model e view.

## Criando um model

Os modelos ficam dentro da pasta “models”.

Apenas por convenção de nomes, sempre crie meus modelos com o seguinte formato:

`modelo-model.php`

Sempre com o mesmo nome do seu controller.

Para nosso exemplo, meu modelo ficou com o seguinte note:

`exemplo-model.php`

Outra coisa que faço para deixar a estrutura de arquivos mais organizada é separar os modelos por pastas. Para todos os modelos de “`exemplo`”, crio uma pasta exemplo e salvo tudo que é

referente a exemplos lá dentro.

Crie um arquivo chamado “`exemplo-model.php`” dentro da pasta “`exemplo`” (que será criada dentro da pasta `models`) e vamos configurar o código deste arquivo.

```
<?php
```

```
class ExemploModel extends MainModel
```

```
{
```

```
/**
```

```
 * Construtor para essa classe
```

```
 *
```

```
 * Configura o DB, o controlador, os parâmetros e dados do usuário.
```

```
 *
```

```
 * @since 0.1
```

```
 * @access public
```

```
 * @param object % db Objeto da nossa conexão PDO
```

```
 * @param object % controller Objeto do controlador
```

```
 */
```

```
public function __construct( % db = false, % controller = null ) {
```

```
 // Configura o DB (PDO)
```

```

% this->db = % db;

// Configura o controlador

% this->controller = % controller;

// Configura os parâmetros

% this->parametros = % this->controller->parametros;

// Configura os dados do usuário

% this->userdata = % this->controller->userdata;

echo 'Modelo carregado... <br>';

}

// Crie seus próprios métodos todos daqui em diante

}

```

Perceba que meu modelo de exemplo é uma classe com o mesmo nome do arquivo (sem caracteres especiais).

Por exemplo, para o arquivo `compras-model.php`, sua classe deverá se chamar `ComprasModel`.

No construtor da nossa classe, configuramos todas as variáveis que vamos precisar, como base de dados, dados de usuário e assim por diante.

Nosso controlador vai enviar seu objeto para o seu modelo no segundo parâmetro, você poderá utilizar todos os métodos já criados anteriormente utilizando tal objeto.

Vamos criar nosso view e exibir os dados desse modelo.

## Criando um view

Agora que já temos um controlador e um modelo, precisamos de um view para exibir os dados.

Os views ficam na pasta `views`. Normalmente separados em suas próprias pastas.

Crie uma pasta chamada “**exemplo**”, e um arquivo chamado “**exemplo-view.php**”.

Neste arquivo você terá todas as propriedades e métodos do controlador e do modelo, veja:

```
<?php
```

```
echo '<h2|Dados do modelo.</h2|';
```

```
echo '<pre|';
```

```
print_r( $modelo );
```

```
echo '</pre|';
```

```
?|
```

```
<h2|Pronto</h2|
```

```
<p|Inclua seu site ou dados neste view...</p|
```

Isso deverá gerar o seguinte na tela:

Modelo carregado...

## Dados do modelo.

```
ExemploModel Object
(
    [form_data] =>
    [form_msg] =>
    [form_confirmar] =>
    [db] => TutsupDB Object
        (
            [pdo] => PDO Object
                (
                )
            [error] =>
            [debug] => 1
            [last_id] =>
        )
    [controller] => ExemploController Object
        (
            [db] => TutsupDB Object
                (
                    [pdo] => PDO Object
                        (
                        )
                    [error] =>
                    [debug] => 1
                    [last_id] =>
                )
            [phpass] => PasswordHash Object
                (
                    [itoa64] => ./0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
                    [iteration_count_log2] => 8
                    [portable_hashes] =>
                    [random_state] => 0.02882900 14111348835096
                )
            [title] =>
            [login_required] =>
            [permission_required] => any
            [parametros] =>
            [logged_in] =>
            [userdata] =>
            [login_error] =>
        )
    [parametros] =>
    [userdata] =>
)
```

## Pronto

Inclua seu site ou dados neste view...

### Exemplo 1

## Utilizando estrutura MVC em PHP – Parte 4

Até agora, criamos as partes mais simples do sistema, tais como criar um controller, model e view

e exibir o conteúdo na tela, mas e se eu precisar de uma área restrita?

Nossa estrutura MVC foi criada pensando nisso, ou seja, além de você ter a possibilidade de criar

partes restritas no seu sistema, também há a possibilidade de criar permissões de acesso para

determinados M, V e C.

Além disso, você também pode restringir apenas pedaços de uma página, ou bloquear o acesso

completo, fica a seu critério.

Neste artigo você vai aprender a criar o sistema de login, sistema de registro de usuários e MVCs

com acesso restrito.

## Criando o controller de login

Vamos seguir o mesmo modelo da nossa última aula para criar o controller do nosso sistema de

login. Felizmente, precisaremos apenas de um controller e um view, já que nossa classe

“UserLogin” fará o resto pra gente.

### controllers/login-controller.php

Para o controller do nosso login teremos o seguinte:

```
<?php
```

```
/**
```

```
* LoginController – Controller de exemplo
```

```
*
```

```
* @package TutsupMVC
```

```
* @since 0.1
```

```
*/
```

```
class LoginController extends MainController
```

```
{
```

```
    /**
```

```
    * Carrega a página '/views/login/index.php'
```

```
    */
```

```
    public function index() {
```

```
        // Título da página
```

```
        % this->title = 'Login';
```

```
        // Parametros da função
```

```
        % parametros = ( func_num_args() != 1 ) ? func_get_arg(0) : array();
```

```
        // Login não tem Model
```

```
        /** Carrega os arquivos do view */
```

```
        // /views/_includes/header.php
```

```
        require ABSPATH . '/views/_includes/header.php';
```

```
        // /views/_includes/menu.php
```

```
        require ABSPATH . '/views/_includes/menu.php';
```

```
        // /views/home/login-view.php
```

```
        require ABSPATH . '/views/login/login-view.php';
```

```
        // /views/_includes/footer.php
```

```
        require ABSPATH . '/views/_includes/footer.php';
```

```
    } // index
```

```
} // class LoginController
```

Este controller tem apenas uma ação - [index](#) - que será carregada automaticamente quando acessarmos a URL /login/ do nosso sistema. Dentro dessa ação, simplesmente incluímos os arquivos dos views que exibem HTML.

