

Elias Junior

Zend framework2

Passo a passo para iniciante

Resumo

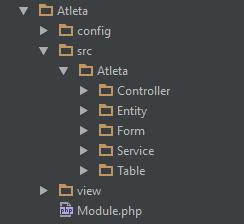
O Framework é uma facilidade da linguagem, onde ele possui alguns métodos já prontos. Ou seja, tudo que é feito no zend, pode ser feito no PHP estruturado normal, porém com muito mais dificuldade. O zend 2 é uma evolução do zend 1, logicamente, mas nele iremos utilizar o modo de desenvolvimento HMVC.

**Entendendo um pouco do Zend2.**

Através deste pequeno resumo ficará um pouco mais simples entender a estrutura Zend Framework 2. Este é um manual simples para passar um pouco direcionamento no entendimento da estrutura do ZF-2. Sabe-se que como é uma ferramenta nova, duvidas podem surgir a qualquer momento. Então vamos lá!

**Vamos começar entendendo um pouco da estrutura.**

**A estrutura do Zend Framework 2 é em modelo HMVC, mas o que é HMVC?**

* HMVC (Hierarquical Model-View-Controller) é uma evolução do MVC (Model View Controller) No MVC tínhamos todas as pastas juntas, já no HMVC, temos uma Hierarquia nos módulos.
* No HMVC, cada pacote (Model – View-Controller são pacotes), funcionam de forma independente, porém, conversando entre si, assim como na imagem abaixo.
* Essa é a estrutura de um modulo. Agora, vamos entender um pouco mais como funciona cada pasta, e arquivos, dessa estrutura. Usaremos como base, o cadastro de Atletas. Porém, toda a informação, poderá ser levada para outros módulos.

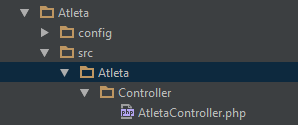
**Atletla/Config.**



* O modulo.config.php, terão todas as informações de rotas. Porém, essa parte ainda não foi explorada no projeto, então inicialmente, ela é padrão. Quando for criar um novo módulo, o mesmo padrão deverá ser empregado e cuidado com a nomenclatura, pois o menor detalhe pode fazer diferença.

**/Atletla/src/Atleta**

* Agora, falaremos da pasta “src”, que contem praticamente toda a programação back-end (Motor da Aplicação).



* Dentro da pasta “src”, temos uma pasta contendo o nome do módulo e dentro desta, uma série de outras pastas. Começaremos pela Controller:

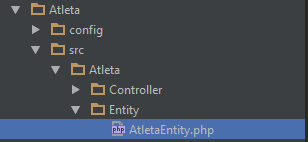
**/Atleta/src/Atleta/Controller**

* Dentro da controller estarão todas as “actions” do sistema. Então tudo que for ser uma ação do sistema, como: gravar, excluir, paginar e toda a programação dessas ações, estarão dentro da controller. Então a “view”, usará todas essas informações para criar uma pagina, mas atenção: aqui vão as ACTIONS, que são parte de back-end. Nelas não tem nenhuma informação de layout. As as informações de layout como coluna, tabelas e botões, ficam na view. Mais abaixo, mostrarei como criar um novo modulo e dai você irão entender a parte de nomes. Abaixo, um exemplo de action:

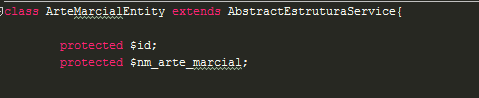
public function cadastroviaacademiaAction()  
{  
 $id\_academia = $this->params('id');  
 $post = $this->getPost();  
  
 if (!empty($post)) {  
 $this->form->setData($post);  
 }  
  
 $dadosView = [  
 'service' => $this->service,  
 'form' => $this->form,  
 'controller' => $this->params('controller'),  
 'atributos' => array(),  
 'id\_academia' =>$id\_academia  
 ];  
  
 return new ViewModel($dadosView);  
}

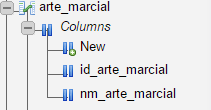
* Acima vemos uma action de cadastro via academia. Todas as Actions vem com o nome seguido de Action. Isso é somente uma boa pratica de programação. Então fica IndexAction, gravarAction. Não vou entrar em detalhes das actions, porque o foco inicial é entender somente a estrutura. Para entender cada action, indico que vocês abram os arquivos e tentem entender o que cada linha faz.

