Carlos Alberto Alegría

Tutorial: realizar CRUD con Zend – Framework para php.

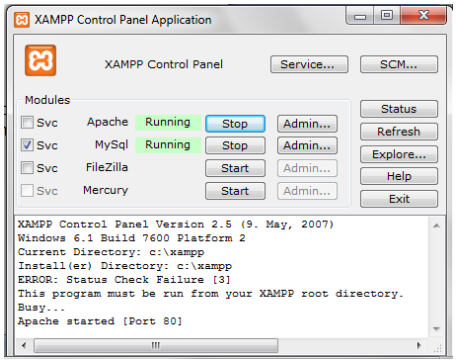
Entorno de Programación: Eclipse PHP.

Servidor de prueba: Xampp.

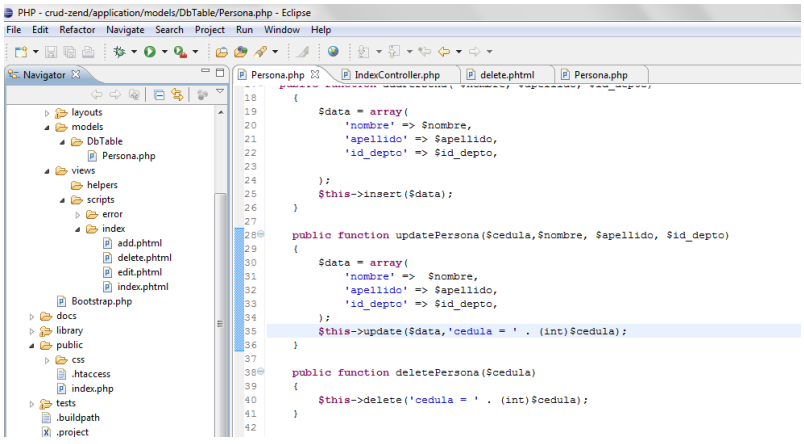
Sistema operativo: Windows 7.

Precondiciones:

- Instalar Xampp contiene el servidor Apache necesario para correr tus archivos php, y un servidor MySql necesario para nuestra base de datos. Servidores esenciales para crear nuestra aplicación CRUD.



- Instalar Eclipse para php. Para escribir tus programas php.



Características de Zend:

Cuenta con soporte para internalización y localización de aplicaciones construir sitios multi-idioma, convertir formatos de fechas, monedas, etc.

Facilita el setup y brinda herramientas para crear la estructura de directorios y clases por línea de comandos.

Tiene adapters para diversos tipos de bases de datos, brinda componentes para la autenticación y autorización de usuarios, envío de mails entre otros.

La idea es invertir menos tiempo en el desarrollo y hacer uso de componentes ya testeados.

1. Consiguiendo el Framework ,instalación y creación de un proyecto.

La versión del framework con la cual trabajo es la 1.11.11 si usas un framework anterior o más actual lee la documentación. Suelen haber cambios.

El sitio de descarga es el siguiente:

http://framework.zend.com/download/archives

Una vez descargado el framework (archico.zip) lo descomprimimos y lo copiamos en la carpeta de php contenida en Xampp esto es porque es un framework de php, la ruta seria C:\xampp\php.

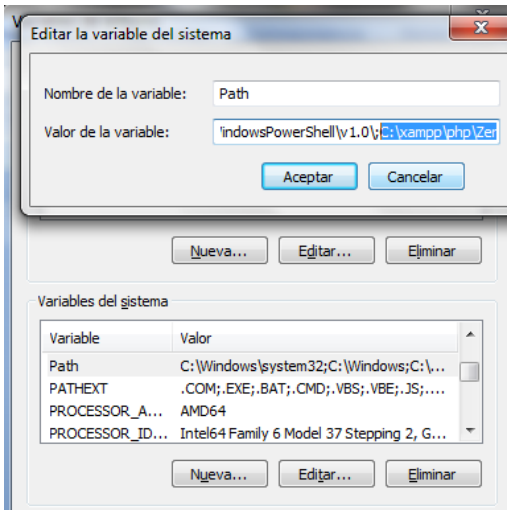
Configurando las variables de entorno:

Esto se hace para poder usar la línea de comandos de Windows para generar proyectos, controladores, vistas etc.