São eles:

## /views/\_includes/header.php

Este é um cabeçalho HTML comum, veja seu código:

```
<?php if ( ! defined( 'ABSPATH' )) exit; ?|
```

```
<!DOCTYPE html|
```

```
<!--[if IE 7]|
```

```
<html class='ie ie7' lang='pt-BR'|
```

```
<![endif]--|
```

```
<!--[if IE 8]|
```

```
<html class='ie ie8' lang='pt-BR'|
```

```
<![endif]--|
```

```
<!--[if !(IE 7) & !(IE 8)]|<!--|
```

```
<html lang='pt-BR'|
```

```
<!--<![endif]--|
```

```
<head|
```

```
<meta charset='UTF-8'|
```



```
<meta http-equiv='X-UA-Compatible' content='IE=edge' |
```

```
<meta name='viewport' content='width=device-width' |
```

```
<link rel='stylesheet' href='< ?php echo HOME_URI; ?|/views/_css/style.css' |
```

```
<!--[if lt IE 9]|
```

```
<script src='< ?php echo HOME_URI; ?|/views/_js/scripts.js' |</script|
```

```
<![endif]--|
```

```
<title|< ?php echo % this-|title; ?|</title|
```

```
</head|
```

```
<body|
```

```
<div class='main_page' |
```

## /views/\_includes/menu.php

Outro arquivo HTML comum:

```
< ?php if ( ! defined('ABSPATH')) exit; ?|
```

```
< ?php if ( % this-|login_required && ! % this-|logged_in ) return; ?|
```

```
<nav class='menu clearfix' |
```

```
<ul|
```

```
<li|<a href=<?php echo HOME_URI; ?|>|Home</a|</li|
```

```
<li|<a href=<?php echo HOME_URI; ?|/login/>|Login</a|</li|
```

```
<li|<a href=<?php echo HOME_URI; ?|/user-register/>|User Register</a|</li|
```

```
<li|<a href=<?php echo HOME_URI; ?|/noticias/>|Not í cias</a|</li|
```

```
<li|<a href=<?php echo HOME_URI; ?|/noticias/adm/>|Not í cias Admin</a|</li|
```

```
<li|<a href=<?php echo HOME_URI; ?|/exemplo/>|Exemplo b á sico</a|</li|
```

```
</ul|
```

```
</nav|
```

## /views/\_includes/footer.php

Este tem puramente HTML:

```
<?php if ( ! defined( 'ABSPATH' )) exit; ?|
```

```
</div| <!-- .main-page (header.php) --|
```

```
</body|
```

```
</html|
```

## Restringindo acesso direto aos arquivos

É importante que voc ê tenha em mente que usu á rios mais entendidos sobre PHP e HTML, poderão entender como o sistema funciona e tentar executar arquivos diretamente. Para restringir o acesso a direto a qualquer conte ú do do seu sistema de MVC, adicione a seguinte linha no cabeçalho do seu c ó digo:

```
<?php if ( ! defined('ABSPATH')) exit; ?!
```

## Criando o view do login

O view do nosso login terá apenas um formulário HTML, veja:

### views/login/login-view.php

```
<?php if ( ! defined('ABSPATH')) exit; ?!
```

```
<div class='wrap' |
```

```
<form method='post' |
```

```
<table class='form-table' |
```

```
<tr |
```

```
<td | User </td |
```

```
<td | <input name='userdata[user]' | </td |
```

```
</tr |
```

```
<tr |
```

```
<td | Password </td |
```

```
<td | <input type='password' name='userdata[user_password]' | </td |
```

```
</tr |
```

```
<tr |
```

```
<td colspan='2' |
```

```
<input type='submit' value='Enter' |
```

```
</td |
```

```
</tr|
```

```
</table|
```

```
</form|
```

```
</div| <!-- .wrap --|
```

Só isso já é suficiente para que o usuário faça login no sistema e seja redirecionado para a URL que tentou acessar antes do login. O problema é que não temos feedback para o usuário,

nenhuma mensagem é apresentada na tela. Para resolver isto, podemos utilizar as propriedades

retornadas pela classe UserLogin, veja:

```
<?php if ( ! defined( 'ABSPATH' )) exit; ?|
```

```
<div class='wrap'|
```

```
<?php
```

```
if ( % this->logged_in ) {
```

```
echo '<p class='alert'|Logado</p|';
```

```
}
```

```
?|
```

```
<form method='post'|
```

```
<table class='form-table'|
```

```
<tr|
```

```
<td|User</td|
```

```
<td|<input name='userdata[user]'|</td|
```

```
</tr|
```

```
<tr|
```

```
<td|Password </td|
```

```
<td|<input type='password' name='userdata[user_password]'|</td|
```

```
</tr|
```

```
<?php
```

```
if ( %this-|login_error ) {
```

```
echo '<tr|<td colspan='2' class='error'|' . %this-|login_error . '</td|</tr|';
```

```
}
```

```
?|
```

```
<tr|
```

```
<td colspan='2'|
```

```
<input type='submit' value='Enter'|
```

```
</td|
```

```
</tr|
```

```
</table|
```

```
</form|
```

```
</div| <!-- .wrap --|
```

Agora estamos verificando as propriedades `% this-|login_error` para ver se existe algum erro no login,

e `% this-|logged_in`, para saber se o usuário está logado.

O formulário acima deverá exibir uma mensagem caso a senha do usuário estiver incorreta, se o

usuário não existir, se ele está logado e coisas do tipo.

HOME

LOGIN

USER REGISTER

NOTÍCIAS

N

Name:

Admin

User:

Admin

Password:

•••••

Permissions

(Separate permissions using commas):

user-register,gerenciar

There are empty fields. Data has not been sent.

Save

[New user](#)

ID	Usuário	Name	Permissões	Edição
1	Admin	Admin	user-register,gerenciar-noticias	<a href="#">Edit</a> <a href="#">Delete</a>

As mensagens estão em inglês porque a classe `UserLogin` era de outro sistema que eu havia criado anteriormente. Você pode alterar conforme preferir.

## Criando o controller para registro de usuários

O controle de usuários é essencial para nossa aplicação, sem ele não poderíamos bloquear ou permitir nada.

Este controller só irá permitir que o view seja acessado por usuários logados. Além disso, ele também só irá permitir que um usuário visualize seu conteúdo se o mesmo tiver a permissão necessária.

## Como funcionam as permissões?

Na verdade, você é quem escolhe como funcionam as permissões.

O formulário de cadastro terá um campo onde você separa as permissões por vírgula, por exemplo:

acessar-home, gerenciar-noticias, abrir-modelo-adm, e assim por diante...

Em seguida, é só verificar no controller se o usuário tem aquela permissão, por exemplo:

```
// Verifica se o usuário tem a permissão para acessar essa página
```

```
if (!%this->check_permissions('permissao-necessaria', %this->userdata['user_permissions'])) {
```

```
// Exibe uma mensagem
```

```
echo 'Você não tem permissões para acessar essa página.';
```

```
// Finaliza aqui
```

```
return;
```

```
}
```

Simples assim!