**/Atleta/src/Atleta/Entity**



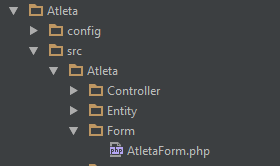
* A pasta Entity é a mais simples de todas. Ela vai conter todos os campos da tabela que você selecionou para usar naquela controller. Vou usar o exemplo do modulo ArteMarcial aqui, pois a tabela da Atleta foi atualizada no Banco, mas não tá atualizada no arquivo





* O id da Entity se refere somente a uma junção que foi feita pra referencia. Mas é possível ver que os mesmos campos de uma, possui na outra.

$select = $sql->select('arte\_marcial')  
 ->where([  
 'arte\_marcial.id\_arte\_marcial = ?' => *$id*,  
 ]);



* Esse foi o select feito para referenciar a id\_arte\_marcial para ID;

**/Atleta/src/Atleta/Form**

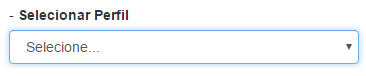
* Na form, vai conter tudo que for referente a formulário. Essa é a definição mais simples e direta que se pode obter. Esse é outro recurso, que ainda não foi bem explorado. Porém vou dar um exemplo de formulário, sem especificar muito, só pra terem uma noção.

$objForm->combo("id\_perfil", '\Perfil\Service\PerfilService','id','nm\_perfil')->required(false)->label("Selecionar Perfil");

* Essa linha de código é referente a uma combobox. Com os devidos dados a serem recuperados e a label, é o nome que ela vai receber.

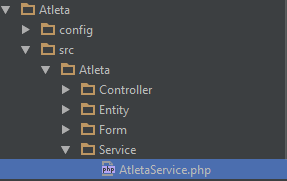
<**div** class=**"col-md-4"**>**-** <?=$this->formRow($form->get('id\_perfil'))?>  
</**div**>

* Já essa linha é a chamada desse form, no Index.



E esse é o resultado na página.

**/Atleta/src/Atleta/Service**

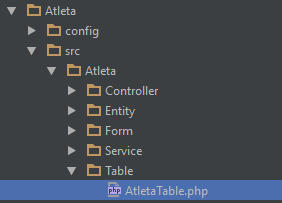


* Na service, terá toda a parte de programação, referente a banco. Então qualquer informação que você deseja recuperar do banco, tudo que for referente a banco, fica na service.

public function getIdAtletaPorNomeToArray(*$nm\_atleta*) {  
  
 $arNomeAtleta = explode('(', *$nm\_atleta*);  
 *$nm\_atleta* = $arNomeAtleta[0];  
  
 $sql = new \Zend\Db\Sql\Sql($this->getAdapter());  
 $filter = new \Zend\Filter\StringTrim();  
 $select = $sql->select('atleta')  
 ->columns(array('id\_atleta'))  
 ->where([  
 'atleta.nm\_atleta = ?' => $filter->filter(*$nm\_atleta*),  
 ]);  
  
 return $sql->prepareStatementForSqlObject($select)->execute()->current();  
}

* Um exemplo simples. Novamente, não vou entrar em detalhe.

**/Atleta/src/Atleta/Table**



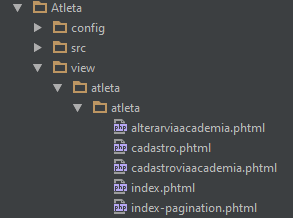
* Como a entity , essa pasta é muito simples.

class ArteMarcialTable extends AbstractEstruturaTable{  
  
 public $table = 'arte\_marcial';  
 public $campos = [  
 'id\_arte\_marcial'=>'id',  
 'nm\_arte\_marcial'=>'nm\_arte\_marcial',  
 ];  
  
}

* Ela diz qual vai ser a tabela usada e os campos que possuem nela, notem que ela faz referencia a entity ( usando o ArteMarcial novamente);

**Atleta/view/atletla/atleta**

* Agora vamos para a view.



