Modelo de Desenvolvimento Software

Tema Wordpress

Marco Túlio Bueno Vieira

Eduardo Humberto Resende Guimarães

Introdução.

Primeiramente explicaremos por qual motivo escolhemos a criação de um tema para o Wordpress que conterá as funções necessárias para que nosso sistema, que vamos intitular SocialDB, trabalhe com todas as funcionalidades e que também o usuário tenha uma boa experiência com a proposta.

Foi escolhido o desenvolvimento de um tema, pois esta opção da ao usuário uma segurança e uma fácil administração de seu site. Exemplificando, o usuário final que é na maioria das vezes leigo no assunto wordpress terá que baixar o tema e instalar o nosso tema, sem precisar fazer nenhuma alteração em relação a aparência e problemas de compatibilidade com outras funcionalidades, forneceremos uma opção completa, controlaremos a experiência do usuário onde ele não precisará se preocupar com o funcionamento das coisas apenas a customização de seus acervos.

Desenvolvimento do Tema

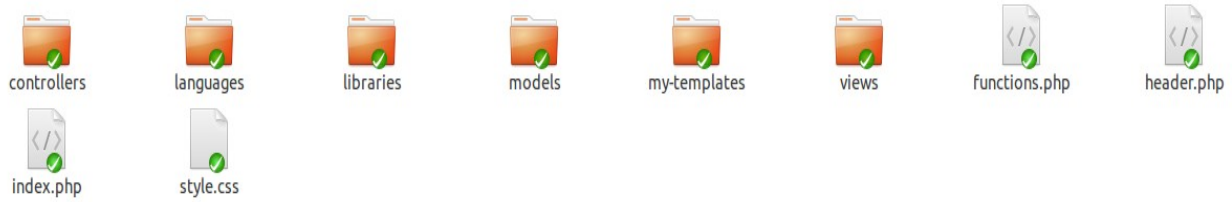
Para o nosso desenvolvimento usamos uma referência bem simples, o próprio wordpress fornece uma documentação de como desenvolver o tema. Seguimos esta documentação para criar o mesmo em relação aos itens necessários para que fosse possível a interpretação do wordpress sabendo que trata-se de um tema.

Trabalhamos também com o conceito de MVC (Model, View, Controller) onde incluímos este conceito ao desenvolvimento de nosso tema. Trabalhamos também com boas práticas de programação, organização de funções, nomenclatura e formatação.

A partir deste momento demostraremos estas boas práticas e a utilização do MVC em nosso tema, este documento será ilustrado com um tema criado com a funcionalidade de um cadastro e a utilização de bibliotecas de terceiros para que seja demonstrado como será o desafio que enfrentaremos na criação deste tema.

Lembramos também que este documento estará em constante modificação de acordo com as definições que virão.

Estrutura de Pastas



Pelo que vemos acima está é a pasta ROOT de nosso tema com os arquivos que serão necessários para a ativação e desativação de nosso tema.

Implementando o nosso MVC podemos ver que já estamos trabalhando com a divisão de pastas, adaptando a estrutura do tema com a estrutura de um MVC.

Controllers = Esta pasta ficara as classes de Controladores, estas são responsáveis por receber as ações do usuário que vem da interface de nosso sistema e mandar esta solicitação para a classe responsável que efetuará está ação. Basicamente controladores são as classes que se comunicam com a interface visual do sistema, são responsáveis por processar a ação requerida pelo usuário e enviar para que o sistema possa trabalhar na mesma.

Languages = Esta pasta ficará os arquivos de tradução, onde cada instalação do wordpress pode ter idiomas diferentes, existirá o padrão e também o português.

Libraries = Nesta pasta ficarão alocados os arquivos de CSS as fontes utilizadas em nosso sistema e os JS JavaScript's globais do sistema. (Breve iremos detalhar a organização destes JS)

Models = As classes de Model ficarão aqui subdivididas em pasta, estas classes são responsáveis por todas as funções de nosso sistema, toda nossa regra de negócio se encontra nestas classes. As classes Model recebem a solicitação das classes Controller e processam o pedido, executam a tarefa desejada.