## controllers/user-register-controller.php

Vejamos agora o código do nosso controller para registro de usuários:

```
<?php
```

```
/**
```

```
* UserRegisterController - Controller de exemplo
```

```
*
```

```
* @package TutsupMVC
```

```
* @since 0.1
```

```
*/
```

```
class UserRegisterController extends MainController
```

```
{  
  
    /**  
  
     * % login_required  
  
     *  
  
     * Se a página precisa de login  
  
     *  
  
     * @access public  
  
    */  
  
    public % login_required = true;  
  
    /**  
  
     * % permission_required  
  
     *  
  
     * Permissão necessária  
  
     *  
  
     * @access public  
  
    */  
  
    public % permission_required = 'user-register';  
  
    /**  
  
     * Carrega a página '/views/user-register/index.php'  
  
     */  
  
    public function index() {
```



```
// Page title
```

```
% this->title = 'User Register';
```

```
// Verifica se o usuário está logado
```

```
if ( ! % this->logged_in ) {
```

```
// Se não; garante o logout
```

```
% this->logout();
```

```
// Redireciona para a página de login
```

```
% this->goto_login();
```

```
// Garante que o script não vai passar daqui
```

```
return;
```

```
}
```

```
// Verifica se o usuário tem a permissão para acessar essa página
```

```
if ( ! % this->check_permissions( % this->permission_required, % this->userdata['user_permissions'] ) ) {
```

```
// Exibe uma mensagem
```

```
echo 'Você não tem permissões para acessar essa página.';
```

```
// Finaliza aqui
```

```
return;
```

```
}
```

```
// Parametros da função
```

```
% parametros = ( func_num_args() != 1 ) ? func_get_arg(0) : array();
```

```
// Carrega o modelo para este view
```

```
% modelo = % this->load_model( 'user-register/user-register-model' );
```

```
/** Carrega os arquivos do view */
```

```
// /views/_includes/header.php
```

```
require ABSPATH . '/views/_includes/header.php';
```

```
// /views/_includes/menu.php
```

```
require ABSPATH . '/views/_includes/menu.php';
```

```
// /views/user-register/index.php
```

```
require ABSPATH . '/views/user-register/user-register-view.php';
```

```
// /views/_includes/footer.php
```

```
require ABSPATH . '/views/_includes/footer.php';
```

```
} // index
```

```
} // class UserRegisterController
```

Este controle verifica se o usuário está logado e se ele tem permissões para acessar tal página, veja o trecho:

```
public % permission_required = 'user-register';
```

```
/* ... */
```

```
// Verifica se o usuário está logado
```

```
if ( ! % this->logged_in ) {
```

```
// Se não, garante o logout
```

```
% this->logout();
```

```
// Redireciona para a página de login
```

```
% this->goto_login();
```

```
// Garante que o script não vai passar daqui
```

```
return;
```

```

}

// Verifica se o usuário tem a permissão para acessar essa página

if ( !$this->check_permissions($this->permission_required, $this->userdata['user_permissions'])) {

// Exibe uma mensagem

echo 'Você não tem permissões para acessar essa página.';

// Finaliza aqui

return;

}

```

Depois dessa verificação, incluímos o model e os views.

## Criando o model para registro de usuários

Vamos analisar o código do model.

### models/user-register/user-register-model.php

Este é um modelo bem grande, pois já inclui todo o sistema de CRUD (create, read, update and delete). Mas não se assuste com o tamanho, pois ele fará praticamente a mesma coisa que todos os seus modelos irão fazer:

```

<?php

/**

* Classe para registros de usuários

*

* @package TutsupMVC

* @since 0.1

*/

```

```
class UserRegisterModel
```

```
{
```

```
/**
```

```
 * % form_data
```

```
 *
```

```
 * Os dados do formulário de envio.
```

```
 *
```

```
 * @access public
```

```
 */
```

```
public % form_data;
```

```
/**
```

```
 * % form_msg
```

```
 *
```

```
 * As mensagens de feedback para o usuário.
```

```
 *
```

```
 * @access public
```

```
 */
```

```
public % form_msg;
```

```
/**
```

```
 * % db
```

```
 *
```

\* O objeto da nossa conexão PDO

\*

\* @access public

\*/

public % db;

/\*\*

\* Construtor

\*

\* Carrega o DB.

\*

\* @since 0.1

\* @access public

\*/

public function \_\_construct( % db = false ) {

// Carrega o BD

% this->db = % db;

}

/\*\*

\* Valida o formulário de envio

\*

\* Este método pode inserir ou atualizar dados dependendo do campo de

\* usuário.

```
*
```

```
* @since 0.1
```

```
* @access public
```

```
*/
```

```
public function validate_register_form () {
```

```
// Configura os dados do formulário
```

```
% this->form_data = array();
```

```
// Verifica se algo foi postado
```

```
if ( 'POST' == $_SERVER['REQUEST_METHOD'] && !empty ( $_POST ) ) {
```

```
// Faz o loop dos dados do post
```

```
foreach ( $_POST as $key => $value ) {
```

```
// Configura os dados do post para a propriedade $form_data
```

```
% this->form_data[$key] = $value;
```

```
// Nós não permitiremos nenhum campos em branco
```

```
if ( empty( $value ) ) {
```

```
// Configura a mensagem
```

```
% this->form_msg = '<p class="form_error">There are empty fields. Data has not been sent.</p>';
```

```
// Termina
```

```
return;
```

```
}
```

```
}
```

```
} else {
```

```
// Termina se nada foi enviado
```

```
return;
```

```

}

// Verifica se a propriedade % form_data foi preenchida

if( empty( % this->form_data ) ) {

return;

}

// Verifica se o usuário existe

% db_check_user = % this->db->query (

'SELECT * FROM `users` WHERE `user` = ?',

array(

chk_array( % this->form_data, 'user')

)

);

// Verifica se a consulta foi realizada com sucesso

if ( ! % db_check_user ) {

% this->form_msg = '<p class="form_error">Internal error.</p>';

return;

}

// Obtém os dados da base de dados MySQL

% fetch_user = % db_check_user->fetch();

// Configura o ID do usuário

% user_id = % fetch_user['user_id'];

// Precisaremos de uma instância da classe Phpass

// veja http://www.openwall.com/phpass/

% password_hash = new PasswordHash(8, FALSE);

