INTRODUCCIÓN A ZEND FRAMEWORK

Versión original: Getting Started with Zend Framework 1.11

CONTENIDOS

Este tutorial está destinado a brindar una introducción al uso de Zend Framework mediante la creación de una aplicación simple con base de datos, utilizando el paradigma de Modelo-Vista-Controlador.

Nota: Este tutorial ha sido testeado **entre las versiones 1.10.1 y 1.11.4** de Zend Framework. Existe una muy buena oportuindad de que funcione con versiones más nuevas dentro de la serie 1.x, pero no funcionará con versiones anteriores a 1.10.1.

REQUERIMIENTOS

Zend Framework tiene los siguientes requerimientos:

- PHP 5.2.4 (o mayor)
- Un servidor web que tenga habilitada la extensión mod_rewrite o similar.

SUPOSICIONES PARA EL TUTORIAL

He supuesto que estás corriendo PHP 5.2.4 o superior en un servidor web Apache. La instalación de Apache ***debe tener la extensión mod_rewrite instalada y configurada***.

También debes asegurarte de que Apache está configurado para soportar archivos .htaccess. Esto normalmente se hace cambiando la configuración:

```
AllowOverride None
```

a

```
AllowOverride All
```

en el archivo httpd.conf. Mirar la documentación de la distribución para detalles más exactos. No vas a poder navegar a ninguna página aparte de la página de inicio en este tutorial si no tienes configurado correctamente mod_rewrite y el archivo .htaccess.

CONSIGUIENDO EL FRAMEWORK

Se puede bajar una copia de Zend Framework en http://framework.zend.com/download/latest, tanto en formato .zip o .tar.gz. Al final de la página hay links de descarga. La versión "Minimal" es la que necesitas.

CONFIGURANDO ZEND_TOOL

Zend Framework posee una herramienta para línea de comando. Comenzamos configurándola.

ZEND_TOOL EN WINDOWS

- En Archivos de Programa crea una carpeta llamada ZendFrameworkCli.
- Hacemos doble click en el archivo descargado, ZendFramework-1.10.6-minimal.zip.
- Copiamos las carpetas bin y library desde la carpeta ZendFramework-1.10.6-minimal.zip hacia la carpeta C:\Archivos de Programa\ZendFrameworkCli. Esta carpeta ahora debería tener dos carpetas internas: bin ylibrary.
- Agregá la carpeta bin a tu ruta de acceso:
  - En el Panel de Control, ir a la sección de Sistema.
  - Elegir Avanzado y luego presionar el botón Variables de Entorno.
  - En el listado de "Variables de Sistema", encontrar la variable Ruta (o Path) y hacer doble click sobre ella.
  - Agregar ;C:\Archivos de Programa\ZendFrameworkCli\bin al final de la caja de texto y presionar Ok. (el punto y coma que está primero es muy importante!)
  - Reiniciar la computadora.

ZEND_TOOL EN MAC OS X (ES SIMILAR EN LINUX)

- Extraer el contenido de ZendFramework-1.10.6-minimal.zip en la carpeta Downloads haciéndole doble click.
- Copiar el contenido a la carpeta /usr/local/ZendFrameworkCli desde la Terminal, escribiendo:

```
sudo cp -r ~/Downloads/ZendFramework-1.10.6-minimal /usr/local/ZendFrameworkCli
```

- Editar el archivo bash profile para agregar un alias:
  - Desde la Terminal, escribir: open ~/.bash_profile
  - Agregar alias zf=/usr/local/ZendFrameworkCli/bin/zf.sh al final del archivo.
  - Guardar y cerrar el TextEdit.
  - Salir de Terminal.

PROBANDO ZEND_TOOL

Podés probar la instalación de interfaz de línea de comando Zend_Tool abriendo una Terminal o Símbolo de Sistema y tipeando:

```
zf show version
```

Si todo funcionó correctamente, deberías leer:

```
Zend Framework Version: 1.10.0
```

Sino, deberías verificar que guardaste la ruta correctamente y que la carpeta bin existe en la carpeta ZendFrameworkCli. Una vez que la herramienta **zf** está funcionando, podés ver todas las opciones disponibles con zf --help.

Nota: Si la distribución de PHP que tienes instalada traía una copia de Zend Framework, por favor verifica que no sea ZF 1.9 ya que no funcionará correctamente con este tutorial. Al momento de escribir esto, la distribución de XAMPP hacía esto.

LA APLICACIÓN TUTORIAL

Ahora que todas las partes están donde debe ser para que podamos construir una aplicación de Zend Framework, veamos un poco sobre la aplicación que vamos a crear. Vamos a crear un simple sistema de inventario para mostrar nuestra colección de CDs. La página principal va a listar nuestra colección y nos va a permitir agregar, modificar y elimiar CDs. Como con cualquier ingeniería de software, ayuda que hagamos un poco de planificación previa. Vamos a necesitar cuatro páginas en nuestro sitio web:

Página de inicio	Se va a mostrar el listado de discos y se van a proveer links para poder editarlos y eliminarlos. Además va a existir un link para agregar nuevos discos.
Agregar nuevo disco	Va a existir un formulario para agregar un nuevo disco.
Editar un disco	Va a existir un formulario para editar un disco.
Eliminar un disco	Va a existir una confirmación para eliminar un disco y luego se eliminará el disco.

También vamos a necesitar guardar la información en una base de datos. Vamos a necesitar una sola tabla con estos campos:

Campo	Tipo de dato	Null?	Comentarios
id	integer	No	Primary key, auto increment
artist	varchar(100)	No	
title	varchar(100)	No	

GENERANDO NUESTRA APLICACIÓN DE LA NADA

Comencemos a armar nuestra aplicación. Donde sea posible, vamos a hacer uso del comando zf para ahorrar tiempo y esfuerzo. La primer tarea es crear el esqueleto del proyecto (carpetas y archivos).

