MinúsculoPHP – MVC Framework

Utiliza algumas classes do http://flourishlib.com/

INSTALAR O MINÚSCULO PHP MVC (minPHP)

1. Abrir “config.php” e modificar as URLs e o nome do banco de dados
2. Abrir o “.htaccess” e modificar a URL
3. Executar o SQL Inicial em “/documentação” que criará
   1. Pessoa\_fisica
   2. Usuário
   3. Modulo
   4. User\_tipo
   5. User\_permissao
   6. Token

SOBRE O SISTEMA MVC – aqui não existe o controller e o model passa a ser o motor

1. Funcionamento básico:
   1. Inicia o “.htaccess” e abre o index.php?exe=controlador/acao/parâmetros
   2. Instancia AppMVC que separa o controlador, a ação e os parâmetros
   3. Inclui o arquivo “/controllers/[controlador]-controller.php” se existir
   4. Inclui o “home-controller.php” caso não foi passado nenhum controlador
   5. Inclui o “PG\_404” caso o controlador passado não exista (ver “config.php”)
   6. Instancia a classe com o mesmo nome do controlador (só letras), ou seja, todo arquivo de controlador, deve ser uma classe com o mesmo nome, herdando de “MainController”: class ControladorController extends MainController
   7. Chama o método “AÇÃO” do controlador. Caso não exista tenta chamar o método “INDEX”. Caso “INDEX” também não exista, chama “PG\_404”.

Sistema baseado no MVC: (Model View Controller) e JSON com requisições AJAX.  
As requisições tem o padrão “/controlador/acao/parametro1/parametro2/...”  
Também pode-se usar query-string sem problemas: “/control/acao?q=algo”

Utilizarei o MVC como padrão de projeto, sendo o

* MODEL: as classes em PHP responsáveis pela manipulação dos dados no banco de dados, o CRUD (Create, Read, Update e Delete), etc
* VIEW: o que é exibido ao usuário, ou seja, as páginas de exibição em HTML.
* CONTROLLER: chama o VIEW e o MODEL correspondente.

CONTROLLER -> MODEL -> VIEW

Exemplo:

user/new: POST {nome=vinicius, nascimento=04/04/1989}  
user/update: POST {id=2, nome=vini}  
user/delete: POST {id=3}  
user/load: POST {id=3}  
user/list: QUERY

Tudo pensado como se fosse uma API.  
O sistema é literalmente uma API, utilizada por ele mesmo!

O diretório **VIEWS** contém todo o front-end;  
O diretório **MODELS** contém todo o back-end;  
O diretório **CONTROLLERS** contém os controladores que chamam os views e models corretos e as ações;

Exemplo tutorial: <http://www.tutsup.com/2014/09/17/mvc-em-php-parte-1/>

Padrão de projeto das classes:  
<http://www.tutsup.com/2014/08/10/criar-classes-em-php/>”

CÓDIGO LIMPO:   
<http://www.tutsup.com/2014/05/31/padroes-de-codificacao-wordpress/>

FUNCIONAMENTO BÁSICO

Já cria o Login com permissões para cada módulo, tudo definido no banco  
Já existe as classes de banco de dados, form e CRUD.

Todas as inclusões ou alterações são feitas com a classe FORM e são enviados um array na forma:

* ARR [ CAMPO ] [ VALUE ] = valor do campo
* ARR [ CAMPO ] [ DBFIELD ] = nome do campo no banco
* ARR [ CAMPO ] [ DBINT ] = diz se é campo numérico ou não, para ter ou não aspas

Isso para fazer o código ficar enxuto e mais objetivo e genérico.

PARÂMETROS BÁSICOS

1. Todas as tabelas necessitam ter o campo primário Id
2. Todas as tabelas necessitam ter o campo bool\_excluido

**PERMISSÕES** - próximo à ideia do Linux: RWX

O MainController é responsável por verificar para cada ação.

* RWX se torna **CRU** (inserir, ler, alterar/apagar)
* 100 = 4: pode apenas inserir
* 111 = 7: pode inserir, ler e alterar

Para cada ação de um controller, existe uma permissão exigida.

Cada módulo está registrado no banco de dados, e cada ação dos módulos está registrado em permissões.php

**IMPORTANTE**: módulos não registrados no banco são tratados como módulos públicos (acessíveis a qualquer usuário, logado ou não). Caso um módulo esteja registrado no banco, uma ação dele pode ser tratada como pública, caso (em permissoes.php) não tenha sido definido aquela ação para este módulo, ou ela tenha sido definida como 0;

Todo arquivo de CONTROLLER pode ter o método sem\_permissao()

1. Você pode colocar permissão para uma ação de um módulo específico em $\_PERMS[]
2. Você pode colocar permissões padrões para ações para qualquer módulo em $\_PERMS\_ACAO[]
3. A prioridade é dada primeiro para uma ação de um módulo, se ela não existe então verifica se existe para aquela ação genericamente.
4. Se o usuário não tem permissão definida no banco para um módulo, então verifica a permissão padrão daquele módulo e aplica.
5. Caso um módulo não tenha sido registrado no banco (exemplo de módulos públicos) este é visto como um módulo público, e torna-se acessível a qualquer usuário, logado ou não.

PADRÃO DE RESPOSTAS JSON

* SUCESSO: '{"success": true, “msg”: ”algo”, “redirect”:”URL”}';
* ERRO: '{"success": false, "errors":["erro 1", "erro 2"]}';
* RESULTADOS: '{"success": true, “total”: 4, “results”: [ { }, { } ] }';

COMPLEMENTO

* Minusculo MinMakeCRUD:

1. Utiliza semantic-ui em views/includes/libs
2. Utiliza jQuery em views/includes/libs
3. Cria um sistema de CRUD automaticamente a partir de uma tabela do banco

INCLUDES

Todos os includes de arquivos devem conter o <?php echo ABSPATH; ?> antes do endereço para não ter problemas nas inclusõe.

SUGESTÃO: usar sempre arquivos de cabeçalho com todas as inclusões para todas as views.

SEGURANÇA

Todo arquivo de view deve conter

if ( !defined('ABSPATH')) exit;

Antes de qualquer código, para evitar que seja acessado diretamente pelo endereço físico.