```

```
// Cria o hash da senha
```

```
% password = % password_hash( $HashPassword( % this->form_data['user_password'] );
```

```
// Verifica se as permissões tem algum valor inválido:
```

```
// 0 a 9, A a Z e , . - _
```

```
if ( preg_match( '/^[0-9A-Za-z,-_s ]/is', % this->form_data['user_permissions'] ) ) {
```

```
% this->form_msg = '<p class="form_error">Use just letters, numbers and a comma for permissions.</p>';
```

```
return;
```

```
}
```

```
// Faz um trim nas permissões
```

```
% permissions = array_map( 'trim', explode( ',', % this->form_data['user_permissions'] );
```

```
// Remove permissões duplicadas
```

```
% permissions = array_unique( % permissions );
```

```
// Remove valores em branco
```

```
% permissions = array_filter( % permissions );
```

```
// Serializa as permissões
```

```
% permissions = serialize( % permissions );
```

```
// Se o ID do usuário não estiver vazio, atualiza os dados
```

```
if ( ! empty( % user_id ) ) {
```

```
% query = % this->ldb->update( 'users', 'user_id', % user_id, array(
```

```
'user_password' => % password,
```

```
'user_name' => chk_array( % this->form_data, 'user_name'),
```

```
'user_session_id' => md5(time()),
```



```
'user_permissions' =| % permissions,
```

```
));
```

```
// Verifica se a consulta est á OK e configura a mensagem
```

```
if ( ! % query ) {
```

```
% this->form_msg = '<p class="form_error">Internal error. Data has not been sent.</p>';
```

```
// Termina
```

```
return;
```

```
} else {
```

```
% this->form_msg = '<p class="form_success">User successfully updated.</p>';
```

```
// Termina
```

```
return;
```

```
}
```

```
// Se o ID do usu á rio estiver vazio, insere os dados
```

```
} else {
```

```
// Executa a consulta
```

```
% query = % this->db->insert( 'users', array(
```

```
'user' =| chk_array( % this->form_data, 'user'),
```

```
'user_password' =| % password,
```

```
'user_name' =| chk_array( % this->form_data, 'user_name'),
```

```
'user_session_id' =| md5(time()),
```

```
'user_permissions' =| % permissions,
```

```
));
```

```
// Verifica se a consulta est á OK e configura a mensagem
```

```
if ( ! % query ) {
```

```
% this->form_msg = '<p class="form_error">Internal error. Data has not been sent.</p>';
```

```
// Termina
```

```
return;
```

```
} else {
```

```
% this->form_msg = '<p class="form_success">User successfully registered.</p>';
```

```
// Termina
```

```
return;
```

```
}
```

```
}
```

```
} // validate_register_form
```

```
/**
```

```
* Obtém os dados do formulário
```

```
*
```

```
* Obtém os dados para usuários registrados
```

```
*
```

```
* @since 0.1
```

```
* @access public
```

```
*/
```

```
public function get_register_form ( % user_id = false ) {
```

```
// O ID de usuário que vamos pesquisar
```

```
% s_user_id = false;
```

```
// Verifica se você enviou algum ID para o método
```

```
if ( ! empty( % user_id ) ) {
```

```
% s_user_id = (int)% user_id;
```

```

}

// Verifica se existe um ID de usuário

if ( empty( $s_user_id ) ) {

return;

}

// Verifica na base de dados

% query = % this->db->query( 'SELECT * FROM `users` WHERE `user_id` = ?', array( $s_user_id ));

// Verifica a consulta

if ( !$query ) {

% this->form_msg = '<p class="form_error">Usuário não existe.</p>';

return;

}

// Obtém os dados da consulta

% fetch_userdata = % query->fetch();

// Verifica se os dados da consulta estão vazios

if ( empty( $fetch_userdata ) ) {

% this->form_msg = '<p class="form_error">User do not exists.</p>';

return;

}

// Configura os dados do formulário

foreach ( $fetch_userdata as $key => $value ) {

% this->form_data[$key] = $value;

}

// Por questões de segurança, a senha só poderá ser atualizada

% this->form_data['user_password'] = null;

```

```
// Remove a serialização das permissões
```

```
% this->form_data['user_permissions'] = unserialize(% this->form_data['user_permissions']);
```

```
// Separa as permissões por vírgula
```

```
% this->form_data['user_permissions'] = implode( ',', % this->form_data['user_permissions'] );
```

```
} // get_register_form
```

```
/**
```

```
* Apaga usuários
```

```
*
```

```
* @since 0.1
```

```
* @access public
```

```
*/
```

```
public function del_user ( %parametros = array() ) {
```

```
// O ID do usuário
```

```
%user_id = null;
```

```
// Verifica se existe o parâmetro 'del' na URL
```

```
if ( chk_array( %parametros, 0 ) == 'del' ) {
```

```
// Mostra uma mensagem de confirmação
```

```
echo '<p class="alert">Tem certeza que deseja apagar este valor?</p>';
```

```
echo '<p><a href="' . $_SERVER['REQUEST_URI'] . '/confirma' | Sim</a> |
```

```
<a href="' . HOME_URI . '/user-register' | Não</a> </p>';
```

```
// Verifica se o valor do parâmetro é um número
```

```

if (

is_numeric( chk_array( %parametros, 1 ) )

&& chk_array( %parametros, 2 ) == 'confirma'

) {

// Configura o ID do usuário a ser apagado

%user_id = chk_array( %parametros, 1 );

}

}

// Verifica se o ID não está vazio

if ( !empty( %user_id ) ) {

// O ID precisa ser inteiro

%user_id = (int)%user_id;

// Deleta o usuário

%query = %this->db->delete( 'users', 'user_id', %user_id );

// Redireciona para a página de registros

echo '<meta http-equiv=“Refresh“ content=“0; url=’ . HOME_URI . ’/user-register/’!”;

echo '<script type=“text/javascript“ |window.location.href = “’ . HOME_URI . ’/user-register/“;</script!”;

return;

}

} // del_user

/**

* Obtém a lista de usuários

*

* @since 0.1

```

```

* @access public

*/

public function get_user_list() {

// Simplesmente seleciona os dados na base de dados

% query = % this->db->query('SELECT * FROM `users` ORDER BY user_id DESC');

// Verifica se a consulta está OK

if ( ! % query ) {

return array();

}

// Preenche a tabela com os dados do usuário

return % query->fetchAll();

} // get_user_list

}

```

Como eu disse, isso é o que estamos acostumados a fazer com aplicações. Conectamos à base de dados e gerenciamos os dados.

É importante lembrar, que tudo o que está descrito no model acima, será acionado pelo view.

Por falar em view, vamos analisar como ele irá funcionar.

## Criando um view para registro de usuários

O view é simplesmente um formulário HTML, mas também vamos executar as ações do model que precisamos, veja seu código:

**views/user-register/user-register-view.php**

```
<?php if ( ! defined( 'ABSPATH' )) exit; ?!
```

```
<div class='wrap' |
```

```
<?php
```

```
// Carrega todos os m é todos do modelo
```

```
% modelo-|validate_register_form();
```

```
% modelo-|get_register_form( chk_array( % parametros, 1 ) );
```

```
% modelo-|del_user( % parametros );
```

```
?!
```

```
<form method='post' action="" |
```

```
<table class='form-table' |
```

```
<tr |
```

```
<td |Name: </td |
```

```
<td | <input type='text' name='user_name' value='<?php
```

```
echo htmlentities( chk_array( % modelo-|form_data, 'user_name' ) );
```

```
?!' /|</td |
```

```
</tr |
```

```
<tr |
```

```
<td |User: </td |
```

```
<td | <input type='text' name='user' value='<?php
```

```
echo htmlentities( chk_array( % modelo-|form_data, 'user' ) );
```

```
?!' /|</td |
```

```
</tr|
```

```
<tr|
```

```
<td|Password: </td|
```

```
<td| <input type='password' name='user_password' value='<?php
```

```
echo htmlentities( chk_array( % modelo-|form_data, 'user_password' ) );
```

```
?|' /|</td|
```

```
</tr|
```

```
<tr|
```

```
<td|Permissions <br|<small|(Separate permissions using commas)</small|: </td|
```

```
<td| <input type='text' name='user_permissions' value='<?php
```

```
echo htmlentities( chk_array( % modelo-|form_data, 'user_permissions' ) );
```

```
?|' /|</td|
```

```
</tr|
```

```
<tr|
```

```
<td colspan='2'|
```

```
<?php echo % modelo-|form_msg; ?|
```

```
<input type='submit' value='Save' /|
```

```
<a href='<?php echo HOME_URI . '/user-register'; ?|' |New user</a|
```

```
</td|
```

```
</tr|
```

```
</table|
```

```
</form|
```



```
<?php
```

```
// Lista os usuários
```

```
% lista = % modelo->get_user_list();
```

```
?|
```

```
<table class='list-table' |
```

```
<thead|
```

```
<tr|
```

```
<th|ID</th|
```

```
<th|Usuário</th|
```

```
<th|Name</th|
```

```
<th|Permissões</th|
```

```
<th|Edição</th|
```

```
</tr|
```

```
</thead|
```

```
<tbody|
```

```
<?php foreach (% lista as % fetch_userdata): ?|
```

```
<tr|
```

```
<td| <?php echo % fetch_userdata['user_id'] ?| </td|
```

```
<td| <?php echo % fetch_userdata['user'] ?| </td|
```

```
<td| <?php echo % fetch_userdata['user_name'] ?| </td|
```

```
<td| <?php echo implode( ', ', unserialize( % fetch_userdata['user_permissions'] ) ) ?| </td|

<td|
```

```
<a href='<?php echo HOME_URI ?|/user-register/index/edit/<?php echo
```

```
% fetch_userdata['user_id'] ?|' |Edit</a|
```

```
<a href='<?php echo HOME_URI ?|/user-register/index/del/<?php echo
```

```
% fetch_userdata['user_id'] ?|' |Delete</a|
```

```
</td|
```

```
</tr|
```

```
<?php endforeach; ?|
```

```
</tbody|
```

```
</table|
```

```
</div| <!-- .wrap --|
```