En la variable Path: pegas las siguientes rutas:

;C:\xampp\php\ZendFramework\bin;C:\xampp\php

No te olvides del ; es obligatorio cuando añades un Nuevo valor, si es el ultimo valor no lleva ; al final.



Editamos el archivo php.ini en la carpeta php la ruta es la siguiente: C:\xampp\php con el fin de incluir la librería de Zend y nuestro aplicación que usa el framework funcione el paso es el siguiente (suponiendo que todo lo que has hecho hasta ahora este bien):

Fijamos la librería que esta esta en C:\xampp\php\ZendFramework\library en el archivo php.ini en la siguiente región (path):

```
; Always populate the $HTTP_RAW_POST_DATA variable. PHP's
default behavior is
; to disable this feature.
; http://php.net/always-populate-raw-post-data
;always_populate_raw_post_data = On

;;;;;;;;;;;;;;;;;;;;;;;;;
; Paths and Directories ;
;;;;;;;;;;;;;;;;;;;;;;;;;

; UNIX: "/path1:/path2"
;include_path = ".:/php/includes"
;
; Windows: "\path1;\path2"
include_path = ".;C:\xampp\php\PEAR"
include_path = ".;C:\xampp\php\ZendFramework\library"

;
; PHP's default setting for include_path is
".;/path/to/php/pear"
; http://php.net/include-path
```

Y luego procedemos a copiar :

```
include_path = ".;C:\xampp\php\ZendFramework\library"
```

en el archivo php.ini, y listo apache ya nos puede interpretar zend.

Configuración del nuestro proyecto:

Para crear nuestro proyecto vamos al símbolo de sistema de Windows nos situamos en

`C:\xampp\htdocs` Y tecleamos el siguiente comando: zf create project nombre_proyecto

```
C:\xampp\htdocs>zf create project crudzend
Creating project at C:/xampp/htdocs/crudzend
Note: This command created a web project, for more information s
HOST, please see docs/README
Testing Note: PHPUnit was not found in your include_path, theref
ctions will be created.

C:\xampp\htdocs>
```

Se crea el árbol de carpetas del patrón MVC en la carpeta htdocs.

En la carpeta library del proyecto que acabamos de crear copiamos la carpeta zend que se encuentra en : C:\xampp\php\ZendFramework\library

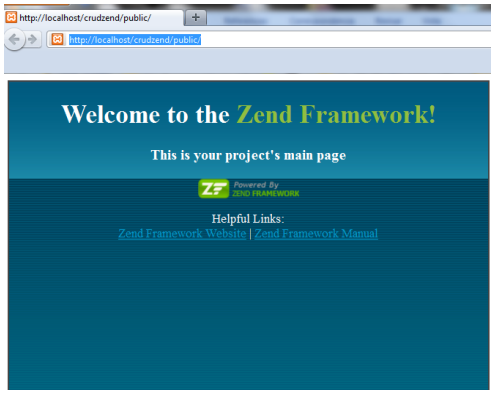
Y listo tenemos configurado completamente nuestro proyecto.

Creando el proyecto en Eclipse php

Creamos un proyecto php en eclipse y luego copiamos el árbol de carpetas generado que está dentro de tu proyecto Zend y listo ya podemos dedicarnos a desarrollar nuestra aplicación.

Verificando:

Si todo está bien, inicias el servidor de apache en Xampp y al escribir la siguiente ruta en el navegador http://localhost/tuproyecto/public/ te debe aparecer una página de bienvenida de Zend.



2. Creado la base de datos, establecimiento del modelo.

En el patrón MVC la base de datos no proporciona el modelo de nuestra aplicación. El modelo es la parte que se ocupa con la parte principal del proyecto (las reglas de negocio) 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.