-

A view é simples, tudo que for de visual, de front-end, vem aqui. Toda a parte

<?php  
/\*\* @var $form \Atleta\Form\Atleta \*/  
/\*\* @var $lista \Atleta\Service\Atleta[] \*/  
?>  
<**div** class=**"page-head"**>  
 <**h2**>**Lista de Atletas**</**h2**>  
</**div**>  
<**div** class=**"cl-mcont"**>  
 <**div** class=**"row"**>  
 <**div** class=**"col-md-12"**>  
 <**a** class=**"btn btn-default"** href=**"**<?= $this->url('navegacao', array('controller' => $controller, 'action' => 'cadastro')) ?>**"** title=**"Adiciona um novo atleta"**>**Novo Atleta**</**a**>  
 </**div**>  
 </**div**>  
 <**br** />  
 <**div** class=**"row"** id=**"atleta-pagination"**></**div**>   
</**div**>  
  
<**script**>  
 $(function () {  
  
 $.ajax({  
 type: "post",  
 dataType: "text",  
 cache: false,  
 url: '/atleta-atleta/index-pagination',  
 async: true,  
 data: {  
 },  
 beforeSend: function () {  
  
 $('#atleta-pagination').html(  
 '<div class="row"><div class="col-md-12 text-center"><p><img src="/assets/img/carregando.gif"><p></div></div>'  
 );  
 },  
 success: function (*data*) {  
  
 $('#atleta-pagination').html(*data*);  
 }  
 });  
  
 })  
</**script**>

* Essa é a pagina index, ela te simples assim, porque possui uma paginação entro. Quando começar a olhar cada arquivo, vão entender melhor. Mas a explicação básica é essa, a view, é toda a parte visual.
* Pra finalizar vamos dar uma explicação rápida. A view usa a controller pra montar a sua página com as actions e a form quando se tratar de formulários, a controller usa a service para a maioria dos actions, a service para montar seus métodos usa a entity, e a entity usa a table.
* Então no final fica. View usa controller e Form- Controller usa Service – Service usa entity e a entity usa a table.

**Criando um novo modulo.**

* Aqui vou mostrar como criar a estrutura de um novo modulo. Pastas, nome dos arquivos. Mas o conteúdo dos arquivos não será mostrado, até por que isso varia de modulo pra modulo, para saber mais sobre o conteúdo, consulte os módulos já criados.
* Aqui usarei o PHPStorm. Mas isso pode ser feito dentro da própria pasta, não faz diferença, mostrarei tanto com o PhPstorm, quando pela pasta, só que pela pasta não haverá imagens, até porque são passos simples , como criar pasta, criar arquivos. Será dividido em três partes:

Criação do novo modulo .

Adicionar o modulo a estrutura geral.

Autorizar o modulo no Banco.

**Criação do Novo modulo**

* Para criar um novo modulo, é bem simples. É só criar uma nova pasta, como o nome do modulo em Maiúsculo, caso seja um modulo composto o segundo nome também é composto.
* Exemplo: Modulo, Criando, NovoModulo, CriandoModulo.
* Vamos trabalhar com um nome composto, pelo fato de ser mais complicada. Na verdade é só mais trabalhoso, para ficar mais atento.
* O nome do nosso modulo, será: NovoModulo.

**Passo 1**

Criar a pasta com o nome do modulo desejado, no nosso caso “NovoModulo”. Essa pasta deverá ser criada dentro da pasta module.  
No meu caso /xampp/htdocs/ejur/module.



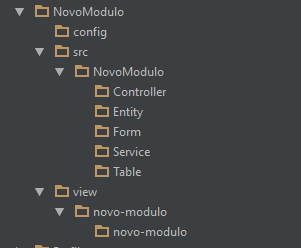
**Passo 2**

Criar todas as sub-pastas com os nomes que foi mostrado no explicativo lá em cima, sobre a estrutura. As pastas são – config – src – view

Dentro da config, não haverá mais subpastas.  
Dentro da src, haverá mais uma sub-pasta chamada “NovoModulo”,ou o nome que você escolheu. E dentro dessa pasta, haverá mais 5 sub-pastas.

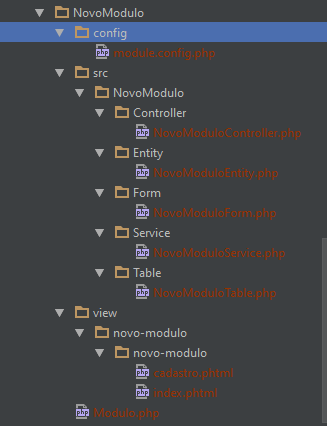
* Controller
* Entity
* Form
* Service
* Table

Na view – Uma sub-pasta chamada “novo-modulo” e dentro da novo modulo, mais uma subpasta, com o mesmo nome. Note que aqui, ela está separada por um traço e em minúsculo.