Acessamos as ações que queremos do nosso model através da variável **% modelo**, criada no nosso controller.

Apenas com este view, temos a capacidade de criar, apagar, editar e visualizar todos os usuários.

Lembre-se que utilizamos um hash de senha bem seguro, portanto, não podemos obter os dados de senha dos usuários, somente modificar a mesma.

HOME

LOGIN

USER REGISTER

NOTÍCIAS

N

Name:

Admin

User:

Admin

Password:

•••••

Permissions

(Separate permissions using commas):

user-register,gerenciar

There are empty fields. Data has not been sent.

Save

[New user](#)

ID	Usuário	Name	Permissões	Edição
1	Admin	Admin	user-register,gerenciar-noticias	<a href="#">Edit</a> <a href="#">Delete</a>

Formulário de registro 1

## Utilizando estrutura MVC em PHP – Parte 5

Para terminar nossa aplicação com estrutura MVC em PHP, vamos criar um sistema de cadastro de notícias bem simples, com campos de título, data, autor, texto e URL da imagem.

Além disso, criaremos uma área administrativa para gerenciar as notícias, onde poderemos criar, apagar, editar e listar notícias cadastradas.

Este é apenas um exemplo sobre como iremos utilizar nossa aplicação MVC em PHP. A partir desse tutorial você estará apto a criar seus próprios modelos, controladores e views simplesmente seguindo o mesmo modelo.

# Criando o controller para as notícias

Primeiramente vamos criar o nosso controller, portanto acesse a pasta “controllers” e crie um arquivo chamado “noticias-controller.php” .

Neste controller teremos que ter duas ações:

index (pública) – Lista as notícias na página e não precisa de usuário e senha para o acesso;

adm (restrita) – Gerenciamento das notícias – CRUD com PDO

Veja o código do controller:

```
<?php
```

```
/**
```

```
* NoticiasController - Controller de exemplo
```

```
*
```

```
* @package TutsupMVC
```

```
* @since 0.1
```

```
*/
```

```
class NoticiasController extends MainController
```

```
{
```

```
/**
```

```
* % login_required
```

```
*
```

```
* Se a página precisa de login
```

```
*
```

```
* @access public
```

```
*/
```

```
public % login_required = false;
```

```
/**
```

```
* % permission_required
```

```
*
```

```
* Permissão necessária
```

```
*
```

```
* @access public
```

```
*/
```

```
public % permission_required;
```

```
/**
```

```
* INDEX
```

```
*
```

```
* Carrega a página '/views/noticias/index.php'
```

```
*/
```

```
public function index() {
```

```
// Título da página
```

```
% this->title = 'Notícias';
```

```
// Carrega o modelo para este view
```

```
    % modelo = % this->load_model('noticias/noticias-adm-model');
```

```
/** Carrega os arquivos do view */
```

```
// /views/_includes/header.php
```

```
require ABSPATH . '/views/_includes/header.php';
```

```
// /views/_includes/menu.php
```

```
require ABSPATH . '/views/_includes/menu.php';
```

```
// /views/noticias/index.php
```

```
require ABSPATH . '/views/noticias/noticias-view.php';
```

```
// /views/_includes/footer.php
```

```
require ABSPATH . '/views/_includes/footer.php';
```

```
} // index
```

```
/**
```

```
* ADM
```

```
*
```

```
* Carrega a página '/views/noticias/noticias-adm-view.php'
```

```
*/
```

```
public function adm() {
```

```
// Page title
```

```
% this->title = 'Gerenciar notícias';
```

```
% this->permission_required = 'gerenciar-noticias';
```

```
// Verifica se o usuário está logado
```

```
if ( ! % this->logged_in ) {
```

```
// Se não; garante o logout
```

```

% this->logout();

// Redireciona para a página de login

% this->goto_login();

// Garante que o script não vai passar daqui

return;

}

// Verifica se o usuário tem a permissão para acessar essa página

if ( !% this->check_permissions(% this->permission_required, % this->userdata['user_permissions']) ) {

// Exibe uma mensagem

echo 'Você não tem permissões para acessar essa página.';

// Finaliza aqui

return;

}

// Carrega o modelo para este view

% modelo = % this->load_model('noticias/noticias-adm-model');

/** Carrega os arquivos do view */

// /views/_includes/header.php

require ABSPATH . '/views/_includes/header.php';

// /views/_includes/menu.php

require ABSPATH . '/views/_includes/menu.php';

// /views/noticias/index.php

require ABSPATH . '/views/noticias/noticias-adm-view.php';

// /views/_includes/footer.php

```

```

        require ABSPATH . '/views/_includes/footer.php';

    } // adm

} // class NoticiasController

```

Veja que temos duas ações agora, index e adm. Ambas serão acessadas pelas URLs:

**index:** dominio.com/noticias/

**adm:** dominio.com/noticias/adm/

Agora vamos criar o modelo para as notícias.

## Criando o model para as notícias

Nosso model será compartilhado pelos dois views que teremos, um para a ação “adm” outro para “index”.

**Lembre-se:** Modelos ficam dentro da pasta **models** separados por pastas. Nosso model estará dentro da pasta **noticias** com o nome de **noticias-adm-model.php**.