My-Templates= Esta pasta ficará os templates que o nosso tema irá criar, esta pasta é uma recomendação da própria documentação do wordpress, por exemplo caso nosso tema tenha a possibilidade de criar uma página de Contact, dentro desta pasta My-Templates deverá ter um arquivo com a estrutura desta página a alocação do formulário de contato e as infomrações necessárias de organização.

Views = Esta pasta ficam alocadas as classes View de nosso sistema, estas classes tem a função de apresentação para o usuário. Nas views que criamos a estrutua visual, botões, tabelas e outros.

Controllers

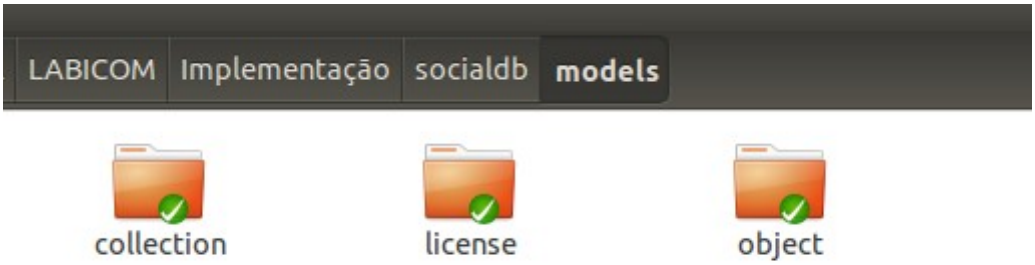
Dentro desta pasta Controllers ficarão nossas classes de controladores, subdivididas nos módulos de nosso sistema. Estas classes serão nomeadas todas com o sufixo _controller. Assim quando o desenvolvedor efetuar uma busca saberá o que realmente estará nesta classe e como proceder para criar outras e dar a manutenção necessária.

A figura acima mostra como é disposto os módulos dentro da pasta de controllers, na pasta License estará os controladores responsáveis pelo módulo de Licenças do sistema.

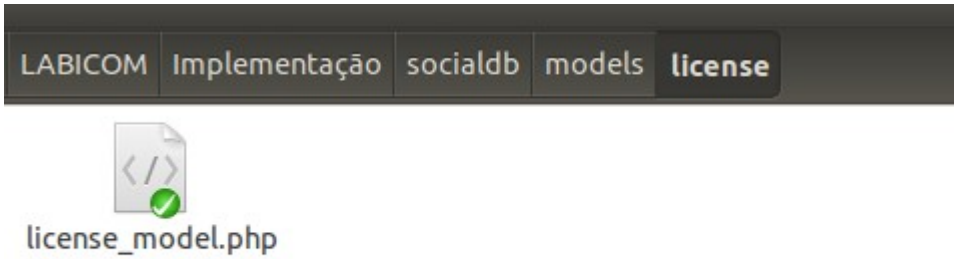
A imagem acima mostra como será nomeado as classes de controller.

Models

Na pasta models estarão dispostos como a pasta controllers, subdivididas nos módulos de nosso sistema. A classe model é responsável por executar a regra de negócio, por efetuar a tarefa requisitada pelo usuário na interface, a nomenclatura das classes de model serão parecidas com as de controllers, o sufixo irá ser alterado para _model. Para mais uma vez facilitar o trabalho do desenvolvedor.



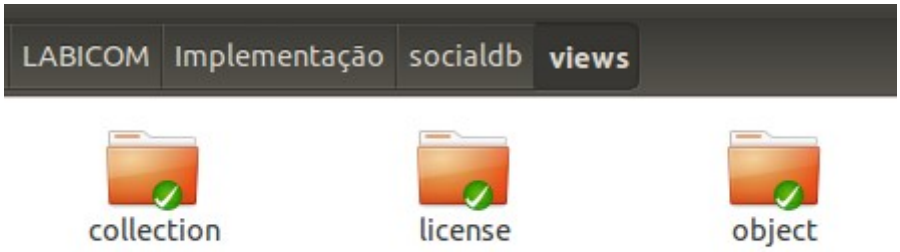
Como mostrado acima, a pasta models tem as mesmas subdivisões de pastas que a pasta controllers assim sendo de fácil identificação e localização destas classes.