Abrimos Terminal o el Símbolo de Sistema y nos dirigimos hacia la carpeta raíz de nuestro servidor web, utilizando el comando cd. Nos tenemos que asegurar de tener los permisos necesarios para poder crear archivos en esa carpeta, y que el servidor web tiene permisos de lectura. Entonces escribimos:

```
zf create project zf-tutorial
```

La herramienta ZF va a crear una carpeta llamada zf-tutorial y la va a llenar con la estructura de carpetas recomendada. Esta estructura asume que tenés completo control por sobre la configuración de Apache, por lo que vas a poder mantener la mayor parte de los archivos fuera del directorio raíz del servidor web. Ahora deberías ver la siguiente estructura de archivos y carpetas:

(También existe un archivo oculto .htaccess en la carpeta public/).

La carpeta application/ es donde vive el código fuente de este sitio web. Como puedes ver, tenemos carpetas separadas para los modelos, las vistas y los controladores de nuestra aplicación. La carpeta public/ es la parte pública de nuestro sitio, lo cual significa que la URL para poder ver la aplicación va a ser http://localhost/zf-tutorial/public/. Está armado de esta forma para que la mayoría de los archivos de esta aplicación no puedan ser accedidos directamente y el sistema sea más seguro.

Nota: un sitio web que se encuentra en producción, debería tener configurado un virtual host para tener seleccionada la entrada al sitio directamente a la carpeta public/. Por ejemplo, puede crear un virtual host llamado zf-tutorial.localhost que sea parecido a esto:

```
<VirtualHost *:80>

    ServerName zf-tutorial.localhost

    DocumentRoot /var/www/html/zf-tutorial/public

    <Directory "/var/www/html/zf-tutorial/public">

        AllowOverride All

    </Directory>

</VirtualHost>
```

Se podrá acceder al sitio a través de http://zf-tutorial.localhost/ (asegúrate de actualizar el archivo /etc/hosts o C:\Windows\system32\drivers\etc\hosts para que zf-tutorial.localhost apunte a 127.0.0.1). No vamos a configurar esto en el tutorial, ya que es más fácil utilizar un subdirectorio para probar el código.

Las imágenes, código javascript y estilos CSS están almacenados por separado dentro de la carpeta public/. Los archivos descargados de Zend Framework deben ir en la carpeta library/. Si necesitamos alguna otra librería, también tiene que ir en esta carpeta.

Copia la carpeta library/Zend/ del archivo descargado (ZendFramework-1.10.6-minimal.zip) en la carpeta zf-tutorial/library/, de modo tal que la carpeta zf-tutorial/library/ contenga la carpeta Zend/.

Ahora podés probar que todo está funcionando navegando a http://localhost/zf-tutorial/public. Deberías ver algo parecido a esto:

PRINCIPIOS DE BOOTSTRAPPING

El controlador de Zend Framework usa el patrón de diseño del Front Controller y dirige todas las solicitudes a un único archivo, index.php. Esto asegura que el entorno de trabajo esté configurado correctamente para hacer funcionar la aplicación (proceso conocido como bootstrapping). Podemos conseguir esto gracias al archivo .htaccess en la carpeta zf-tutorial/public/ previamente generada por Zend_Tool, el cual redirige todas las solicitudes a public/index.php, el cual también fue generado por Zend_Tool.

El archivo index.php es el punto de entrada a nuestra aplicación y es utilizado para crear una instancia de Zend_Application para inicializar nuestra aplicación y luego hacerla funcionar. Este archivo también define dos constantes: APPLICATION_PATH y APPLICATION_ENV, los cuales definen la ruta de la carpeta application/ y el entorno o modo de trabajo de la aplicación. Por defecto está configurado en production en index.php, pero deberías definirlo como development en el archivo .htaccess agregando la línea:

```
SetEnv APPLICATION_ENV development
```

El componente Zend_Application es utilizado para iniciar la aplicación y está configurado para usar las directivas guardadas en el archivo application/configs/application.ini. Este archivo también es auto-generado para nosotros.

Tenemos disponible la clase Bootstrap que extiende de Zend_Application_Bootstrap_Bootstrap se encuentra en application/Bootstrap.php, en la cual podemos agregarle código cualquier tarea requerida al inicio de la aplicación.

El archivo application.ini, el cual se encuentra alojado en la carpeta application/configs es cargado desde el componente Zend_Config_Ini. Zend_Config_Ini entiende el concepto de herencia de secciones, las cuales están delimitadas utilizando el símbolo de dos puntos. Por ejemplo:

```
[staging : production]
```

Esto significa que la sección staging hereda todas las configuraciones de production. La constante APPLICATION_ENV define que sección está cargada. Obviamente, durante el desarrollo, es mejor utilizar la secciónd**evelopment** y cuando se encuentre el sitio en el servidor final, se debería utilizar la sección production. Vamos a poner todos los cambios que realicemos en applications.ini dentro de la sección de production para que todas las configuraciones tomen los cambios que hagamos.

EDITANDO EL ARCHIVO APPLICATION.INI

El primer cambio que debemos realizar es agregar nuestra información de zona horaria (timezone) para las funciones de fecha y hora de PHP. Edita el archivo application/configs/application.ini y agrega:

```
phpSettings.date.timezone = "America/Argentina/Buenos_Aires"
```

luego de los otros valores de phpSettings en la sección de [production]. Obviamente, vos deberías utilizar tu propia zona horaria. Estamos en una posición de agregar el código específico de nuestra aplicación.