Se crea la base de datos desde phpmyadmin para esto escribe esta url en el navegador:

http://localhost/phpmyadmin/

creas la base de datos bd_zend (base de datos que uso en este ejemplo)

y corres los siguientes escript sql:

Creando la tabla persona:

```
CREATE TABLE IF NOT EXISTS `persona` (
  `cedula` int(20) NOT NULL AUTO_INCREMENT,
  `nombre` varchar(30) NOT NULL,
  `apellido` varchar(30) NOT NULL,
  `id_depto` int(20) NOT NULL,
  PRIMARY KEY (`cedula`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=3 ;
```

Haciendo un insert:

```
INSERT INTO `persona` (`cedula`, `nombre`, `apellido`, `id_depto`)
VALUES
(1, 'carlos', 'alegria', 123),
(2, 'sofia', 'cardona', 789);
```

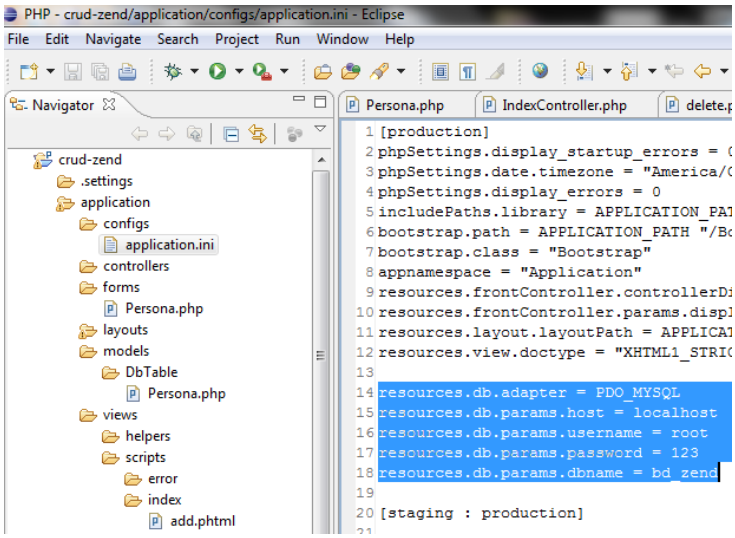
Conectando la base de datos con nuestra aplicación:

Para usar Zend_Db_Table, necesitamos decir que base de datos usar junto con un usuario y una contraseña. 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 application.ini y el sistema hará el resto. Para esto hacemos lo siguiente:

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

```
resources.db.adapter = PDO_MYSQL
resources.db.params.host = localhost
resources.db.params.username = root
resources.db.params.password = 123
resources.db.params.dbname = bd_zend
```

Obviamente deberías usar tu nombre de usuario, contraseña y nombre de base de datos, esta información la puedes encontrar en phpmyadmin.



Modelo:

Para este tutorial vamos a crear un modelo que extiende de Zend_Db_Table y utiliza Zend_Db_Table_Row(clase que contiene una fila individual de un objeto Zend_Db_Table).

Zend_Db_Table_Abstract es una clase abstracta, de la cual vamos a derivar nuestra clase específica para administrar personas. No importa como nombremos a nuestra clase, pero tiene sentido que la nombremos con relación a la tabla.

Ya que estamos nombrando a partir de la tabla, las personas (persona) que utilicen Zend_Db_Table van a tener una clase llamada Application_Model_DbTable_Persona la cual estará guardad en applications/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.

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

Primero nos situamos en C: /Xampp/htdocs/nuestroProyecto

C:\xampp\htdocs\crud-zend>zf create db-table Persona persona

La herramienta creó el archivo Persona.php en la carpeta application/models/DbTable/. Dentro del archivo se encuentra una clase llamada Application_Model_DbTable_Persona 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/Persona.php y agregar los métodos getPersona(), addPersona(), updatePersona() y deletePersona()