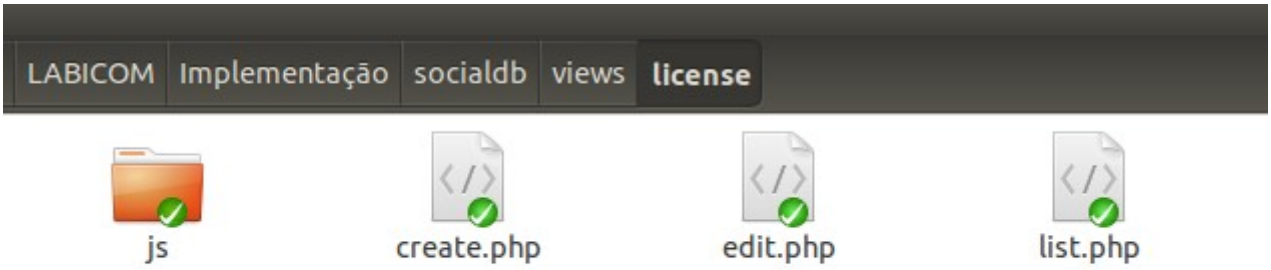


' A imagem acima mostra como será nomeado a classe de model.

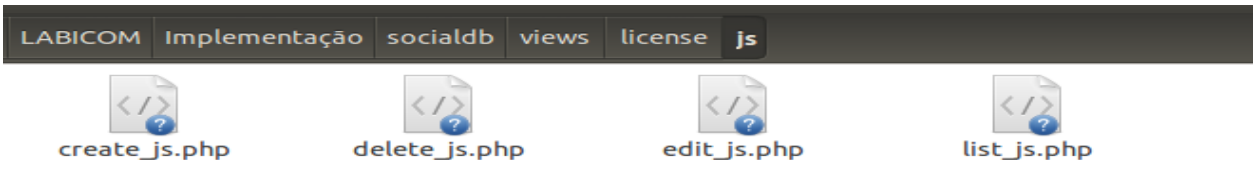
Views

Na pasta views estarão dispostos como a pasta controllers, subdivididas nos módulos de nosso sistema. As classes de views são definitivamente a visão, onde informamos a estrutura da página, com as cores, tabelas, botões. Está classe é realmente o que o usuário terá acesso e verá, nela serão feitas as solicitações para os controllers que passarão para os models executarem a ação desejada pelo usuário.





Como mostrado acima, a pasta views tem as mesmas subdivisões de pastas que a pasta controllers assim sendo de fácil identificação e localização destas classes. Apasta acima mostra detalhadamente como é a subdivisão de cada módulo, mas esta pasta view tem uma particularidade. Dentro de cada pasta de módulo dentro da pasta das views, existe a pasta JS onde os JavaScripts necessários para a funcionalidade daquela view são alocados. Lembrando que estes arquivos são para funções específicas de cada módulo. Pois Javascript de terceiros e globais estarão localizados na pasta Libraries, onde poderão ser acessados de qualquer view. Assim sendo fácil a reaproveitação destes arquivos js. A imagem abaixo mostra bem detalhado como será trabalho essa pasta de JavaScript dentro de cada módulo.



Classe Controller

Como exemplificado anteriormente sobre a classe controle, vamos mostrar aqui como é a parte de implementação da mesma.

