“Formularios con ZF2”

Módulo 5 - 1

AUTOR: Andrés Guzmán Fontecilla.
Email: andresguzf@gmail.com WEB: http://www.bolsadeideas.cl/servicios/cursos-de-capacitacion/

Objetivos

En éste laboratorio buscaremos profundizar el conocimiento del funcionamiento de las clases que permiten agregar más funcionalidad a nuestra interfaz de usuario, en este caso, con el componente Zend\Form.

Zend\Form es un componente que sirve principalmente como un Puente entre la lógica de negocio o dominio y la capa de presentación o vista. Se compone de objetos que representan elementos de formulario, fieldset, filtros, y métodos para poblar con datos desde el formulario hacia un objeto de negocio.

Introducción

Zend\Form es considerado por muchos uno de los componentes estrella más destacado de Zend Framework. Este componente nos simplifica la creación y manejo de formularios en nuestras aplicaciones web, agregando validación y filtros de datos, manejo y control de errores, generación (render), estilos y decoradores XHTML/CSS, entre otras posibilidades.

El objeto formulario Zend Form es un contenedor de elementos, estos a su vez son objetos que representan un elemento de un formulario HTML/XHTML, como por ejemplo un campo de texto, un área de texto, listado de selección, checkbox, etc. **En la capa del controlador debemos crear/inicializar el objeto "formulario"** y agregar los elementos que necesitemos, luego asignaremos el formulario a un atributo de la vista, y esta (la vista) se encargará de generar el formulario.

Los elementos que maneja Zend Framework son básicamente los objetos en HTML a los que estamos acostumbrados.
Un elemento puede tener validadores (subclases de Zend\Validator\AbstractValidator) que nos permiten validar la entrada, aunque ya existen muchos validadores muy útiles y de uso

Para todo desarrollador de aplicaciones web esta es una tarea cotidiana, tener que lidiar con formularios una y otra vez, trabajo monótono y tedioso, pero ahora esto es cosa del pasado, con **ZF crear formularios será simple y entretenido**, aprovechando todo el potencial de la POO al hacer uso de un diseño que contempla un completo conjunto de funcionalidades que nos ahorra tiempo y dolores de cabeza en tareas como las validaciones de datos, filtros, configuración, orden de los elementos, estética CSS y mucho más.

Creando nuestro primer formulario

Crear un objeto "formulario" es muy simple, basta con instanciar Zend\Form\Form:

```
$form = new Zend\Form\Form();
```

O bien

```
use Zend\Form\Form;
/* etc...*/
$form = new Form();
```

Luego veremos, que lo ideal y recomendado por Zend es **crear una subclase que extienda a Zend\Form y definir nuestra propia clase de formulario implementando el constructor de la clase: __construct($name = null)**, pero a veces podría bastar con usar por defecto la clase estándar de ZF e instanciarla:

```
// Modulo/src/Form/MiPropioForm.php

namespace Modulo\Form;

use Zend\Form\Element;
use Zend\Form\Form;

class MiPropioForm extends Form
{
    /* constructor usado para inicializar el form */
    public function __construct($name = null){
      //aquí creamos los elementos y lo demás.
    }
}

// en el controlador
$form = new MiPropioForm();
```

Si deseamos especificar los atributos *action* y *method* del formulario lo haremos a través del método *setAttribute()*:

```
$form-> setAttribute('action', $this->url(' admin', array('controller'=>' login',  'action' =>'index')));
$form-> setAttribute('method', 'post');
```

El ejemplo de arriba agrega al formulario el *action* hacia la URL *"/admin/login"* y el atributo *method* con valor "POST". **Cuando se genere nuestra vista (render) veremos en el código fuente HTML, generado con el componente Zend_Form, tal como si lo hubiéramos escrito nosotros a mano**

Adicionalmente podemos agregar más atributos al <form> mediante los métodos *setAttribute()* o *setAttributes()*. Por ejemplo, si necesitamos especificar el atributo id del formulario:

```
$form->setAttribute('id', 'login');
```

Lo más común es asignar el id, name, action, class, etc., lo cual como veremos es muy fácil:

```
public function __construct($name = null) {
      $this->setName('frm-ejemplo')
      ->setAttribute('action', $this->url('alguna-accion'))
      ->setAttribute('method', 'post')
      ->setAttribute('id', 'login')
      ->setAttribute('class', 'frmClass');
// cualquier atributo html se puede setear con setAttribute()
}
```

Agregar elementos nuestro formulario

Hasta el momento tenemos un formulario vacío, sin elementos. Zend\Form incorpora la mayoría de elementos de los formularios en código XHTML, siempre siguiendo los estándares de la W3C.