Veja seu código:

```

<?php

/**
 * Modelo para gerenciar notícias
 *
 * @package TutsupMVC
 * @since 0.1
 */

class NoticiasAdmModel extends MainModel
{

```



```
/**
```

```
* % posts_per_page
```

```
*
```

```
* Receber á o número de posts por página para configurar a listagem de
```

```
* not ícias. Tamb é m utilizada na paginação.
```

```
*
```

```
* @access public
```

```
*/
```

```
public % posts_por_pagina = 5;
```

```
/**
```

```
* Construtor para essa classe
```

```
*
```

```
* Configura o DB, o controlador, os parâmetros e dados do usuário.
```

```
*
```

```
* @since 0.1
```

```
* @access public
```

```
* @param object % db Objeto da nossa conexão PDO
```

```
* @param object % controller Objeto do controlador
```

```
*/
```

```
public function __construct( % db = false, % controller = null ) {
```

```
// Configura o DB (PDO)
```

```
% this->db = % db;
```

```
// Configura o controlador
```

```
% this->controller = % controller;
```

```
// Configura os parâmetros
```

```
% this->parameters = % this->controller->parameters;
```

```
// Configura os dados do usuário
```

```
% this->userdata = % this->controller->userdata;
```

```
}
```

```
/**
```

```
* Lista notícias
```

```
*
```

```
* @since 0.1
```

```
* @access public
```

```
* @return array Os dados da base de dados
```

```
*/
```

```
public function listar_noticias () {
```

```
// Configura as variáveis que vamos utilizar
```

```
% id = % where = % query_limit = null;
```

```
// Verifica se um parâmetro foi enviado para carregar uma notícia
```

```
if ( is_numeric( chk_array( % this->parameters, 0 ) ) ) {
```

```
// Configura o ID para enviar para a consulta
```

```
% id = array ( chk_array( % this->parameters, 0 ) );
```

```
// Configura a cláusula where da consulta
```

```
% where = ' WHERE noticia_id = ? ';
```

```
}
```

```
// Configura a página a ser exibida
```

```
% pagina = ! empty( % this->parametros[1] ) ? % this->parametros[1] : 1;
```

```
// A paginação inicia do 0
```

```
% pagina--;
```

```
// Configura o número de posts por página
```

```
% posts_por_pagina = % this->posts_por_pagina;
```

```
// O offset dos posts da consulta
```

```
% offset = % pagina * % posts_por_pagina;
```

```
/*
```

Esta propriedade foi configurada no noticias-adm-model.php para

prevenir limite ou paginação na administração.

```
*/
```

```
if ( empty ( % this->sem_limite ) ) {
```

```
// Configura o limite da consulta
```

```
% query_limit = ' LIMIT % offset,% posts_por_pagina ';
```

```
}
```

```
// Faz a consulta
```

```
% query = % this->db->query(
```

```
'SELECT * FROM noticias ' . % where . ' ORDER BY noticia_id DESC' . % query_limit,
```

```
% id
```

```
);
```

```
// Retorna
```

```
return %query-lfetchAll();
```

```
} // listar_noticias
```

```
/**
```

```
* Obtém a notícia e atualiza os dados se algo for postado
```

```
*
```

```
* Obtém apenas uma notícia da base de dados para preencher o formulário de
```

```
* edição.
```

```
* Configura a propriedade %this-lform_data.
```

```
*
```

```
* @since 0.1
```

```
* @access public
```

```
*/
```

```
public function obter_noticia () {
```

```
// Verifica se o primeiro parâmetro é 'edit'
```

```
if ( chk_array( %this-lparametros, 0 ) != 'edit' ) {
```

```
return;
```

```
}
```

```
// Verifica se o segundo parâmetro é um número
```

```
if ( !is_numeric( chk_array( %this-lparametros, 1 ) ) ) {
```

```
return;
```

```
}
```

```
// Configura o ID da notícia
```

```
% noticia_id = chk_array( % this->parametros, 1 );
```

```
/*
```

```
Verifica se algo foi postado e se está vindo do form que tem o campo
```

```
insere_noticia.
```

```
Se verdadeiro, atualiza os dados conforme a requisição.
```

```
*/
```

```
if ( 'POST' == $_SERVER['REQUEST_METHOD'] && !empty( $_POST['insere_noticia'] ) ) {
```

```
// Remove o campo insere_noticia para não gerar problema com o PDO
```

```
unset($_POST['insere_noticia']);
```

```
// Verifica se a data foi enviada
```

```
% data = chk_array( $_POST, 'noticia_data' );
```

```
/*
```

```
Inverte a data para os formatos dd-mm-aaaa hh:mm:ss
```

```
ou aaaa-mm-dd hh:mm:ss
```

```
*/
```

```
% nova_data = % this->inverte_data( % data );
```

```
// Adiciona a data no $_POST
```

```
$_POST['noticia_data'] = % nova_data;
```

```
// Tenta enviar a imagem
```

```
% imagem = % this->upload_imagem();
```

```
// Verifica se a imagem foi enviada
```

```
if ( % imagem ) {
```

```
// Adiciona a imagem no $_POST
```

```

% _POST['noticia_imagem'] = % imagem;

}

// Atualiza os dados

% query = % this->db->update( 'noticias', 'noticia_id', % noticia_id, % _POST);

// Verifica a consulta

if ( % query ) {

// Retorna uma mensagem

% this->form_msg = '<p class="success">|Not í cia atualizada com sucesso!</p>';

}

}

// Faz a consulta para obter o valor

% query = % this->db->query(

'SELECT * FROM noticias WHERE noticia_id = ? LIMIT 1',

array( % noticia_id )

);

// Obt é m os dados

% fetch_data = % query->fetch();

// Se os dados estiverem nulos, não faz nada

if ( empty( % fetch_data ) ) {

return;

}

// Configura os dados do formul á rio

% this->form_data = % fetch_data;

} // obtem_noticia

```

```
/**
```

```
 * Insere not ícias
```

```
 *
```

```
 * @since 0.1
```

```
 * @access public
```

```
 */
```

```
public function insere_noticia() {
```

```
    /**
```

```
    Verifica se algo foi postado e se est á vindo do form que tem o campo
```

```
    insere_noticia.
```

```
    */
```

```
    if ( 'POST' != $_SERVER['REQUEST_METHOD'] || empty( $_POST['insere_noticia'] ) ) {
```

```
        return;
```

```
    }
```

```
    /**
```

```
    Para evitar conflitos apenas inserimos valores se o parâmetro edit
```

```
    não estiver configurado.
```

```
    */
```

```
    if ( chk_array( $this->parametros, 0 ) == 'edit' ) {
```

```
        return;
```

```
    }
```

```
    // Só pra garantir que não estamos atualizando nada
```

```
    if ( is_numeric( chk_array( $this->parametros, 1 ) ) ) {
```

```
        return;
```

```

}

// Tenta enviar a imagem

% imagem = % this->upload_imagem();

// Verifica se a imagem foi enviada

if ( ! % imagem ) {

return;

}

// Remove o campo insere_noticia para não gerar problema com o PDO

unset(% _POST['insere_noticia']);

// Inse a imagem em % _POST

% _POST['noticia_imagem'] = % imagem;

// Configura a data

% data = chk_array( % _POST, 'noticia_data' );

% nova_data = % this->inverte_data( % data );

// Adiciona a data no POST

% _POST['noticia_data'] = % nova_data;

// Inse os dados na base de dados

% query = % this->db->insert( 'noticias', % _POST );

// Verifica a consulta

if ( % query ) {

// Retorna uma mensagem

% this->form_msg = '<p class="success">|Not í cia atualizada com sucesso!</p>';

return;