Código:

```php
<?php

class Application_Model_DbTable_Persona extends Zend_Db_Table_Abstract
{
    protected $_name = 'persona';

    public function getPersona($cedula)
    {
      $cedula=(int)$cedula;
      $row=$this->fetchRow('cedula = '.$cedula);
      if(!$row)
      {
            throw  new  Exception("No se puedo encontrar la fila
$cedula");
      }
      return $row->toArray();
    }
  public function addPersona( $nombre, $apellido, $id_depto)
    {
        $data = array(
            'nombre' => $nombre,
            'apellido' => $apellido,
            'id_depto' => $id_depto,

        );
        $this->insert($data);
    }

    public function updatePersona($cedula,$nombre, $apellido, $id_depto)
    {
        $data = array(
            'nombre' =>  $nombre,
            'apellido' => $apellido,
            'id_depto' => $id_depto,
        );
        $this->update($data,'cedula = ' . (int)$cedula);
    }

    public function deletePersona($cedula)
    {
        $this->delete('cedula = ' . (int)$cedula);
    }

}
```

Necesitamos llenar el controlador con la información del modelo y armar las vistas para mostrarla.

3. Creando los Controladores

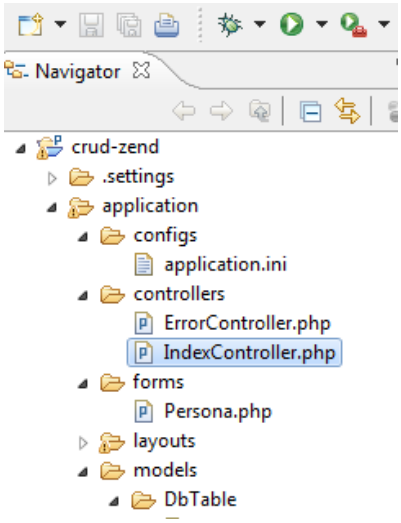
Esta aplicación contiene contiene un controlador con cuatro acciones (metodods) index,add, edit, delete.

Ejemplo de cómo se ve en la url:

Public/index/add

Public/index/edit

Donde el controlador es index(el FontController) y las acciones son add y edit.



Creamos las acciones. Abrimos el Terminal o el Símbolo de Sistema y nos posicionamos en el directorio del proyecto (crud-zend). Luego escribir estos comandos:

zf create action add Index

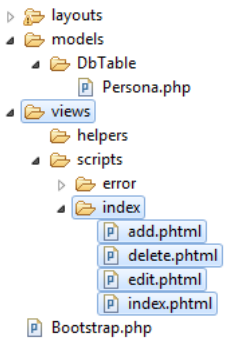
zf create action edit Index

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, con sus respectivas vistas.

El código que va en cada acción del controlador lo vamos creando a medida que vamos desarrollando el crud (adicionar,eliminar,listar,editar)

4. Vistas(layuots)



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 crud-zend:

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:

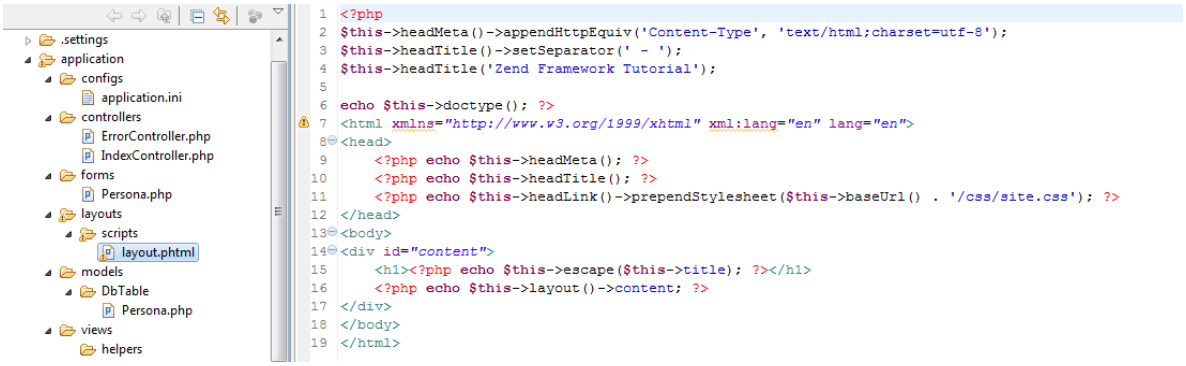
```
<?php
$this->headMeta()->appendHttpEquiv('Content-Type',
'text/html;charset=utf-8');
$this->headTitle()->setSeparator(' - ');
$this->headTitle('Zend Framework Tutorial');

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>
```