CÓDIGO ESPECÍFICO DE LA APLICACIÓN

Antes de que definamos nuestros archivos, es importante entender como Zend Framework espera que la página esté organizada. Cada página de la aplicación es conocida como **action** y las acciones están agrupadas dentro de **controllers**. Para una URL del formato http://localhost/zf-tutorial/public/news/view, el controlador es News y la acción es view. Esto es para permitir el agrupamiento de acciones relacionadas. Por ejemplo, el controlador News puede tener acciones como ser list, archived y view. El sistema MVC de Zend Framework también soporta módulos para agrupar controladores, pero esta aplicación no es tan grande como para preocuparnos por utilizarlos.

Por defecto, cada controlador de Zend Framework tiene una acción especial reservada llamada index, que sirve como la acción por defecto propia. Esto es para casos como http://localhost/zf-tutorial/public/news/ donde la acción index de News es ejecutada. Además hay un nombre de controlador

por defecto, que también es llamado index por lo que la URL http://localhost/zf-tutorial/public/ va a causar que la acción index del controladorIndex sea ejecutada.

Como este es un tutorial básico, no nos vamos a complicar con cosas "complicadas" como regisro de usuarios. Eso puede esperar por otro tutorial (o pueden leer sobre esto en *Zend Framework in Action!*).

Como tenemos cuatro páginas que aplican a los discos, vamos a agruparlos en un único controlador como cuatro acciones. Vamos a utilizar el controlador por defecto y las cuatro acciones serán:

Página	Controlador	Acción
Página de inicio	Index	index
Agregar nuevo disco	Index	add
Editar un disco	Index	edit
Eliminar un disco	Index	delete

A medida que un sitio se vuelve más complicado, controladores adicionales son necesarios y hasta se pueden agrupar controladores dentro de módulos si es necesario.

CREANDO EL CONTROLADOR

Ahora estamos listos para escribir nuestro controlador. En Zend Framework, el controlador es una clase que tiene que ser llamada {Controller name}Controller. Tengan en cuenta que {Controller name} debe comenzar con la primer letra en mayúscula. Esta clase debe estar dentro de un archvo llamado {Controller name}Controller.php dentro de la carpeta application/controllers. Cada acción es una función pública dentro de la clase del controlador que debe llamarse {action name}Action. En este caso {action name} tiene que ser escrito en letra en minúscula. Los nombres con mayúsculas y minúsculas son permitidas en los controladores y las acciones, pero tiene algunas reglas especiales que tienes que comprender antes de que las utilices. ¡Primero debes verificar la documentación!

La clase de nuestro controlador se llama IndexController la cual está definida en el archivo application/controllers/IndexController.php, el cual fue generado automaticamente por nosotros con Zend_Tool. Además ya contiene el primer método/acción, indexAction(). Ahora necesitamos agregar las acciones adicionales.

Agregar más acciones del controlador es posible usando el comando create action de la herramienta **zf**. Abrimos el Terminal o el Símbolo de Sistema y nos posicionamos en el directorio del proyecto (zf-tutorial/). Luego escribí estos tres comandos:

```
zf create action add Index

zf create action edit Index

zf create action delete Index
```

Estos comandos crean tres nuevos métodos: addAction, editAction y deleteAction en el controlador IndexController, junto con el código de las vistas que vamos a necesitar más adelante. Ahora tenemos las cuatro acciones que queremos utilizar.

Las URLs para cada acción son:

URL	Método/Acción
http://localhost/zf-tutorial/public/	IndexController::indexAction()
http://localhost/zf-tutorial/public/index/add/	IndexController::addAction()
http://localhost/zf-tutorial/public/index/edit/	IndexController::editAction()
http://localhost/zf-tutorial/public/index/delete/	IndexController::deleteAction()

Ahora puedes probar las tres nuevas acciones, y deberias ver un mensaje como el siguiente:

View script for controller **index** and script/action name **add**

Nota: Si llegás a tener un error 404, significa que no configuraste el servidor Apache con el módulo mod_rewrite o no configuraste el comando AllowOverride correctamente en los archivos de configuración de Apache, para que el archivo .htaccess en la carpeta public/ pueda funcionar.

LA BASE DE DATOS

Ahora que tenemos armados el esqueleto de nuestra aplicación, los controladores, las acciones y las vistas, es hora de ver el modelo de nuestra aplicación. Recuerda que el modelo es la parte que se ocupa con la parte principal del proyecto (mejor conocido como las "reglas de negocio") y, en nuestro caso, tiene que ver con la base de datos. Vamos a usar la clase Zend_Db_Table de Zend Framework, la cual es utilizada para encontrar, insertar, actualizar y eliminar registros de una tabla en la base de datos.

CONFIGURACIÓN DE LA BASE DE DATOS

Para usar Zend_Db_Table, necesitamos decir que base de datos usar junto con un usuario y una contraseña. Como preferimos no escribir código de más en nuestra aplicación con este tipo de información vamos a utilizar un archivo de configuración para guardar estos datos. El componente de Zend_Application viene por defecto con un recurso para configurar la base de datos, por lo que debemos hacer es guardar la información apropiada en el archivo configs/application.ini y el sistema hará el resto.

Abrimos el archivo application/configs/application.ini y agregamos lo siguiente al final de la sección [production] (por ejemplo, sobre la sección [staging]):

```
resources.db.adapter = PDO_MYSQL

resources.db.params.host = localhost

resources.db.params.username = rob

resources.db.params.password = 123456

resources.db.params.dbname = zf-tutorial
```

Obviamente deberías usar tu nombre de usuario, contraseña y nombre de base de datos, no mis datos! La conexión a la base de datos va a ser realizada automáticamente para nosotros y el adapter por defecto deZend_Db_Table va a estar configurado. Puedes leer más sobre los otros recursos disponibles aquí: http://framework.zend.com/manual/en/zend.application.available-resources.html.