A classe controle deve obrigatoriamente ter uma chamada para a classe model, onde serão executados todas as regras de negócio. E também deve conter uma chamada para cada método que se encontra implementado no módel. Veja abaixo que esta classe extende de Controller que é um controlador genérico que conterá funções que serão necessárias para todos os controladores.

```php
 require_once(dirname(__FILE__).'../../../models/license/license_model.php');
 require_once(dirname(__FILE__).'../../../controllers/general/general_controller.php');


class LicenseController extends Controller{
        public function operation($operation,$data){
                $license_model = new LicenseModel();
                switch ($operation) {
                                case "add":
                                                echo $license->add($_POST['license_name'],$_POST['license_content']);
                                                break;
                                        case "edit":
                                                echo $license->edit($_POST['id']);
                                                break;
                                        case "remove":
                                                $license->remove();
                                                break;
                                        case "list":
                                                echo $license->list_licenses();
                                                break;
                }
        }
}
 /*
* Controller execution
*/
if($_POST['operation']){
     $operation = $_POST['operation'];
     $data = $_POST;
}else{
    $operation = $_GET['operation'];
    $data = $_GET;
}

$license_controller = new LicenseController();
echo $license_controller->operation($operation,$data);


?>
```

Classe Model

As classes model como explicado anteriormente são responsáveis por toda a regra de negócio de nosso sistema. Então ela precisa ter as funções todas implementadas.

A classe model necessita que seja incluído algumas funções padrões do

wordpress por isso no começo da classe existe a chamada para estas páginas.

```php
<?php
include_once ('../../../../../wp-config.php');
include_once ('../../../../../wp-load.php');
include_once ('../../../../../wp-includes/wp-db.php');


class LicenseModel{
        public function add($name,$content){
                $post = array(
                        'post_title' => $name,
                        'post_content' => $content,
                        'post_status' => 'publish',
                        'post_type' => 'licenses'
        );
                $post_id =  wp_insert_post($post);
            return $this->return_License($post_id);
        }



        public function edit($object_id){
             return $this->return_License($object_id);
        }



        public function return_License($post_id){
                $array_json = [];
                $post = get_post($post_id);
                $array_json['license_name'] = $post->post_title;
                $array_json['license_content'] = $post->post_content;
                $array_json['license_id'] = $post->ID;
                return json_encode($array_json);
        }

       public function list_licenses(){
                $list_licenses = [];
                global $wp_query;
                $args = array( 'post_type' => 'licenses','posts_per_page'=> 10) ;
                query_posts( $args );
                        if (have_posts()) :
                                while (have_posts()) :
                                        the_post();
                                        $list_licenses[] = $this->return_License(get_the_ID());
                                endwhile;
                        endif;
                return utf8_encode(json_encode($list_licenses));
        }

}
?>
```

Classe View

As views ficam a visão que o usuário final possui, então é uma página php que possui os formulários os inputs e todos os itens necessários para que seja possível a utilização por parte do usuário. Como as views são dividias em várias colocaremos aqui o CRUD básico exemplificado.

View List

```
<?php
include_once ('../../../../../wp-config.php');
include_once ('../../../../../wp-load.php');
include_once ('../../../../../wp-includes/wp-db.php');

?>
        <div id="list_post">
              <div class="post">
                              <table class="table table-hover">
                                      <thead>
                                            <tr>
                                              <th><?php _e('License Name'); ?></th>
                                              <th><?php _e('License Description'); ?></th>
                                              <th><?php _e('Edit'); ?></th>
                                              <th><?php _e('Delete'); ?></th>
                                            </tr>
                                      </thead>
                                      <tbody>
          <?php
               global $wp_query;
                       $args = array( 'post_type' => 'ideas','posts_per_page'=> 10) ;
                       query_posts( $args );

               if (have_posts()) : while (have_posts()) : the_post();?>
                                    <tr>
                                              <td><?php the_title(); ?></td>
                                              <td><?php the_content(); ?></td>
                                              <td><input type="hidden" class="post_id" name="post_id" value="<?= get_the_ID() ?>"><a href="#" class="edit"><span
class="glyphicon glyphicon-edit"></span></a></td>
                                              <td><input type="hidden" class="post_id" name="post_id" value="<?= get_the_ID() ?>"><a href="#" class="remove"> <span
class="glyphicon glyphicon-remove"></span></a></td>
                                            </tr>
               <?php endwhile; endif; ?>
                                      </tbody>
                  </table>
        </div>
            <script>
               $(function(){

                console.log('here');

                });
           </script>
```