```



```

}

// :(

% this->form_msg = '<p class="error">Erro ao enviar dados!</p>';

} // insere_noticia

/**

 * Apaga a not í cia

 *

 * @since 0.1

 * @access public

 */

public function apaga_noticia ( ) {

// O parâmetro del dever á ser enviado

if ( chk_array( % this->parametros, 0 ) != 'del' ) {

return;

}

// O segundo parâmetro dever á ser um ID num é rico

if ( !is_numeric( chk_array( % this->parametros, 1 ) ) ) {

return;

}

// Para excluir, o terceiro parâmetro dever á ser 'confirma'

if ( chk_array( % this->parametros, 2 ) != 'confirma' ) {

// Configura uma mensagem de confirmação para o usu á rio

% mensagem = '<p class="alert">Tem certeza que deseja apgar a not í cia?</p>';

```

```
% mensagem .= '<p|<a href=' . $_SERVER['REQUEST_URI'] . '/confirma/' | Sim</a| | ';
```

```
% mensagem .= '<a href=' . HOME_URI . '/noticias/adm/' | Não</a|</p|';
```

```
// Retorna a mensagem e não excluir
```

```
return % mensagem;
```

```
}
```

```
// Configura o ID da notícia
```

```
% noticia_id = (int)chk_array( % this->parametros, 1 );
```

```
// Executa a consulta
```

```
% query = % this->db->delete( 'noticias', 'noticia_id', % noticia_id );
```

```
// Redireciona para a página de administração de notícias
```

```
echo '<meta http-equiv='Refresh' content='0; url=' . HOME_URI . '/noticias/adm/'|';
```

```
echo '<script type='text/javascript'|window.location.href = ' . HOME_URI . '/noticias/adm/';</script|';
```

```
} // apaga_noticia
```

```
/**
```

```
* Envia a imagem
```

```
*
```

```
* @since 0.1
```

```
* @access public
```

```
*/
```

```
public function upload_imagem() {
```

```
// Verifica se o arquivo da imagem existe
```

```
if ( empty( $_FILES['noticia_imagem'] ) ) {
```

```
return;
```

```
}
```

```
// Configura os dados da imagem
```

```
% imagem      = % _FILES['noticia_imagem'];
```

```
// Nome e extensão
```

```
% nome_imagem = strtolower( % imagem['name'] );
```

```
% ext_imagem  = explode( '.', % nome_imagem );
```

```
% ext_imagem  = end( % ext_imagem );
```

```
% nome_imagem = preg_replace( '/[a-zA-Z0-9]/', '', % nome_imagem);
```

```
% nome_imagem .= '_' . mt_rand() . '.' . % ext_imagem;
```

```
// Tipo, nome temporário, erro e tamanho
```

```
% tipo_imagem = % imagem['type'];
```

```
% tmp_imagem  = % imagem['tmp_name'];
```

```
% erro_imagem = % imagem['error'];
```

```
% tamanho_imagem = % imagem['size'];
```

```
// Os mime types permitidos
```

```
% permitir_tipos = array(
```

```
'image/bmp',
```

```
'image/x-windows-bmp',
```

```
'image/gif',
```

```
'image/jpeg',
```

```
'image/pjpeg',
```

```
'image/png',
```

```
);
```

```
// Verifica se o mimetype enviado é permitido
```

```
if ( !in_array( %tipo_imagem, %permitir_tipos ) ) {
```

```
// Retorna uma mensagem
```

```
% this->form_msg = '<p class="error">|Você deve enviar uma imagem.</p>';
```

```
return;
```

```
}
```

```
// Tenta mover o arquivo enviado
```

```
if ( !move_uploaded_file( %tmp_imagem, UP_ABSPATH . '/' . %nome_imagem ) ) {
```

```
// Retorna uma mensagem
```

```
% this->form_msg = '<p class="error">|Erro ao enviar imagem.</p>';
```

```
return;
```

```
}
```

```
// Retorna o nome da imagem
```

```
return %nome_imagem;
```

```
} // upload_imagem
```

```
/**
```

```
* Paginação
```

```
*
```

```
* @since 0.1
```

```
* @access public
```

```
*/
```

```
public function paginacao () {
```

```
/*
```

Verifica se o primeiro parâmetro não é um número. Se for é um single

e não precisa de paginação.

```
*/
```

```
if ( is_numeric( chk_array( %this-lparametros, 0 ) ) ) {
```

```
return;
```

```
}
```

```
// Obtém o número total de notícias da base de dados
```

```
% query = %this-ldb-lquery(
```

```
'SELECT COUNT(*) as total FROM noticias '
```

```
);
```

```
% total = %query-lfetch();
```

```
% total = %total['total'];
```

```
// Configura o caminho para a paginação
```

```
% caminho_noticias = HOME_URI . '/noticias/index/page/';
```

```
// Itens por página
```

```
% posts_per_page = %this-lposts_por_pagina;
```

```
// Obtém a última página possível
```

```
% last = ceil(%total/%posts_per_page);
```

```
// Configura a primeira página
```

```
% first = 1;
```

```
// Configura os offsets
```

```
% offset1 = 3;
```

```
% offset2 = 6;
```

```
// Página atual
```

```
% current = % this->{parametros[1]} ? % this->{parametros[1]} : 1;
```

```
// Exibe a primeira página e reticências no início
```

```
if ( % current < 4 ) {
```

```
echo '<a href="% caminho_noticias% first"% first</a> ... ';
```

```
}
```

```
// O primeiro loop toma conta da parte esquerda dos números
```

```
for ( % i = ( % current - % offset1 ); % i < % current; % i++ ) {
```

```
if ( % i < 0 ) {
```

```
echo '<a href="% caminho_noticias% i"% i</a>';
```

```
// Diminui o offset do segundo loop
```

```
% offset2--;
```

```
}
```

```
}
```

```
// O segundo loop toma conta da parte direita dos números
```

```
// Obs.: A primeira expressão realmente não é necessária
```

```
for ( ; % i < % current + % offset2; % i++ ) {
```

```
if ( % i <= % last ) {
```

```
echo '<a href="% caminho_noticias% i"% i</a>';
```

```
}
```

```
}
```

```
// Exibe reticências e a última página no final
```

```
if ( % current <= ( % last - % offset1 ) ) {
```

```
echo ' ... <a href="% caminho_noticias% last"% last</a>';
```

```
}
```

```
} // paginacao  
  
} // NoticiasAdmModel
```

Este é um model bem grande, mas ele fará tudo o que precisamos.

Seus métodos todos fazem o seguinte:

`listar_noticias()` - Retorna um array com todas as notícias;

`obtem_noticia()` - Obtém uma única notícia. Também atualiza os dados caso algo seja enviado;

`insere_noticia()` - Insere uma notícia na base de dados;

`apaga_noticia()` - Apaga uma notícia da base de dados;

`upload_imagem()` - Faz o upload de imagens se alguma for enviada;

`paginacao()` - Cria uma paginação para apresentar as notícias na página inicial de notícias;

Os métodos estão bem comentados, portanto, leia os comentários e se ficar com dúvidas, basta questionar.

Agora vamos criar os views.

