Fgsl Framework Tutorial by Flávio Gomes da Silva Lisboa

Version of FGSL2ZF: 1.0.0

Table of Contents

This tutorial aims to teach programmers to use Fgsl library to extend features of Zend Framework.

Project FGSL2ZF intend to show how to extend components of Zend Framework. There is no commercial purpose, only to help newbies of ZF Community.

FGSL2ZF also intend to be a laboratory to generate candidates to components of Zend Framework. Since components of Fgsl library reaching maturity (after refactoring, building of tests, etc), they will be proposed to.

In this tutorial, I will show how to build a CRUD application quickly with Zend Framework and Fgsl library.

CRUD component of FGSL2ZF intend to become easy creating controllers that handle operations of insert, select, update and delete.

1 Database

We use a database called **fgsl_books**.
It has two tables, **books** and **categories**.
There is a many-to-one relationship between table books and categories.
The figure 1 is the DER of our database.

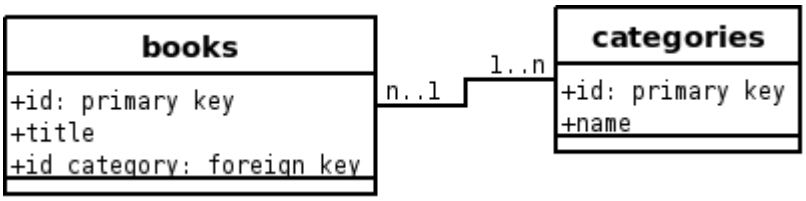

Figure 1: DER fgsl_books

You can use any relational database (preference for that have driver in PHP). To do this tutorial, I used a PostgreSQL database. If you want to follow this example, the table scripts are that:

```
-- 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 PHP Project

The project of our example was built with Zend Framework 1.11.11. I've used Zend_Tool to create the application structure.

The name of PHP Project is fgslbooks.

The first steps to configure application are:

2.1 Create .htaccess files

In root of project, create a .htaccess file with the following content:

```
SetEnv APPLICATION_ENV development

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

Into folder application, create another .htaccess file with that content:

```
deny from all
```

2.2 Point to libraries

Place the folders Zend of Zend Framework and Fgsl of FGSL2ZF into folder library of your application, or make a symbolic link to them.

2.3 Configure baseUrl

Into file application.ini (in application/configs), add this line:

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

2.4 Add Fgsl namespace to autoloader

Add this method into class Bootstrap :

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

2.5 Configure database connection

First, into file application.ini (in application/configs), include this section:

```
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 Create your mappers

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 Create your models

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 Create CRUD controllers

Create a CRUD controller for category:

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();
	}


}
```

Create a CRUD controller for book:

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 Create shared view scripts

Fgsl_Crud_Controller allows that you mantain only two view scripts for each one of your CRUD controllers. So, we only need to create two files, into folder **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 Testing

You can see the working of application calling the URLs of each controller:

First, handle categories. But at end, let some categories, because the other controller will need of them.

http://localhost/fgslbooks/category-crud

After, handle books.

http://localhost/fgslbooks/book-crud

Do you perceive that is easy to add another controller to work with another model, without a lot of lines of code?

2.11 And that it's all, folks!

Contributions are welcome.

Thank you!

flaviogomesdasilvalisboa@yahoo.com.br

www.fgsl.eti.br