CREANDO UNA TABLA EN LA BASE DE DATOS

Como vimos en el plan inicial, vamos a utilizar una base de datos para guardar la información de nuestros discos. Vamos a estar utilizando MySQL y el comando SQL para crear la tabla es:

```
CREATE TABLE albums (

  id int(11) NOT NULL auto_increment,

  artist varchar(100) NOT NULL,

  title varchar(100) NOT NULL,

  PRIMARY KEY (id)

);
```

Pasamos el comando SQL en un cliente MySQL como ser phpMyAdmin o el cliente MySQL de la línea de comandos.

INSERTAMOS DATOS DE PRUEBA

Vamos a insertar algunos registros en la tabla para que podamos probar que funcione en la página de inicio. Voy a tomar algunos de los primeros CDs "Bestsellers" de Amazon UK. Pasamos el siguiente comando SQL en el cliente MySQL que prefieras:

```
INSERT INTO albums (artist, title)

VALUES

('Paolo Nutine', 'Sunny Side Up'),

('Florence + The Machine', 'Lungs'),

('Massive Attack', 'Heligoland'),

('Andre Rieu', 'Forever Vienna'),

('Sade', 'Soldier of Love');
```

Ahora tenemos información en la base de datos y podemos escribir un modelo muy simple para ello.

EL MODELO

Zend Framework no provee una clase de Zend_Model, ya que el modelo es la lógica de tu negocio y es tu trabajo decidir cómo va a trabajar. Hay muchos componentes que puedes utilizar para esto, dependiendo de cada necesidad. Una opción es tener clases de modelos que representan cada entidad en tu aplicación, y luego utilizar objetos que devuelvan y guarden las entidades en la base de datos. Esta opción está documentada en el tutorial QuickStart de Zend Framework aquí: http://framework.zend.com/manual/en/learning.quickstart.create-model.html.

Para este tutorial vamos a crear un modelo que extiende de Zend_Db_Table y utiliza Zend_Db_Table_Row. Zend Framework provee la clase Zend_Db_Table, la cual implementa el patrón de diseño Table Data Gateway para permitir una interfaz con la información en la tabla de la base de datos. Ten en cuenta que el patrón Table Data Gateway puede ser limitado en sistemas más grandes. También puede suceder que estemos tentados de escribir el código de acceso a la base de datos dentro de alguna acción de un controlador ya que se puede realizar gracias a Zend_Db_Table.

Zend_Db_Table_Abstract es una clase abstracta, de la cual vamos a derivar nuestra clase específica para administrar nuestros discos. No importa como nombremos a nuestra clase, pero tiene sentido que la nombremos con relación a la tabla que vamos a comunicar. Nuestro proyecto tiene por defecto un autoloader instanciado por la clase Zend_Application la cual mapea los recursos dentro de un módulo hacia la carpeta donde está definida. Para la carpeta principal application/ vamos a utilizar el prefijo Application_.

El autoloader mapea recursos a directorios utilizando este mapeo:

Prefijo	Carpeta
Form	forms
Model	models
Model_DbTable	models/DbTable
Model_Mapper	models/mappers
Plugin	plugins
Service	services
View_Filter	views/filters
View_Helper	views/helpers

Ya que estamos nombrando a partir de la tabla, los discos (albums) que utilicen Zend_Db_Table van a tener una clase llamada Application_Model_DbTable_Albums la cual estará guardad enapplications/models/DbTable/Albums.php.

Para poder decirle a Zend_Db_Table el nombre de la tabla que vamos a administrar, debemos crear la variable protegida $_name con el nombre de la tabla. Además, Zend_Db_Table asume que to tabla tiene un primary key llamado id, el cual es auto-incrementado por la base de datos. El nombre de este campo también puede ser cambiado si es requerido.

Podemos utilizar la herramienta **zf** para hacer algo del trabajo, por lo que vamos a pasar el siguiente comando:

```
zf create db-table Albums albums
```

La herramienta creó el archivo Albums.php en la carpeta application/models/DbTable/. Dentro del archivo se encuentra una clase llamada Application_Model_DbTable_Albums la cual tiene declarado el nombre de la tabla con la que se va a comunicar a través de sus métodos.

Ahora necesitamos algo de funcionalidad, por lo que vamos a editar el archivo application/models/DbTable/Albums.php y agregar los métodos getAlbum(), addAlbum(), updateAlbum() y deleteAlbum(). El archivo se va a ver así:

zf-tutorial/application/models/DbTable/Albums.php

```
<?php


class Application_Model_DbTable_Albums extends Zend_Db_Table_Abstract

{

    protected $_name = 'albums';


    public function getAlbum($id)

    {

        $id = (int)$id;

        $row = $this->fetchRow('id = ' . $id);

        if (!$row) {

            throw new Exception("Could not find row $id");

        }

        return $row->toArray();
```

```
    }

    public function addAlbum($artist, $title)
    {
        $data = array(
            'artist' => $artist,
            'title' => $title,
        );
        $this->insert($data);
    }

    public function updateAlbum($id, $artist, $title)
    {
        $data = array(
            'artist' => $artist,
            'title' => $title,
        );
        $this->update($data, 'id = ' . (int)$id);
    }

    public function deleteAlbum($id)
    {
        $this->delete('id = ' . (int)$id);
    }
}
```

Acabamos de crear cuatro métodos que nuestra aplicación va a utilizar para armar una interface de comunicación con la tabla relacionada. getAlbum() devuelve un solo registro dentro de un array, addAlbum() crea un registro en la base de datos, updateAlbum() actualiza el registro de un disco y deleteAlbum() remueve el registro completamente. El código para cada uno de estos métodos se explica por si mismos. Aunque no sea necesario en este tutorial, también podés decirle a Zend_Db_Table que existen otras tablas relacionadas y que también puede devolver información relacionada.