## Criando um view para mostrar notícias

Agora vamos criar um dos views que precisamos, o para a área administrativa.

**Lembre-se:** Views ficam dentro da pasta `views` separados por pastas. Nosso view estará dentro da pasta `noticias` com o nome de `noticias-adm-view.php` e `noticias-view.php`.

Veja seu código do arquivo `noticias-adm-view.php`.

## /views/noticias/noticias-adm-view.php

```
<?php
```

```
// Evita acesso direto a este arquivo
```

```
if ( ! defined('ABSPATH')) exit;
```

```
// Configura as URLs
```

```
% adm_uri = HOME_URI . '/noticias/adm/';
```

```
% edit_uri = % adm_uri . 'edit/';
```

```
% delete_uri = % adm_uri . 'del/';
```

```
// Carrega o m é todo para obter uma not í cia
```

```
% modelo-lobtem_noticia();
```

```
// Carrega o m é todo para inserir uma not í cia
```

```
% modelo-linsere_noticia();
```

```
// Carrega o m é todo para apagar a not í cia
```

```
% modelo-lform_confirma = % modelo-lapaga_noticia();
```

```
// Remove o limite de valores da lista de not í cias
```

```
% modelo-lsem_limite = true;
```

```
?|
```

```
<div class='wrap' |
```



```
<?php
```

```
// Mensagem de configuração caso o usuário tente apagar algo
```

```
echo %modelo-|form_confirma;
```

```
?|
```

```
<!-- Formulário de edição das notícias --|
```

```
<form method='post' action='' enctype='multipart/form-data' |
```

```
<table class='form-table' |
```

```
<tr|
```

```
<td|
```

```
Título: <br|
```

```
<input type='text' name='noticia_titulo' value='<?php
```

```
echo htmlentities( chk_array( %modelo-|form_data, 'noticia_titulo' ) );
```

```
?|' /|
```

```
</td|
```

```
</tr|
```

```
<tr|
```

```
<td|
```

```
Imagem: <br|
```

```
<input type='file' name='noticia_imagem' value='' /|
```

```
</td|
```

```
</tr|
```

```
<tr|
```

<td|

Data: <br|

<input type='text' name='noticia\_data' value='<?php

% data = chk\_array( % modelo-|form\_data, 'noticia\_data');

if ( % data && % data != '0000-00-00 00:00:00' )

echo date('d-m-Y H:i:s', strtotime( % data ) );

?|' /|

</td|

</tr|

<tr|

<td|

Autor: <br|

<input type='text' name='noticia\_autor' value='<?php

echo htmlentities( % \_SESSION['userdata']['user\_name'] );

?|' /|

</td|

</tr|

<tr|

<td|

Texto da not í cia: <br|

<textarea name='noticia\_texto' |<?php

echo htmlentities( chk\_array( % modelo-|form\_data, 'noticia\_texto' ) );

```
?|</textarea|
```

```
</td|
```

```
</tr|
```

```
<tr|
```

```
<td colspan='2'|
```

```
<?php
```

```
// Mensagem de feedback para o usu á rio
```

```
echo % modelo-|form_msg;
```

```
?|
```

```
<input type='submit' value='Save' /|
```

```
</td|
```

```
</tr|
```

```
</table|
```

```
<input type='hidden' name='insere_noticia' value='1' /|
```

```
</form|
```

```
<!-- LISTA AS NOTICIAS --|
```

```
<?php % lista = % modelo-|listar_noticias( ); ?|
```

```
<table class='list-table'|
```

```
<?php foreach( % lista as % noticia ): ?|
```

```
<tr|
```

```
<td|<?php echo % noticia['noticia_titulo'] ?|</td|
```

```
<td|
```

```
<a href='<?php echo % edit_uri . % noticia['noticia_id'] ?|' |
```

Editar

```
</a|
```

```
<a href='<?php echo % delete_uri . % noticia['noticia_id'] ?|' |
```

Apagar

```
</a|
```

```
</td|
```

```
</tr|
```

```
<?php endforeach; ?|
```

```
</table|
```

```
</div| <!-- .wrap --|
```

Este view faz todo o sistema de CRUD das notícias utilizando as ações do nosso model.

## /views/noticias/noticias-view.php

Veja agora como apresentaremos os dados para o usuário final (este é o view da ação index):

```
<?php
```

```
// Evita acesso direto a este arquivo
```

```
if ( ! defined('ABSPATH')) exit;
```

```
?|
```

```
<div class='wrap'|
```

```
<?php
```

```
// Número de posts por página
```

```
% modelo->posts_por_pagina = 10;
```

```
// Lista notícias
```

```
% lista = % modelo->listar_noticias();
```

```
?|
```

```
<?php foreach( % lista as % noticia ):?|
```

```
<!-- Título --|
```

```
<h1|
```

```
<a href='<?php echo HOME_URI ?|/noticias/index/<?php echo % noticia['noticia_id'] ?|' |
```

```
<?php echo % noticia['noticia_titulo'] ?|
```

```
</a|
```

```
</h1|
```

```
<?php
```

```
// Verifica se estamos visualizando uma única notícia
```

```
if ( is_numeric( chk_array( % modelo->parametros, 0 ) ) ): // single
```

```
?|
```

```
<p|
```

```
<?php echo % modelo->inverte_data( % noticia['noticia_data'] );?| |
```

```
<?php echo % noticia['noticia_autor'];?|
```

```
</p|
```

```
<p|
```

```
<img src=<?php
```

```
echo HOME_URI . '/views/_uploads/' . % noticia['noticia_imagem']; ?|'
```

```
</p|
```

```
<?php echo % noticia['noticia_texto']; ?|
```

```
<?php endif; // single ?|
```

```
<?php endforeach; ?|
```

```
<?php % modelo->paginacao(); ?|
```

```
</div| <!-- .wrap --|
```

Isso deverá gerar uma lista com as notícias, links para a página de uma notícia única e uma paginação em baixo.

Dentro da notícia outros campos serão apresentados, como imagem, autor, data e texto.

Veja um exemplo:

| HOME  | LOGIN | USER REGISTER | NOTÍCIAS | NOTÍCIAS ADMIN |
|---|-------|---------------|----------|----------------|
| <p>Título:<br/><input type="text"/></p> <p>Imagem:<br/><input type="button" value="Selecionar arquivo..."/> Nenhum arquivo selecionado.</p> <p>Data:<br/><input type="text"/></p> <p>Autor:<br/><input type="text" value="Admin"/></p> <p>Texto da notícia:<br/><input type="text"/></p> <p><input type="button" value="Save"/></p> |       |               |          |                |

Cadastro de notícias 1

|                   |  |
|-------------------|--|
| Título:           | <input type="text" value="é apenas um exemplo"/>                       |
| Imagem:           | <input type="button" value="Selecionar arquivo..."/> Chrysanthemum.jpg |
| Data:             | <input type="text" value="19/09/2014 08:45:12"/>                       |
| Autor:            | <input type="text" value="Admin"/>                                     |
| Texto da notícia: | <input type="text" value="Texto da notícia."/>                         |
|                   | <input type="button" value="Save"/>                                    |

Campos preenchidos 1

