

# COM222

## DESENVOLVIMENTO DE SISTEMAS WEB

Aula 13: Rotas PHP

# Conteúdo

- Rotas amigáveis
- Controlando rotas
- Bloqueio de acesso à arquivos da aplicação

# Rotas Amigáveis

- Rotas amigáveis seguem o seguinte padrão:
  - `www.site.com/rota/amigavel/`
- Enquanto rotas não amigáveis segue o seguinte:
  - `www.site.com/meu_arquivo.php`

Devemos usar esse recurso para esconder a estrutura da aplicação e também ajudam robôs de busca(crawlers e afins) a indexar o site.

O próprio google dá mais pontos a sites assim.

# Rotas Amigáveis

Veja como são muito mais fáceis de ler:

[www.site.com/produto/notebook/15/](http://www.site.com/produto/notebook/15/)

comparadas as não amigáveis

[www.site.com/produto.php?tipo=notebook&id=15](http://www.site.com/produto.php?tipo=notebook&id=15)

# Controlando Rotas: Configuração

O primeiro passo para começarmos a criar uma aplicação MVC que controla rotas amigáveis é modificar um arquivo do APACHE.

No linux o arquivo se encontra em:

**`/etc/apache2/apache2.conf`**

Iremos modificá-lo para que o módulo de reescrita do apache funcione e fazendo com que todas(quase) sejam redirecionadas para o arquivo index na raiz de nossa aplicação.

# Controlando Rotas: Configuração

Abra o arquivo (com permissão de root) e altere a seguinte parte:

...

```
<Directory /var/www/>
```

```
Options Indexes FollowSymLinks
```

```
AllowOverride All
```

```
Require all granted
```

```
</Directory>
```

...

Salve e habilite o módulo de reescrita:

```
sudo a2enmod rewrite
```

# Controlando Rotas: Configuração

Reinicie o apache:

```
service apache2 reload
```

Agora que já temos as configurações prontas vamos criar a aplicação. Vá até o diretório de sites do apache, para sistemas linux o padrão é:

```
/var/www/html/
```

Crie uma pasta MVC: Dentro dessa pasta crie um arquivo index.php:

```
<?php  
    echo $_SERVER['REQUEST_URI'];
```

# Controlando Rotas: Configuração

Crie também um arquivo .htaccess:

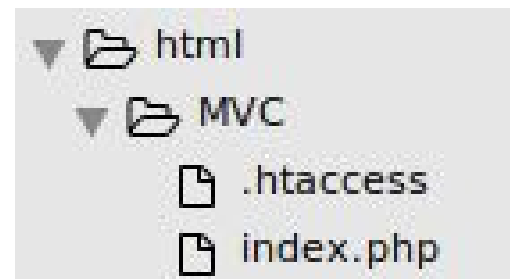
```
RewriteEngine on
```

```
RewriteBase /MVC/
```

```
RewriteCond %{REQUEST_FILENAME} !-f
```

```
RewriteCond %{REQUEST_FILENAME} !-d
```

```
RewriteRule ^(.*)$ index.php?/$1 [L,QSA]
```



Salve e faça acessos pelo browser como:

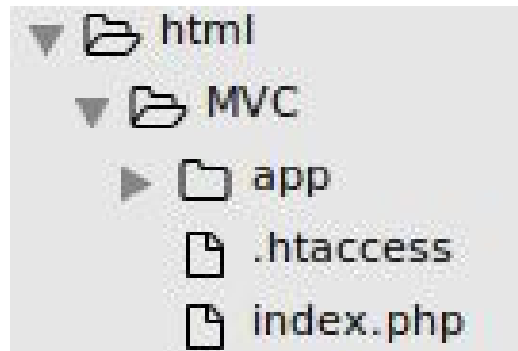
<http://localhost/MVC/uma/rota/qualquer/>

Veja que qualquer rota será direcionada a esse arquivo agora!



# Controlando Rotas: Tratamento

Agora que nosso **index.php** já recebe as rotas vamos começar a tratar a rota e direcioná-la. Crie uma pasta app.