Necesitamos llenar el controlador con la información del modelo y armar las vistas para mostrarla, aunque, antes de hacer esto, tenemos que entender como funciona el sistema de vistas en Zend Framework.

LAYOUTS Y VISTAS

El componente de Zend Framework para el manejo de vistas es llamado, sin sorpresa alguna, Zend_View. El componente de la vista nos va a permitir separar el código que muestra la página del código que está en las funciones de las acciones.

El uso básico de Zend_View es:

```
$view = new Zend_View();
$view->setScriptPath('/path/to/scripts');
echo $view->render('script.php');
```

Se puede ver muy fácil que si vamos a escribir este código directamente en cada una de las acciones vamos a estar repitiendo código "estructural" muy aburrido que no es de uso específico de la acción. Preferimos realizar la inicialización de la vista en otro lado y luego acceder al ya inicializado objeto de la vista en cada acción. Zend Framework provee de un Action Helper (una función que se utiliza muchas veces) llamada ViewRenderer. Este se ocupa de inicializar la vista en el controlador ($this->view) para que nosotros la utilicemos y además renderice el código de la vista al terminar de enviar la respuesta de la acción.

Para el renderizado, el ViewRenderer define que el objeto Zend_View busque en views/scripts/{controller name} el código de vista que se va a mostrar y (por defecto) va a elegir mostrar el archivo que se llame como la acción que se está ejecutando y que termine con la extensión phtml. Entonces, la vista que va a ser renderizada es views/scripts/{controller name}/{action name}.phtml y el contenido renderizado se va a agregar al cuerpo del objeto de respuesta (u objeto Response). El objeto Response se utilia para recopilar todas las cabeceras (headers) del protocolo HTTP, contenido del cuerpo y excepciones generadas en el resultado del sistema MVC. El Front Controller luego envia automaticamente las cabeceras, seguidas por el contenido (body) al final de la respuesta para el cliente.

Esto se genera solo gracias a Zend_Tool cuando creamos el proyecto y agregamos los controladores y las acciones utilizando los comandos zf create controller y zf create action.

CÓDIGO HTML REPETIDO: LAYOUTS

Algo que se hace demasiado obvio es que va a existir un montón de código HTML repetido durante el desarrollo de cada proyecto en nuestras vistas, como mínimo el encabezado (header) y el pie de página (footer), y tal vez algún menú o dos. Este es un problema muy común y el componente Zend_Layout está diseñado para resolver este problema. Zend_Layout nos permite mover todo el código del encabezado, pie de página y demás código repetido a una vista de layout, el cual luego incluirá el código de la vista específico de la acción que ejecutemos.

El lugar por defecto para guardar nuestros layouts es en application/layouts/ y hay un recurso disponbile para Zend_Application que configura al Zend_Layout por nosotros. Vamos a utilizar Zend_Tool para crear el archivo del layout y actualizar application.ini apropiadamente. Nuevamente, desde la Terminal o el Símbolo de Sistema, escribimos el siguiente comando en la carpeta de zf-tutorial:

```
zf enable layout
```

Zend_Tool acaba de crear la carpeta application/layouts/scripts/ y dentro de la misma un archivo llamado layout.phtml. También actualizó el archivo application.ini y agregó la línea resources.layout.layoutPath = APPLICATION_PATH "/layouts/scripts/" a la sección [production].

Al final del ciclo de respuesta, antes de que la acción del controlador termine sus tareas, Zend_Layout va a renderizar nuestro layout. Zend_Tool provee un layout bastante básico que solo muestra el contenido de la vista de la acción. Nosotros vamos a agregarle más contenido HTML, requerido por nuestro sitio web. Abrimos el archivo layout.phtml y reemplazamos el código que existente por este otro:

zf-tutorial/application/layouts/scripts/layout.phtml

```
<?php

$this->headMeta()->appendHttpEquiv('Content-Type', 'text/html;charset=utf-8');

$this->headTitle()->setSeparator(' - ');

$this->headTitle('Zend Framework Tutorial');



echo $this->doctype(); ?>

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>

    <?php echo $this->headMeta(); ?>

    <?php echo $this->headTitle(); ?>

</head>

<body>

<div id="content">

    <h1><?php echo $this->escape($this->title); ?></h1>

    <?php echo $this->layout()->content; ?>

</div>

</body>

</html>
```

El archivo de layout contiene el código HTML que se encuentra "afuera", el cual es bastante común. Como es un archivo de PHP normal, podemos utilizar PHP en el. Hay una variable disponible ($this) la cual es una instancia del objeto de la vista, el cual fue creado durante el bootstrapping. Podemos utilizar esta variable para mostrar información que fue asignada a la vista y también llamar métodos. Los métodos (también conocidos como view helpers) devuelven texto (string) que podemos mostrar en pantalla utilizando echo.

Primeramente configuramos algunos view helpers para la sección del encabezado de la página web y luego imprimimos el doctype correcto. Dentro del <body>, creamos un div con un <h1> conteniendo el título. Para poder mostrar la respuesta de la acción que estamos ejecutando, hacemos uso del view helper que imprime el contenido correspondiente, llamado layout(): echo $this->layout()->content; el cual realiza el trabajo por nosotros. Esto significa que las vistas de las acciones se ejecutan antes del código del layout.

