Tutorial do Fgsl Framework por Flávio Gomes da Silva Lisboa

Versão do FGSL2ZF: 1.0.0

Sumário

Este tutorial tem por objetivo ensinar programadores a usar a biblioteca Fgsl para estender funcionalidades do Zend Framework.

O projeto FGSL2ZF pretende mostrar como estender componentes do Zend Framework. Não há propósito comercial, apenas o de ajudar iniciantes da comunidade ZF.

FGSL2ZF também pretende ser um laboratório para gerar candidatos a componentes do Zend Framework. Uma vez que os componentes da biblioteca Fgsl library alcancem maturidade (depois de refatoração, construção de testes, etc), eles serão propostos.

Neste tutorial, eu mostrarei como criar uma aplicação CRUD rapidamente com Zend Framework e a biblioteca Fgsl.

O componente CRUD de FGSL2ZF pretende tornar mais fácil a criação de controladores que manipulam operações de inclusão, consulta, atualização e remoção.

1 Banco de dados

Nós usamos um banco de dados chamado **fgsl_books**.
Ele tem duas tabelas, **books** e **categories**.
Há um relacionamento muitos-para-um entre a tabela books e a tabela categories.
A figura 1 é o DER de nosso banco de dados.

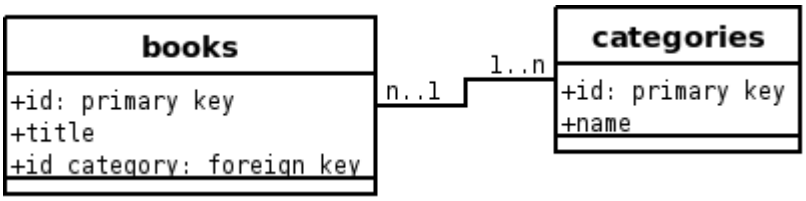

Figure 1: DER fgsl_books

Você pode usar qualquer banco de dados relacional (de preferência um que tenha driver em PHP). Para fazer isso tutorial, eu usei um banco de dados PostgreSQL. Se você quiser seguir este exemplo, os scripts de criação das tabelas são estes:

```
-- Table: categories

-- DROP TABLE categories;

CREATE TABLE categories
(
  id serial NOT NULL,
  "name" character varying(80) NOT NULL,
  CONSTRAINT category_pkey PRIMARY KEY (id)
)
WITH (OIDS=FALSE);
ALTER TABLE categories OWNER TO postgres;
```

```
-- Table: books

-- DROP TABLE books;

CREATE TABLE books
(
  id serial NOT NULL,
  title character varying(80) NOT NULL,
  id_category integer NOT NULL,
  CONSTRAINT books_pkey PRIMARY KEY (id),
  CONSTRAINT books_id_categories_fkey FOREIGN KEY (id_category)
      REFERENCES categories (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (OIDS=FALSE);
ALTER TABLE books OWNER TO postgres;
```

2 Projeto PHP

O projeto de nosso exemplo foi construído com Zend Framework 1.11.11. Eu usei Zend_Tool para criar a estrutura da aplicação.

O nome do projeto PHP é fgslbooks.

Os primeiros passos para configurar a aplicação são:

2.1 Criar arquivos .htaccess

Na raiz do projeto, crie um arquivo .htaccess com o seguinte conteúdo:

```
SetEnv APPLICATION_ENV development

RewriteEngine On
RewriteRule ^.*$ public/index.php
```

Dentro da pasta application, crie outro arquivo .htaccess com este conteúdo:

```
deny from all
```

2.2 Aponte para as bibliotecas

Coloque as pastas Zend do Zend Framework e Fgsl do FGSL2ZF dentro da pasta library de sua aplicação, ou faça um link simbólico para elas.

2.3 Configure baseUrl

Dentro do arquivo application.ini (em application/configs), adicione esta linha:

```
resources.frontController.baseUrl = /fgslbooks
```

2.4 Adicione o namespace Fgsl ao autoloader

Adicione este método na classe Bootstrap:

```
	public function _initNamespace()
	{
		Zend_Loader_Autoloader::getInstance()->registerNamespace('Fgsl');
	}
```

2.5 Configure a conexão com o banco de dados

Dentro do arquivo application.ini (em application/configs), na seção production, inclua estas linhas:

```
resources.db.adapter = PDO_MYSQL
resources.db.params.username = [POSTGRES USER]
resources.db.params.password = [POSTGRES PASSWORD]
resources.db.params.dbname = fgsl_books
resources.db.params.host = localhost
```