Agora vamos editar **index.php**:

# Controlando Rotas: Tratamento

```
<?php  
// Iremos usar o define a seguir  
// para limpar a URL recebida  
// e direcionar para algum controller.
```

```
define('BASEURI', 'MVC');
```

```
// Esse define irá ser usado para  
// orientar a aplicação quanto ao  
// diretório da aplicação.
```

```
define('APPFOLDER','app');
```

**OBS: Não se esqueça de apagar o echo que usamos para testar!**

# Controlando Rotas: Tratamento

```
// A função 'ltrim' irá remover do inicio da string
// e 'trim' do inicio e do final da string
$url = trim($_SERVER['REQUEST_URI'],'/');

$url = ltrim($url,BASEURI);
// obs: se seu .htaccess não funcionou, remova index.php do inicio da string
// $url = ltrim(url,'index.php');
$url .= '/';

// Agora que já temos a url tratada.
// Ela terá o seguinte formato /Uma/Rota/
```

**Agora que a URL está tratada vamos usar um mecanismo muito útil do php para nos ajudar!**

# Controlando Rotas: Direcionando

```
spl_autoload_register(function ($classname){  
    if(include(APPFOLDER.'/controllers/'.$classname.'.php')) return;  
    if(include(APPFOLDER.'/models/'.$classname.'.php')) return;  
});
```

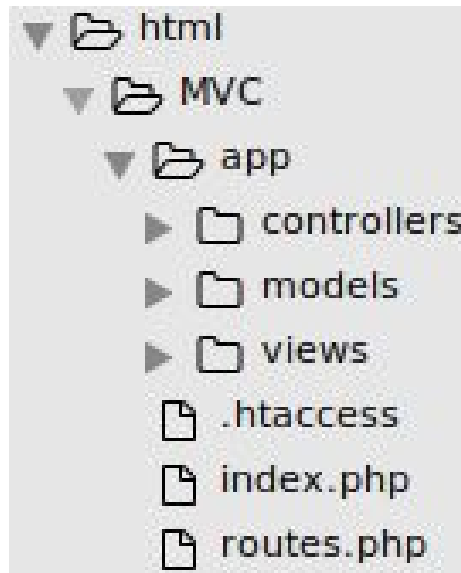
```
// A função de autoload irá ser chamada sempre que uma de nossas classes  
// for criada. A vantagem de usá-lo é que não precisaremos usar includes  
// sempre que precisarmos de algum arquivo de classe. Fique atento pois o  
// nome da classe e de seu arquivo devem ser os mesmos para funcionar.
```

**OBS: O parâmetro que estamos enviando é uma função anônima. É um recurso muito útil e é comum em códigos de JavaScript.**

# Controlando Rotas: Direcionando

Agora já temos a rota tratada e um mecanismo que irá carregar as classes automaticamente!

Crie as pastas: **models**, **views** e **controllers** e um arquivo na pasta raiz **routes.php** .



**OBS: Se você notou que o autoload não puxa views automaticamente, não se preocupe. Isso será feito de outra maneira.**

# Controlando Rotas: Direcionando

Vamos direcionar as rotas com **routes.php**:

```
<?php
```

```
// Aqui iremos determinar qual controller e qual método serão usados para  
// cada url. Iremos seguir o seguinte padrão Nome/Método que irá instanciar  
// um controller(classe) com 'Nome' e chamar seu 'Método'.
```

```
function route($url) {  
    $route['/'] = 'home/hello';  
    $route['/user/signin/'] = 'home/login';  
    $route['/user/signup/'] = 'home/createUser';  
    if($route[$url] != "")  
        return $route[$url];  
    die(include_once(APPFOLDER.'/views/templates/404.php'));  
}
```

**OBS: A última linha serve para o caso de uma rota inexistente ser recebida.  
Portanto crie o arquivo: MVC/app/views/templates/404.php**

# Controlando Rotas: Direcionando

Agora já temos uma função básica para nos dizer quem chamar, para realizar as chamadas adicione em **index.php**:

```
include_once('routes.php');  
$route = route($url);  
$route = explode('/', $route);  
// Após traduzirmos nossa rota para "controller/metodo" iremos transformar  
// essa string em um array, tomando como ponto de separação a '/'  
$controller = $route[0];  
$method = $route[1];  
$obj = new $controller;  
$obj->$method();  
// Podemos melhorar essa estrutura usando call_user_func_array()
```

**OBS: A função explode transforma uma string em um array de acordo com o ponto de divisão (primeiro parâmetro)**

# Controlando Rotas: Direcionando

Crie uma classe na pasta controllers **home.php**:

```
class home {  
    public function hello() {  
        echo 'PAGINA INICIAL';  
    }  
}
```

Veja que a classe é criada e o método chamado sem includes!