Necesitamos configurar el doctype para la página web antes de renderizar cualquier vista. Como las vistas de las acciones son renderizadas antes, necesitamos conocer que doctype vamos a utilizar. Esto es necesario para cuando utilicemos Zend_Form.

Para configurar el doctype vamos a agregar una línea en nuestro application.ini, en la sección de [production]:

```
resources.view.doctype = "XHTML1_STRICT"
```

El view helper doctype() ahora va a devolver el doctype correcto y los componentes con código HTML correcto como ser los elementos autogenerados de Zend_Form.

ESTILOS

Aunque este es "solo" un tutorial, vamos a necesitar un archivo CSS para que podamos hacer nuestra aplicación un poco más presentable. Esto puede ser un problema ya que no sabemos como referenciar correctamente al archivo CSS, ya que la URL no apunta a una ruta real. Afortunadamente, un view helper llamado baseUrl() está disponible para nuestro uso. Esta función guarda la información de las URLs de nuestra aplicación y nos devuelve la parte de URL que no conocemos.

Ahora podemos agregar archivos CSS en la sección <head> en el archivo application/layouts/scripts/layout.phtml, utilizando la función headLink():

zf-application/application/layouts/scripts/layout.phtml

```
...
<head>
    <?php echo $this->headMeta(); ?>
    <?php echo $this->headTitle(); ?>
    <?php echo $this->headLink()->prependStylesheet($this->baseUrl() . '/css/site.css'); ?>
</head>
...
```

Usando el método prependStylesheet() de headLink(), podemos agregar más archivos CSS dentro del código de la vista para cada controlador, dentro de la sección <head>, luego de site.css.

Finalmente, necesitamos agregar los estilos de CSS, por lo que vamos a crear una carpeta llamada css dentro de la carpeta public/ y crear el archivo site.css con el siguiente código:

zf-tutorial/public/css/site.css

```
body,html {
    margin: 0 5px;
    font-family: Verdana,sans-serif;
}
h1 {
    font-size: 1.4em;
    color: #008000;
}
a {
    color: #008000;
}

/* Table */
```

```
th {
    text-align: left;
}
td, th {
    padding-right: 5px;
}

/* style form */
form dt {
    width: 100px;
    display: block;
    float: left;
    clear: left;
}
form dd {
    margin-left: 0;
    float: left;
}
form #submitbutton {
    margin-left: 100px;
}
```

Esto debería hacer que se vea un poco mejor, pero como puedes ver, no soy diseñador.

Ahora podemos preparar las cuatro acciones que fueron auto generadas, por lo que vamos a vaciar los archivos index.phtml, add.phtml, edit.phtml y delete.phtml, los cuales sin duda recuerdas, se encuentran en la carpetaapplication/views/scripts/index/.

LISTANDO DISCOS

Ahora que terminamos de armar la configuración, la información de la base de datos y el esqueleto de nuestras vistas, podemos ir al centro de nuestra aplicación y mostrar algunos discos. Esto se lleva a cabo en la claseIndexController y vamos a comenzar listando los discos en una tabla a partir de la función indexAction():

zf-tutorial/application/controllers/IndexController.php

```
...
public function indexAction()
{
    $albums = new Application_Model_DbTable_Albums();
    $this->view->albums = $albums->fetchAll();
}
...
```

Instanciamos una instancia de nuestra tabla a traves de modelo basado en la ella. La función fetchAll() devuelve un objeto del tipo Zend_Db_Table_Rowset que nos permite iterar sobre los registros devueltos el código de la vista de la acción.

Ahora podemos llenar el código de la vista asociada, el archivo index.phtml:

zf-tutorial/application/views/scripts/index/index.phtml

```
<?php
$this->title = "My Albums";
$this->headTitle($this->title);
?>
<p><a href="<?php echo $this->url(array('controller'=>'index',
        'action'=>'add'));?>">Add new album</a></p>
<table>
<tr>
    <th>Title</th>
    <th>Artist</th>
    <th> </th>
</tr>
<?php foreach($this->albums as $album) : ?>
<tr>
    <td><?php echo $this->escape($album->title);?></td>
    <td><?php echo $this->escape($album->artist);?></td>
    <td>
        <a href="<?php echo $this->url(array('controller'=>'index',
            'action'=>'edit', 'id'=>$album->id));?>">Edit</a>
        <a href="<?php echo $this->url(array('controller'=>'index',
            'action'=>'delete', 'id'=>$album->id));?>">Delete</a>
    </td>
</tr>
<?php endforeach; ?>
</table>
```

La primer cosa que hacemos es crear el título para la página (usada en el layout) y también para el título de headTitle() en <head>, el cual se muestra en la barra del navegador. Luego creamos un link para agregar un nuevo disco. La función url() la provee el framework y nos ayuda a crear links con la URL correcta. Simplemente pasamos un array con los parámetros necesarios, y solo termina de armar el resto del link.

Luego creamos una tabla que va a mostrar el título y el artista de cada disco, junto con los links para poder editar y eliminar cada registro. Usamos un foreach: común para iterar sobre el listado de discos, y utilizamos el formato alternativo para las vistas terminando con endforeach;, ya que de esta forma es mucho más cómodo para leer que tratar de ubicar cuando comienza una llave y cuando termina la misma. Nuevamente, hacemos uso de la función url() para crear los links para editar y eliminar.

Si abres http://localhost/zf-tutorial/public/ (o la url que hayas armado para este tutorial) deberías ver un lindo listado de discos, algo parecido a esto:

AGREGANDO NUEVOS DISCOS

Vamos a programar la funcionalidad para agregar nuevos discos. Hay dos cosas que hacer aquí:

- Brindar un formulario para poder tomar los detalles.
- Procesar el formulario una vez enviado y luego guardar la información en la base de datos.