Estilos

Usamos 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():

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

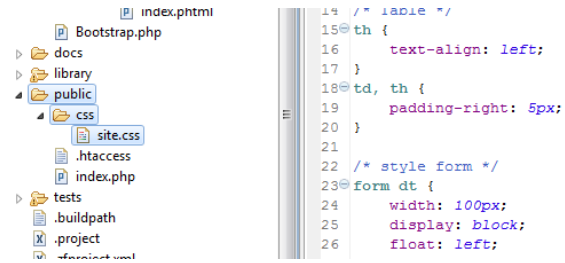


Usando el método prependStylesheet() de headLink(), podemos agregar más archivos CSS dentro del código de la vista para cada controlador, dentro de la sección <head>, luego de site.css

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:

```
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;
}
```



5. creación del crud

Listar personas:

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 las personas. Esto se lleva a cabo en la clase IndexController y vamos a comenzar listando los discos en una tabla a partir de la función indexAction():

```
public function indexAction()
    {
        $persona = new Application_Model_DbTable_Persona();
        $this->view->persona = $persona->fetchAll();
    }
```

Instanciamos nuestra tabla persona 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:

```
<?php
$this->title = "Personas";
$this->headTitle($this->title);
?>
<p><a href="<?php echo $this->url(array('controller'=>'index',
        'action'=>'add'));?>">Agregar una persona</a></p>
```

```
<table>
<tr>
    <th>Cedula</th>
    <th>Nombre</th>
    <th>Apellido</th>
    <th>Id Departamento</th>
    <th> </th>
</tr>
<?php foreach($this->persona as $personas) : ?>
<tr>
    <td><?php echo $this->escape($personas->cedula);?></td>
    <td><?php echo $this->escape($personas->nombre);?></td>
    <td><?php echo $this->escape($personas->apellido);?></td>
    <td><?php echo $this->escape($personas->id_depto);?></td>
    <td>
        <a href="<?php echo $this->url(array('controller'=>'index',
            'action'=>'edit', 'cedula'=>$personas->cedula));?>">Edit</a>
        <a href="<?php echo $this->url(array('controller'=>'index',
            'action'=>'delete', 'cedula'=>$personas-
>cedula));?>">Delete</a>
    </td>
</tr>
<?php endforeach; ?>
</table>
```

Deberá de verse asi:

Al digitar las siguiente url: http://localhost/crud-zend/public/

Personas

Agregar una persona

Cedula	Nombre	Apellido	Id Departamento	
1	carlos	alegria	123	Edit Delete
2	sofia	cardona	789	Edit Delete

Adicionar personas:

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

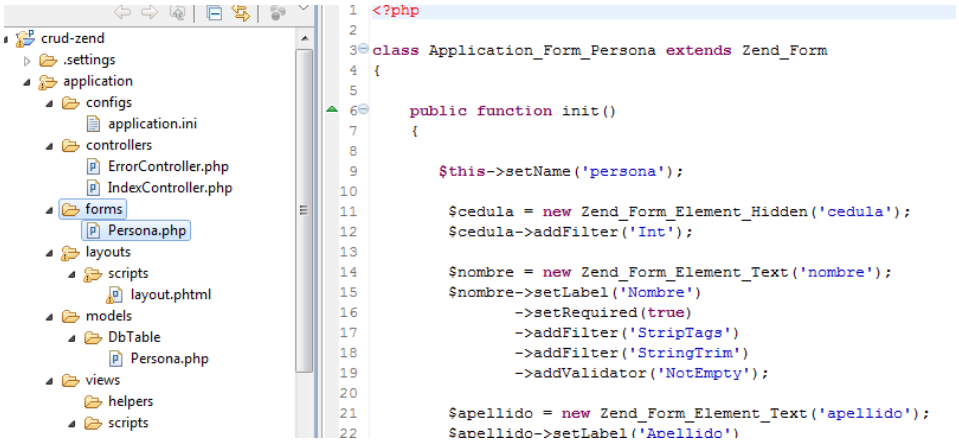
- Brindar un formulario para poder tomar los detalles.
- Procesar el formulario una vez enviado y luego guardar la información en la base de datos.