Agora vamos lidar com o carregamento das views: crie na pasta controllers o arquivo **Controllers.php**.

**OBS: Faça um teste para os MODELS! Crie um arquivo php na pasta models e instancie dentro da função hello!**



# Controlando Rotas: Estruturando MVC

Para chamar nossas views iremos criar uma classe abstrata que, também será responsável por carregar bibliotecas, então crie a pasta:

**MVC/app/lib/**

```
<?php
```

```
abstract class Controller {  
    public function loadView($file,array $args = NULL) {  
        if($args !== NULL) extract($args);  
        include_once(APPFOLDER.'/views/'.$file.'.php');  
    }  
    public function loadLib($file) {  
        include_once(APPFOLDER.'/lib/'.$file.'.php');  
    }  
}
```

# Controlando Rotas: Estruturando MVC

A função **extract** usada, transforma um array em variáveis da seguinte forma, enviando:

```
extract(array('nome'=>'Baldochi' , 'professor' => 'COM222' , 'idade' => 20))
```

Teremos:

```
$nome; // Com valor 'Baldochi'
```

```
$professor; // Com valor 'COM222'
```

```
$idade; // Com valor 20
```

Através disso podemos passar múltiplos valores com apenas dois parâmetros para nossas views!

# Controlando Rotas: Estruturando MVC

Altere sua classe **home.php** para herdar **Controller** e faça um teste.

```
class home extends Controller
{
    public function hello() {
        $this->loadView('home_page',array('title'=>'Página
Inicial','message'=>'Bem vindo!'));
    }
}
```

Não se esqueça de criar o arquivo da página:

**MVC/app/views/home\_page.php**

# Controlando Rotas: Estruturando MVC

```
<!DOCTYPE html>
<html>
  <head>
    <title><?=$title?></title>
  </head>
  <body>
    <h1><?=$message?></h1>
  </body>
</html>
```

# Controlando Rotas: Estruturando MVC

Vamos agora criar uma classe abstrata para nossos Models, crie o seguinte arquivo:

## **MVC/app/models/Model.php**

```
<?php
abstract class Model{
    public function loadLib($file) {
        include_once(APPFOLDER.'/lib/'.$file.'.php');
    }
}
```

Vamos agora testar nossos models. Crie um model **MVC/app/models/user.php**

# Controlando Rotas: Estruturando MVC

```
class user extends Model{  
    public function save($uname,$pass) {  
        $pass = hash('sha512',$pass);  
        // $this->loadLib('DAO');  
        // $db = new DAO;  
        $value='Cadastrado!';  
        // unset($db);  
        echo 'Olhe o que fiz com sua senha:'. $pass;  
        return $value;  
    }  
    public function login($uname,$pass) {}  
}
```

**OBS: A função hash é usada para criptografar. Existem vários algoritmos prontos para usarmos. Nesse caso estamos usando SHA512.**

**[http://php.net/manual/pt\\_BR/function.hash.php](http://php.net/manual/pt_BR/function.hash.php)**

# Controlando Rotas: Estruturando MVC

Crie um arquivo para usuários realizarem Login:

```
<!-- login.php -->
```

```
<h1>LOGIN</h1>
```

```
<form method="POST" action="">
```

```
  <label>User Name:</label><br><input type="text" name="uname"><br>
```

```
  <label>Password</label><br><input type="password" name="pass"><br>
```

```
  <input type="submit" value="Login">
```

```
</form>
```

```
<a href="/MVC/">Página Inicial</a>
```

Com a rota já definida (você copiou o arquivo routes.php), olhe e crie o método em seu controller que carregue essa pagina.