Para esto vamos a hacer uso de Zend_Form. El componente Zend_Form nos permite crear formularios y validar la información que recibe. Creamos una nueva clase llamada Form_Album que extiende de Zend_Formpara definir nuestro formulario. Como esto es un recurso de la aplicación, la clase va a estar guardada en el archivo Album.php dentro de la carpeta forms. Comenzamos utilizando el comando zf para crear el archivo correctamente:

```
zf create form Album
```

Esto crea el archivo Album.php en la carpeta application/forms/ e incluye un método llamado init() donde podemos configurar el formulario y agregar los elementos que necesitamos. Editemos el archivoapplication/forms/Album.php y eliminamos el comentario dentro del método init() para poder agregar lo siguiente:

zf-tutorial/application/forms/Album.php

```
<?php


class Application_Form_Album extends Zend_Form

{


    public function init()

    {

        $this->setName('album');


        $id = new Zend_Form_Element_Hidden('id');

        $id->addFilter('Int');


        $artist = new Zend_Form_Element_Text('artist');

        $artist->setLabel('Artist')

               ->setRequired(true)

               ->addFilter('StripTags')

               ->addFilter('StringTrim')

               ->addValidator('NotEmpty');


        $title = new Zend_Form_Element_Text('title');

        $title->setLabel('Title')

               ->setRequired(true)

               ->addFilter('StripTags')

               ->addFilter('StringTrim')

               ->addValidator('NotEmpty');


        $submit = new Zend_Form_Element_Submit('submit');

        $submit->setAttrib('id', 'submitbutton');


        $this->addElements(array($id, $artist, $title, $submit));

    }

}
```

Dentro del método init() de la clase Application_Form_Album creamos cuatro elementos: el id, el artista, el título y el botón de submit. Para cada elemento le asignamos varios atributos, incluído el título (label) que va a mostrar cada uno. Para el id, queremos asegurar que el valor que puede tener asignado sea un entero, para prevenir cualquier potencial ataque de SQL injection. El filtro Int se va a ocupar de ello por nosotros.

Para los elementos de texto, vamos a agregar dos filtros, StripTags y StringTrim para remover código HTML no deseado y espacios innecesarios en blanco. Además los configuramos para que sean requeridos y al agregarle un validador del tipo NotEmpty nos aseguramos que el usuario realmente

ingrese la información que requerimos. (el validador NotEmpty no es requerido técnicamente ya que va a ser agregado automáticamente al utilizar la función setRequired() con el parámetro true; está aquí presente para demostrar cómo utilizar un validador.)

Ahora vamos a necesitar que el formulario se muestre y luego vamos a tener que procesar la información enviada. Esto se realiza en la función addAction() de la clase IndexController:

zf-tutorial/application/controllers/IndexController.php

```
...
public function addAction()
{
    $form = new Application_Form_Album();
    $form->submit->setLabel('Add');
    $this->view->form = $form;

    if ($this->getRequest()->isPost()) {
        $formData = $this->getRequest()->getPost();
        if ($form->isValid($formData)) {
            $artist = $form->getValue('artist');
            $title = $form->getValue('title');
            $albums = new Application_Model_DbTable_Albums();
            $albums->addAlbum($artist, $title);

            $this->_helper->redirector('index');
        } else {
            $form->populate($formData);
        }
    }
}
...
```

Vamos a examinar el código un poco más en detalle:

```
$form = new Application_Form_Album();
$form->submit->setLabel('Add');
$this->view->form = $form;
```

Instanciamos nuestro Form_Album, configuramos el título (label) del botón de submit para que sea "Add" y luego asignamos el formulario a la vista que vamos a renderizar.

```
if ($this->getRequest()->isPost()) {
    $formData = $this->getRequest()->getPost();
    if ($form->isValid($formData)) {
```

Si la respuesta del método isPost() es true, entonces el formulario fue enviado por lo que vamos a tomar los valores recibidos con el método getPost() y vamos a verificar que los datos sean válidos utilizando la funciónisValid().

```
$artist = $form->getValue('artist');
$title = $form->getValue('title');
$albums = new Application_Model_DbTable_Albums();
```

```
$albums->addAlbum($artist, $title);
```

Si el formulario es válido, vamos a instanciar la clase del modelo Application_Model_DbTable_Albums y usamos el método addAlbum() que creamos anteriormente para poder guardar un nuevo registro en la base de datos.

```
$this->_helper->redirector('index');
```

Luego de guardar el nuevo disco, vamos a redirigir al usuario con el action helper Redirector para ir a la acción index (por ejemplo, vamos a ir a la página de inicio).

```
} else {
    $form->populate($formData);
}
```

Si los datos del formulario no son válidos, vamos a popular (rellenar) el formulario con la información que brindó el usuario y lo volvemos a mostrar.

Ahora necesitamos mostrar el formulario en la vista add.phtml:

zf-tutorial/application/views/scripts/index/add.html

```
<?php
$this->title = "Add new album";
$this->headTitle($this->title);
echo $this->form;
?>
```

Como pueden ver, mostrar un formulario es bastante simple, solo debemos imprimirlo utilizando echo, ya que el fomulario mismo sabe como mostrarse. Ahora deberías poder utilizar el link "Add new album" de la página de inicio de la aplicación para poder agregar nuevo disco.