2.6 Crie seus mapeadores

Category.php

```
<?php

class Application_Model_DbTable_Category extends Fgsl_Db_Table_Abstract
{

    protected $_name = 'categories';

    public function __construct()
    {
      parent::__construct();
      $this->_fieldKey = 'id';
      $this->_fieldNames = array('id','name');
      $this->_fieldLabels = array(
                  'id' => 'Id',
                  'name' => 'Name');
      $this->_orderField = 'name';
      $this->_searchField = 'name';
      $this->_typeValue = array(
                  'id' => Fgsl_Db_Table_Abstract::INT_TYPE,
      );
      $this->_lockedFields = array('id');
      $this->_joinField = 'name';
            $this->_addDependents('Book');
    }


}
```

Book.php

```php
<?php

class Application_Model_DbTable_Book extends Fgsl_Db_Table_Abstract
{

    protected $_name = 'books';

    public function __construct()
    {
      parent::__construct();
      $this->_fieldKey = 'id';
      $this->_fieldNames = array('id','title','id_category');
      $this->_fieldLabels = array(
                  'id' => 'Id',
                  'title' => 'Title',
                  'id_category' =>
                  'Category');
      $this->_orderField = 'title';
      $this->_searchField = 'title';
      $this->_selectOptions = array();
      $this->_typeElement = array('id_category' =>
                  Fgsl_Form_Constants::SELECT);
      $this->_typeValue = array(
                  'id'
                  => Fgsl_Db_Table_Abstract::INT_TYPE,
                  'id_category'
                  => Fgsl_Db_Table_Abstract::INT_TYPE
      );
      $this->_lockedFields = array('id');
      $this->_addRelation('Category', 'id_category', 'Category', 'id');
    }


}
```

2.7 Crie seus modelos

Category.php

```php
<?php

class Application_Model_Category extends Fgsl_Model_Abstract
{


}
```

Book.php

```php
<?php

class Application_Model_Book extends Fgsl_Model_Abstract
{

}
```

2.8 Crie os controladores CRUD

Crie um controlador CRUD para as categorias:

CategoryCrudController.php

```
<?php

class CategoryCrudController extends Fgsl_Controller_Action_Crud_Abstract
{

	public function init()
	{
		parent::init();
		$this->_useModules = false;
		$this->_model = new Application_Model_Category();
		$this->_title = 'Category Listing';
		$this->_searchOptions = array($this->_model->getDbTable()-
>getSearchField()=>'name');
		$this->_searchButtonLabel = 'Search';
		$this->_config();
	}


}
```

Crie um controlador CRUD para os livros:

BookCrudController.php

```
<?php

class BookCrudController extends Fgsl_Controller_Action_Crud_Abstract
{

	public function init()
	{
		parent::init();
		$this->_useModules = false;
		$this->_model = new Application_Model_Book();
		$this->_title = 'Book Listing';
		$this->_searchOptions = array($this->_model->getDbTable()-
>getSearchField()=>'Title');
		$this->_searchButtonLabel = 'Search';
		$this->_config();
	}
}
```

2.9 Crie view scripts compartilhados

Fgsl_Crud_Controller permite que você mantenha somente dois view scripts para todos os seus controladores CRUD. Assim, você precisa criar somente dois arquivos na pasta **application\views\scripts**.

index.phtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title><?=$this->title?></title>
</head>
<body>
	<h1>
		<?=$this->title?>
	</h1>
	<table>
		<tr>
			<td>
				<h2>
					<a href="<?=$this->getInsertLink()?>">Insert</a>
				</h2>
			</td>
			<td>
			<?=$this->getSearchForm()?>
			</td>
		</tr>
	</table>
<table border="1">
<?=$this->renderHTMLTableContent($this->records)?>
</table>
</body>
</html>
```

pre-edit.phtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<?=$this->form?>
</body>
</html>
```

2.10 Testando

Você pode ver o funcionamento da aplicação chamando a URL de cada controlador:

Primeiro, manipule as categorias. Mas deixe algumas categorias, porque o outro controlador precisará delas.

http://localhost/fgslbooks/category-crud

Depois, manipule os livros.

http://localhost/fgslbooks/book-crud

Você percebeu que é fácil adicionar outro controlador para trabalhar com outro modelo sem muitas linhas de código?

2.11 E isso é tudo, pessoal!

Contribuições são bem-vindas.

Obrigado!

flaviogomesdasilvalisboa@yahoo.com.br

www.fgsl.eti.br