# Controlando Rotas: Estruturando MVC

```
public function login() {  
    $rm = $_SERVER['REQUEST_METHOD'];  
    if($rm == 'GET') {  
        $this->loadView('login');  
    }  
    else if($rm == 'POST') {  
        $usr = new user;  
        echo $usr->save($_POST['uname'],$_POST['pass']);  
    }  
}  
else $this->loadView('templates/404');  
}
```

**Obs: Teste a página e veja o resultado. Faça o método createUser em home e também a página a ser carregada.**



# Bloqueio de Acesso

Agora já temos uma estrutura básica para lidar com rotas e chamar controllers para processá-las.






Mas nossa aplicação ainda apresenta algumas vulnerabilidades. Acesse a seguinte url:

**`http://localhost/MVC/app/`**

Note que nosso .htaccess não consegue direcionar todas urls. Um usuário malicioso poderia se aproveitar para ver a estrutura de nossa aplicação!

# Bloqueio de Acesso

## Index of /MVC/app

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 <a href="#">Parent Directory</a>		-	
 <a href="#">controllers/</a>	2017-04-21 06:16	-	
 <a href="#">lib/</a>	2017-04-22 21:23	-	
 <a href="#">models/</a>	2017-04-22 23:07	-	
 <a href="#">views/</a>	2017-04-22 23:33	-	

*Apache/2.4.25 (Ubuntu) Server at localhost Port 80*

Uma forma muito simples de prevenir esse tipo de vulnerabilidade é criar um arquivo index em cada pasta. Dessa forma ele será aberto ao invés de exibir nossa estrutura.

# Bloqueio de Acesso

Crie um arquivo index.php para cada pasta de sua aplicação com o seguinte código:

```
<?php  
header('Location: http://localhost/MVC/NotFound/');  
exit();
```

A partir de agora, quando algum usuário acessar a pasta ele será redirecionado para uma URL inexistente, o que irá abrir a página 404.

**Obs: Existem formas de corrigir esse problema usando o APACHE.**

# Bloqueio de Acesso

Vamos agora tentar fazer acesso direto a um arquivo (não a uma pasta) . Acesse:

<http://localhost/MVC/app/controllers/Controller.php>

O resultado é uma página em branco.

Adicione um **echo 'Executado!';** no fim do arquivo e acesse novamente.

Vamos consertar isso, adicione no início dos arquivos a seguinte linha:

```
defined('BASEURI') or die('Acesso Negado!');
```

# Bloqueio de Acesso

Agora os arquivos também não podem ser acessados diretamente. Pois o script será morto quando descobrir que **BASEURI** não foi definida!

Com essa estrutura básica podemos criar aplicações grandes e fáceis de administrar.

Essa estrutura pode ser evoluída posteriormente, por esse caminho criaremos um **framework**.

Entretanto para o desenvolvimento de projetos, recomenda-se o uso de um já existente.

# Exercício

Prazo: Uma semana

30

1. Crie um sistema com as seguintes funções

- Login
- Cadastrar usuário

Caso o login ou o cadastro falhe, redirecione para uma página de falha. Defina rotas amigáveis para as operações.

2. Adicione uma biblioteca na pasta **lib** para lidar com o mysql (**Dica: Use a feita na aula**)

3. TODAS operações de banco devem ser feitas em **models**

4. TODO código HTML deve ficar em arquivos view

# Exercício

Prazo: Uma semana

31

5. A senha deve ser criptografada com a função **hash** do php.

6. Bloqueie todo acesso a arquivos e pastas da aplicação.

Obs: Assunto do Email: **Exercício - MVC Routes**

Corpo do Email: **nome-matricula**

Envie um arquivo **.zip**

Dica: procure sobre \$mysli->error e \$mysli->errno para lidar com problemas de registro e login

Codigo SQL no próximo **SLIDE**

# Exercício

Prazo: Uma semana

32

```
create database MVC;
```

```
create table if not exists user(  
  id int auto_increment,  
  username varchar(50) unique,  
  password blob not null,  
  primary key(id)  
);
```

-- Como a senha é um valor hash seu campo  
-- é do tipo BINARIO (BLOB)