**Passo 3**

A partir daqui, vamos criar os arquivos. Aqui é simples. O arquivo vai vim com o nome do modulo junto com o nome do submodulo com a extensão .php . Ou seja. NovoModulo Controller, NovoModuloEntity. Isso pra pasta SRC.  
Para pasta config, tem um nome fixo: module.config.php.  
na pasta view, todos os arquivos vão ter a extensão de PHTML.  
Na pasta geral terá um arquivo chamado Module.php



A partir daqui o modulo já está criado. Agora precisamos adicionar ela na estrutura geral.

**Adicionar a Estrutura.**

Abra o arquivo: aplication.config.php que está localizado dentro de /xampp/htdocs/ejur/config.

Nele tem uma pequena estrutura d Array:

return array(  
 'modules' => array(  
 'Application',  
 'Auth',  
 'Estrutura', //Tem que vir antes dos demais módulos  
 'Banco',  
 'Principal',  
 'Email',  
 'Usuario',  
 'Perfil',  
 'EsqueciSenha',  
 'Config',  
 'Cidade',  
 'Estado',  
 'Sexo',  
 'EstadoCivil',  
 'TipoUsuario',  
 'SituacaoUsuario',  
 'Endereco',  
 'ArteMarcial',  
 'Estilo',  
 'Graduacao',  
 'TipoEvento',  
 'Evento',  
 'Academia',  
 'Atleta',  
 'CategoriaPeso',  
 'CategoriaIdade',  
 'RegrasLutas',  
 'DetalhesRegrasLuta',  
 'InscricoesEvento',  
 'ChavesLuta',  
 'CompactAsset', //Compacta o Javascript e CSS para retornar em apenas uma requisição (Responsável pela minificar o css e js: compila os arquivos em um só)  
 #'ContaBancaria',  
 #'DOMPDFModule',  
 //Ronaldo 02/03/2016 - Responsável por melhorar o desempenho da aplicação  
 'EdpSuperluminal', //http://dev.competicaotkd.com.br/?EDPSUPERLUMINAL\_CACHE - Execute isso na URL para compilar os arquivos e ficar mais rapido - em cada requisição, em vês de baixar em tempo de execução cada require do autoload, ele salva um unico arquivo, minificado, com todas as classes dentro  
 #'Gerador',  
 #'Login',  
 #'PhpBoletoZf2',  
 #'Situacao',  
 #'Telefone',  
 #'TipoConta',  
 #'TipoTelefone',  
 'Controller',  
 'GerenciadorPermissao'  
 ),

Vamos adicionar no fim, após a última.

'Controller',  
'GerenciadorPermissao',  
'NovoModulo'

- Dessa forma.

- Pronto, já foi adicionado a estrutura.

- Agora iremos somente autorizar no banco de dados.

**Adicionar no banco de dados**.

-Para adicionar no banco de dados é simples, é só adicionar um novo registro. Usando o seguinte registro.

Use “ banco” – Isso é pra usar a banco de dados do seu projeto. No meu caso: ejur.

- ***Insert into controller ( id\_controller, nm\_controller) values(null, “novo\_modulo-novomodulo”);***

Pronto, dessa forma, ele já está adicionado ao banco.

Esse comando adicionará dentro da tabela controller, a linha com o nm\_controller (novo\_modulo-novomodulo), com o ID de auto incremento.

- Caso seja um modulo composto, ele tem que seguir essa estrutura ai.

- Caso fosse um modulo comum seria da seguinte forma. ***Insert into controller ( id\_controller, nm\_controller) values(null, “modulo-modulo”);***

- Pronto, seu modulo já está criado, adicionado , e autorizado. Agora ele só precisa ser editado de acordo com a sua utilidade.

**Dicas**

* Umas dicas interessantes.
* Sempre comente seu código, isso te ajuda a se situar para caso erre algo, saiba ir direto ao ponto.
* Caso tenha dúvida, em algum funcionamento. Comente toda a linha assim, você saberá o que essa linha altera a no na estrutura e consequentemente no site.

<!-- <div class="row"> -->

- Em caso de duvidas, pergunte.