View Edit

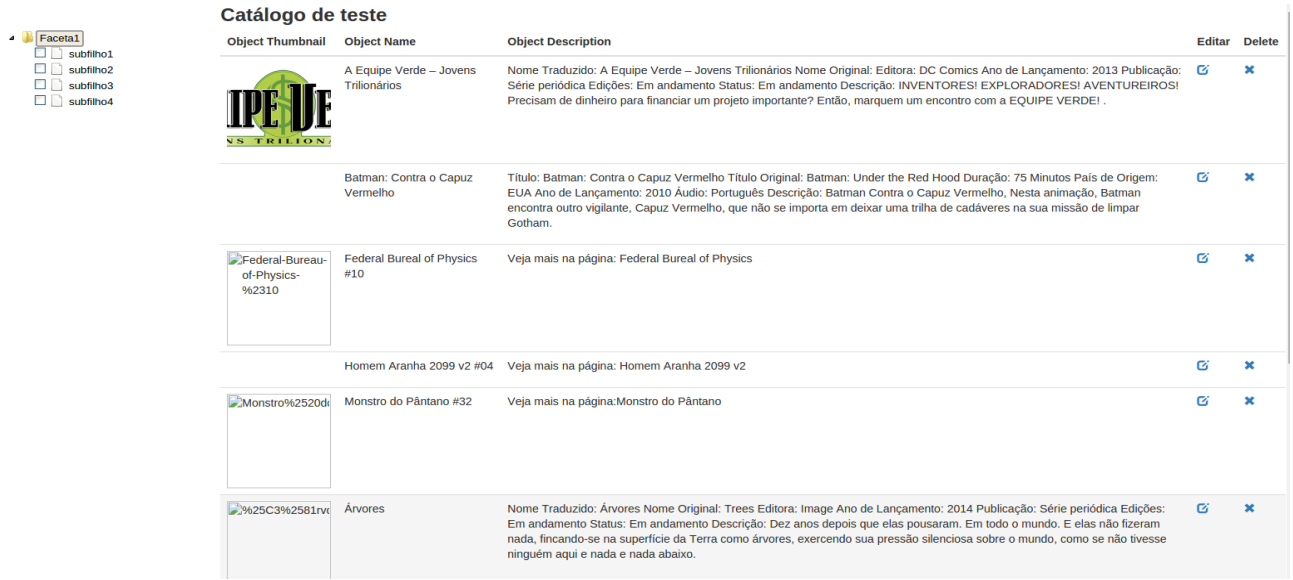
```
<?php
include_once ('../../../../../wp-config.php');
include_once ('../../../../../wp-load.php');
include_once ('../../../../../wp-includes/wp-db.php');
?>
<form  id="submit_form" style="display:none">
           <div class="form-group">
                 <label for="object_name"><?php _e('Object name'); ?></label>
                 <input type="text" class="form-control" id="object_name" placeholder="<?php _e('Object name'); ?>">
           </div>
           <div class="form-group">
                 <label for="object_name"><?php _e('Object Content'); ?></label>
                 <input type="text" class="form-control" id="object_content" placeholder="<?php _e('Object Content'); ?>">
           </div>
           <input type="hidden" id="src" name="src" value="<?= get_template_directory_uri() ?>">
           <input type="hidden" id="operation" name="operation" value="add">
           <button type="submit" id="submit" class="btn btn-default"><?php _e('Submit'); ?></button>
</form>
```

View Create

```php
<?php
include_once ('../../../../../wp-config.php');
include_once ('../../../../../wp-load.php');
include_once ('../../../../../wp-includes/wp-db.php');
?>
<form  id="submit_form">
          <div class="form-group">
                <label for="License_name"><?php _e('License name'); ?></label>
                <input type="text" class="form-control" id="license_name" placeholder="<?php _e('License name'); ?>">
          </div>
          <div class="form-group">
                <label for="License_name"><?php _e('License Content'); ?></label>
                <input type="text" class="form-control" id="license_content" placeholder="<?php _e('License Content'); ?>">
          </div>

          <input type="hidden" id="operation" name="operation" value="add">
          <button type="submit" id="submit" class="btn btn-default"><?php _e('Submit'); ?></button>
</form>
```

Exemplo Tema SocialDB



Qual a necessidade da criação de um MDS?