EDITANDO UN DISCO

Editar un disco es prácticamente idéntico a agregar uno, por lo que el código es bastante similar:

zf-tutorial/application/controllers/IndexController.php

```
...
public function editAction()
{
    $form = new Application_Form_Album();
    $form->submit->setLabel('Save');
    $this->view->form = $form;

    if ($this->getRequest()->isPost()) {
        $formData = $this->getRequest()->getPost();
        if ($form->isValid($formData)) {
            $id = (int)$form->getValue('id');
            $artist = $form->getValue('artist');
            $title = $form->getValue('title');
            $albums = new Application_Model_DbTable_Albums();
            $albums->updateAlbum($id, $artist, $title);

            $this->_helper->redirector('index');
        } else {
```

```
            $form->populate($formData);
        }
    } else {
        $id = $this->_getParam('id', 0);
        if ($id > 0) {
            $albums = new Application_Model_DbTable_Albums();
            $form->populate($albums->getAlbum($id));
        }
    }
}
...
```

Miremos las diferencias con el código que agrega un nuevo disco. Primeramente, cuando mostramos el formulario al usuario vamos a necesitar tomar la información del disco (artista y título) de la base de datos y luego populamos los elementos del formulario con la información. Esto sucede al final del método:

```
$id = $this->_getParam('id', 0);
if ($id > 0) {
    $albums = new Application_Model_DbTable_Albums();
    $form->populate($albums->getAlbum($id));
}
```

Hay que entender que esto sucede si el envío no es hecho a través de POST, ya que ser enviado por POST implica que ya completamos el formulario y ahora queremos procesarlo. Para mostrar el formulario por primera vez, vamos a leer el id solicitado utilizando el método _getParam(). Luego utilizamos el modelo para levantar la información de la base de datos y populamos el formulario directamente con la información del registro. (Ahora pueden ver por qué el método getAlbum() del modelo devuelve un array.)

Luego de validar el formulario, necesitamos guardar la información actualizada en el registro correcto. Esto es hecho utilizando el método updateAlbum() del modelo:

```
$id = (int)$form->getValue('id');
$artist = $form->getValue('artist');
$title = $form->getValue('title');
$albums = new Application_Model_DbTable_Albums();
$albums->updateAlbum($id, $artist, $title);
```

La vista es igual a add.phtml:

zf-tutorial/application/views/scrvipts/index/edit.phtml

```
<?php
$this->title = "Edit album";
$this->headTitle($this->title);
echo $this->form;
?>
```

Ahora deberías poder editar discos.

ELIMINANDO UN DISCO

Para poder cerrar nuestra aplicación, vamos a necesitar agregar la función de eliminar discos. Ya tenemos un link para eliminar discos al lado de cada uno en nuestra lista y lo más inocente que podríamos hacer es permitir al usuario que elimine el registro al instante que le hace click al link. Esto estaría mal. Recordando la especificación de HTTP, nos damos cuenta que no deberíamos realizar una acción irreversible utilizando GET, por lo que deberíamos utilizar POST en su lugar.

Deberíamos mostrar un formulario de confirmación para que el usuario haga click una vez más y confirme que "si", quiere eliminar el registro. Como el formulario es trivial, vamos a escribirlo directamente en nuestra vista (Zend_Form es opcional, después de todo).

Vamos a comenzar con el código de la acción IndexController::deleteAction():

zf-tutorial/application/controlllers/IndexController.php

```
...
public function deleteAction()
{
    if ($this->getRequest()->isPost()) {
        $del = $this->getRequest()->getPost('del');
        if ($del == 'Yes') {
            $id = $this->getRequest()->getPost('id');
            $albums = new Application_Model_DbTable_Albums();
            $albums->deleteAlbum($id);
        }
        $this->_helper->redirector('index');
    } else {
        $id = $this->_getParam('id', 0);
        $albums = new Application_Model_DbTable_Albums();
        $this->view->album = $albums->getAlbum($id);
    }
}
...
```

Como en add y edit, utilizamos el método isPost() del Request para determinar si debemos mostrar el formulario de confirmación o si debemos eliminar el registro. Usamos el modelo Application_Model_DbTable_Albumspara realmente eliminar el registro usando el método deleteAlbum(). Si el request no es un POST, buscamos por el parámetro id y devolvemos el registro correcto y lo asignamos a la vista.

El código de la vista es un formulario simple:

zf-tutorial/application/views/scripts/index/delete.phtml

```
<?php
$this->title = "Delete album";
$this->headTitle($this->title);
?>
<p>Are you sure that you want to delete
  '<?php echo $this->escape($this->album['title']); ?>' by
  '<?php echo $this->escape($this->album['artist']); ?>'?
</p>
<form action="<?php echo $this->url(array('action'=>'delete')); ?>" method="post">
<div>
  <input type="hidden" name="id" value="<?php echo $this->album['id']; ?>" />
  <input type="submit" name="del" value="Yes" />
  <input type="submit" name="del" value="No" />
</div>
```

```
</form>
```

En este código, mostramos un mensaje de confirmación al usuario y luego un formulario con las opciones "Yes" o "No". En la acción verificamos específicamente el valor "Yes" para poder eliminar el registro.

Eso es todo. Ahora tienes una aplicación completamente funcional.

CONCLUSIÓN

Esto concluye nuestra primera visión de cómo construir una aplicación MVC simple (pero funcional) utilizando Zend Framework. Espero que lo hayas encontrado interesante e informativo. Si encuentras algo que esté mal, por favor escríbeme un email a rob@akrabat.com!

Este tutorial muestra el uso básico del framework; hay muchos componentes más para explorar! También pasé por encima varios puntos básicos que no expliqué. Mi sitio web http://akrabat.com contiene varios artículos sobre Zend Framework, y además deberías leer el manual en http://framework.zend.com/manual!

Finalmente, si preferís algo impreso, escribí un libro llamado *Zend Framework in Action* el cual pueden comprarlo. Más detalles del mismo están dispomibles en http://www.zendframeworkinaction.com. Échale un vistazo. ☺