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_Form para 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 archivo application/forms/Album.php y eliminamos el comentario dentro del método init() para poder agregar lo siguiente:

```php
<?php

class Application_Form_Persona extends Zend_Form
{

    public function init()
    {

        $this->setName('persona');

         $cedula = new Zend_Form_Element_Hidden('cedula');
         $cedula->addFilter('Int');

         $nombre = new Zend_Form_Element_Text('nombre');
         $nombre->setLabel('Nombre')
                ->setRequired(true)
                ->addFilter('StripTags')
                ->addFilter('StringTrim')
                ->addValidator('NotEmpty');

         $apellido = new Zend_Form_Element_Text('apellido');
         $apellido->setLabel('Apellido')
                ->setRequired(true)
                ->addFilter('StripTags')
                ->addFilter('StringTrim')
                ->addValidator('NotEmpty');

         $id_depto = new Zend_Form_Element_Text('id_depto');
```

```
        $id_depto->setLabel("Departamento")
                 ->setRequired(true)
                 ->addFilter('StripTags')
                 ->addFilter('StringTrim')
                 ->addValidator('NotEmpty');

        $submit = new Zend_Form_Element_Submit('submit');
        $submit->setAttrib('cedula','submitbutton');

        $this->addElements(array($cedula, $nombre, $apellido,$id_depto,
$submit));
    }

}
```

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:

```
  public function addAction()
    {
            $form = new Application_Form_Persona();
            $form->submit->setLabel('Add');
            $this->view->form = $form;
            if ($this->getRequest()->isPost())
            {
                  $formData = $this->getRequest()->getPost();
                  if ($form->isValid($formData))
                   {
                          //$cedula = (int)($form->getValue('cedula'));
                          $nombre = $form->getValue('nombre');
                          $apellido = $form->getValue('apellido');
                          $id_depto = (int)($form->getValue('id_depto'));
                          $personas = new
                          Application_Model_DbTable_Persona();
                          $personas->
                          addPersona($nombre,$apellido,$id_depto);
                           $this->_helper->redirector('index');
                   }
                   else
                   {
                           $form->populate($formData);
                   }
            }
     }
```

Explicacion del código:

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

Instanciamos nuestro Form_Persona, 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 función isValid().

```
$nombre = $form->getValue('nombre');
$apellido = $form->getValue('apellido');
$id_depto = (int)($form->getValue('id_depto'));
$personas = new
Application_Model_DbTable_Persona();
$personas->addPersona( $nombre,$apellido,$id_depto);
```

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).

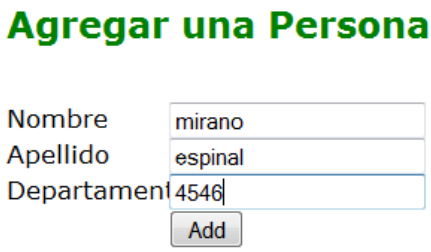
```
} else {
    $form->populate($formData);
}
```

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:

```
<?php
$this->title = "Agregar una Persona";
$this->headTitle($this->title);
echo $this->form;

?>
```

Debería aparecer la siguiente ventana y escribir en la base de datos la nueva persona:



Y luego agregar la persona:

Editar una persona:

Es algo muy parecido en lo que se hizo para agregar una persona:

Código en la function editAction del IndexController:

```
public function editAction()
    {
          $form = new Application_Form_Persona();
          $form->submit->setLabel('Save');
          $this->view->form = $form;
          if ($this->getRequest()->isPost())
          {
              $formData = $this->getRequest()->getPost();
              if ($form->isValid($formData))
              {
                  $cedula = (int)($form->getValue('cedula'));
                  $nombre = $form->getValue('nombre');
                  $apellido = $form->getValue('apellido');
                  $id_depto = (int)($form->getValue('id_depto'));
                  $personas = new Application_Model_DbTable_Persona();
                  $Personas->updatePersona($cedula,$nombre,
                  $apellido,$id_depto);
                  $this->_helper->redirector('index');
              }
              else
              {
                   $form->populate($formData);
              }
          }
```

```
        else
        {
            $cedula = $this->_getParam('cedula', 0);
            if ($cedula > 0)
            {
                $personas = new Application_Model_DbTable_Persona();
                $form->populate($personas->getPersona($cedula));
            }
        }
    }
```