Uma das grandes dificuldades dos sistemas de hoje em dia é a manutenção do código, um bom exemplo que podemos dar é o seguinte: Um desenvolvedor está em uma empresa fez um trabalho de anos codificando de uma forma a qual somente ele entende, este mesmo acaba saindo da empresa. Qual a dificuldade do seu sucessor de entender o código, resolver problemas, e até mesmo melhorar o que já havia sido feito. Está é a grande ideia deste MDS para que além de tudo haja um padrão que possa ser seguido, que seja de fácil entendimento e que possa ser reaproveitado futuramente em outros projetos os quais vieram a ser desenvolvidos.

O que teremos neste modelo?

1. Nomenclatura de funções.
2. Organização de funções.
3. Organização de comentários.

1) Nomenclatura de funções.

Toda as funções geradas, relembram uma ação, CONSULTAR, EXCLUIR, INSERIR, GERAR. Assim sendo ideal seria que as funções que executam ação iniciem sempre com um verbo, lembrando que por padrão iremos adotar os nomes das funções em inglês, e o padrão também de funções o qual o Wordpress atualmente segue.

Errada

```
function addLicense($post_id,$user
```

Correta

```
function add_license($post_id,$user)
```

2) Organização de funções.

Para que fique de uma maneira mais fácil o entendimento do decorrer do método, a cada mudança de ação que é bem genérica cria-se uma nova função. Assim a função inicial não ficará grande e as outras poderão ser reaproveitadas. Apenas em casos bem específicos que se gera uma função longa sem divisão. Vamos abaixo para um exemplo de casos incorretos e de como resolver este problema. Utilizando a mesma função que foi citada no caso anterior.

Errada

```
function addLicense($post_id, $user){
...
Dentro da função um código que faz a inclusão.
....
Dentro da função um código que faz a exclusão.
....
```

Correta

Na forma correta existiria no mínimo 3 funções as quais seriam divididas de uma forma que poderiam ser reaproveitadas.

```
function add_license($post_id,$user){
...
Dentro uma chamada para a função de add() que já foi criada.
...
Dentro uma chamada para a função de remove() que já foi criada
```

E assim sucessivamente. Sempre chamando funções que já foram criadas (funções genéricas) somente algo bem especifico que será inserido aqui dentro da codificação da função

3) Organização de comentários.

Toda e qualquer função deverá ser comentada, com algumas informações bem simples. No comentário colocamos, o que esta função faz, quais são os atributos que ela recebe e se caso exista algum retorno também exemplificar qual seria o retorno. O autor desta função para que caso tenhamos alguma dúvida ser possível contatar o autor. Segue abaixo exemplo de um comentário errado e de um comentário correto. Lembrando que adotamos a forma de tratar com comentários em duas línguas, ou seja colocamos uma parte em inglês e duplicamos o comentário em português também. Abaixo exemplificamos apenas uma forma

Errado

```
/********** FUNÇÃO QUE GERA AS LICENSAS**********/

function add_licence($post_id, $user){
```

Correto

```
/* Função add_license() */
/* Recebe (x,x,x) */
/* Executa a criação de uma contestação que significa ..... */
/* Autor: Marco Túlio */
```

Lembrando que este modelo estará em grande alteração, sendo que após a definição de qual padrão iremos utilizar, será inserido aqui as subdivisões de pastas, arquivos de configuração organização entre outros.

Este modelo está aberto a sugestões para que possa ficar de fácil entendimento e que todos possam seguir para que haja uma grande evolução na forma de codificação.