Cambios:

Primeramente, cuando mostramos el formulario al usuario vamos a necesitar tomar la información de la perona (nombre,apellido,id_depto) de la base de datos y luego colocamos los elementos del formulario con la información de la persona que vamos a editar. Esto sucede al final del método:

```
            $cedula = $this->_getParam('cedula', 0);
            if ($cedula > 0)
            {
                $personas = new Application_Model_DbTable_Persona();
                $form->populate($personas->getPersona($cedula));
            }
        }
```

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 la cedula solicitada utilizando el método _getParam(). Luego utilizamos el modelo para levantar la información de la base de datos y colocamos en el formulario la información del registro de la persona a editar. El método getPersona() del modelo devuelve un array con la información de la persona.

En la vista simplemente colocaríamos el código casi igual al de la vista de adicionar:

Arhivo views/script/index/edit.phtml

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

Eliminar una persona

Para poder cerrar nuestra aplicación, vamos a necesitar agregar la función de eliminar persona. Ya tenemos un link para eliminar persona 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.

Código en la function deleteAction del IndexController:

```
   public function deleteAction()
   {
    if ($this->getRequest()->isPost()) {
        $del = $this->getRequest()->getPost('del');
        if ($del == 'Yes') {
            $cedula = $this->getRequest()->getPost('cedula');
            $personas = new Application_Model_DbTable_Persona();
            $personas->deletePersona($cedula);
        }
        $this->_helper->redirector('index');
    } else {
        $cedula = $this->_getParam('cedula', 0);
        $personas = new Application_Model_DbTable_Persona();
        $this->view->persona = $personas->getPersona($cedula);
    }
  }
```

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_Persona para realmente eliminar el registro usando el método deletePersona(). Si el request no es un POST, buscamos por el parámetro cedula y devolvemos el registro correcto y lo asignamos a la vista.

El código de la vista es un formulario es el siguiente:

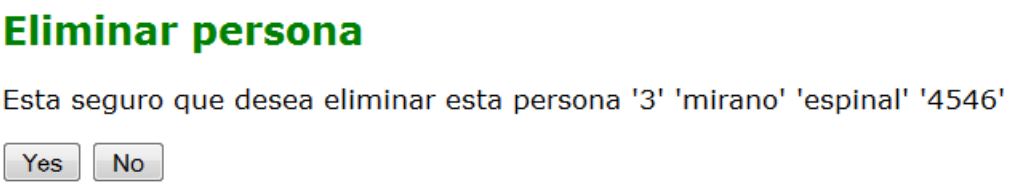
```
<?php
$this->title = "Eliminar persona";
$this->headTitle($this->title);
?>
<p>Esta seguro que desea eliminar esta persona
  '<?php echo $this->escape($this->persona['cedula']); ?>'
  '<?php echo $this->escape($this->persona['nombre']); ?>'
  '<?php echo $this->escape($this->persona['apellido']); ?>'
  '<?php echo $this->escape($this->persona['id_depto']); ?>'
</p>
<form action="<?php echo $this->url(array('action'=>'delete')); ?>"
method="post">
<div>
  <input type="hidden" name="cedula" value="<?php echo $this-
>persona['cedula']; ?>" />
  <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.

Las ventanas relacionadas serian las siguientes:

Vamos a eliminar la siguiente persona (seleccionado):

Debería aparecer una ventana para solicitar una confirmación:



Luego la persona debería eliminarse de la base de datos:

Conclusión:

El uso de este framework requiere un instalación un poco compleja, al tener muchos pasos es probable que tome un poco de tiempo realizarla satisfactoriamente.

Generar código requiere necesariamente leer la documentación, al necesitarse comandos para este propósito.