Nota: esta es una gran ventaja de las funciones "generadoras de código", como lo hacía en su momento Smarty (sistema de templates), ya que **nos evita tener que lidiar con errores a la hora de codificar a mano HTML**, simplemente invocamos a los componentes correctos y les pasamos parámetros, luego será un problema de Zend definir que cada uno cumpla con las reglas de la W3C.

Estos son los siguientes:

- button
- checkbox hidden
- image
- file
- password
- radio
- reset
- select submit
- text
- textarea
- captcha

Tenemos dos formas de agregar elementos al formulario:

1. podemos instanciar el elemento de forma individual y pasarlo al objeto *formulario* o bien
2. pasar un arreglo con las configuraciones del elemento y *Zend\Form* internamente se encargará de hacer el proceso por nosotros.

Algunos Ejemplos

```
use Zend\Form\Element\Text;
use Zend\Form\Form;
use Zend\Form\Factory;

/* Instanciando un element y se lo pasamos por argumento al objeto formulario: */
$username = new Text('username');
$username->setLabel('Nombre de usuario');
$form->add($username);

/* Pasamos por argumento un parámetros de tipo array, con el tipo, el nombre (name) del elemento y
una etiqueta al método add (agregar elemento) del objeto formulario: */

$form->add (array(
        'type' => 'Zend\Form\Element\Text',
        'name' => 'username',
        'options' => array(
            'label' => 'Nombre de usuario',
        ),
    ));

/* Además tenemos una tercera forma usando un Factory: */
$factory = new Factory();

$elemento  =  $factory->createElement(array(
        'type' => 'Zend\Form\Element\Text',
        'name' => 'username',
        'options' => array(
            'label' => 'Nombre de usuario',
        ),
    )
);

$form->add ($elemento);
```

Un segundo ejemplo, más elaborado:

```
namespace Usuarios\Form;

use Zend\Captcha\AdapterInterface as CaptchaAdapter;
use Zend\Form\Element;
use Zend\Form\Form;
use Zend\Captcha;
use Zend\Form\Factory;

/**
 * Description of Contacto
 *
 * @author Andres
 */
class Contacto extends Form {

    protected $captcha;

    public function setCaptcha(CaptchaAdapter $captcha) {
        $this->captcha = $captcha;
    }

    public function getCaptcha() {
        if ($this->captcha === null) {
            $this->captcha = new Captcha\Dumb();
            $this->captcha->setLabel("Por favor, ingrese la palabra al revés");
        }
        return $this->captcha;
    }

    public function __construct($name = null) {

        $this->add(array(
            'name' => 'nombre',
            'options' => array(
                'label' => 'Nombre Completo',
            ),
            'attributes' => array(
                'type' => 'text',
            ),
        ));

        $factory = new Factory();

        $email = $factory->createElement(array(
            'type' => 'Zend\Form\Element\Email',
            'name' => 'email',
            'options' => array(
                'label' => 'Correo',
            ),));
```

```
        $this->add($email);

        $radio = new Element\Radio('genero');
        $radio->setLabel('Genero');
        $radio->setValueOptions(array(
            '0' => 'Mujer',
            '1' => 'Hombre',
        ));

        $this->add($radio);

        $select = new Element\Select('area');
        $select->setLabel('Contactar a');
        $select->setEmptyOption('Seleccione un departamento =>');
        $select->setValueOptions(array(
            '0' => 'Soporte',
            '1' => 'Ventas',
            '2' => 'Ingeniería',
            '3' => 'Compras',
        ));

        $this->add($select);

        $this->add(array(
            'name' => 'tema',
            'options' => array(
                'label' => 'Tema',
            ),
            'attributes' => array(
                'type' => 'text',
            ),
        ));
        $this->add(array(
            'type' => 'Zend\Form\Element\Textarea',
            'name' => 'mensaje',
            'options' => array(
                'label' => 'Mensaje',
            ),
        ));
        $this->add(array(
            'type' => 'Zend\Form\Element\Captcha',
            'name' => 'captcha',
            'options' => array(
                'label' => 'Código de verificación',
                'captcha' => $this->getCaptcha(),
            ),
        ));
        $this->add(new Element\Csrf('security'));
        $this->add(array(
            'name' => 'send',
            'attributes' => array(
                'type' => 'submit',
                'value' => 'Enviar',
```

```
            ),
        ));
    }

}
```

Asignar propiedades al elemento

Asigna el nombre de la etiqueta del elemento:

```
$elemento->setLabel('nombre_etiqueta');

// si es de la forma de arreglo usamos options y labels

etc..
'options' => array(
'label' => 'nombre_etiqueta',
),
```

Asigna el valor del elemento (sobrescribe la entrada del usuario):

```
        $elemento->setValue('valor');
```

Pasar el formulario a la vista para que sea generado (render):

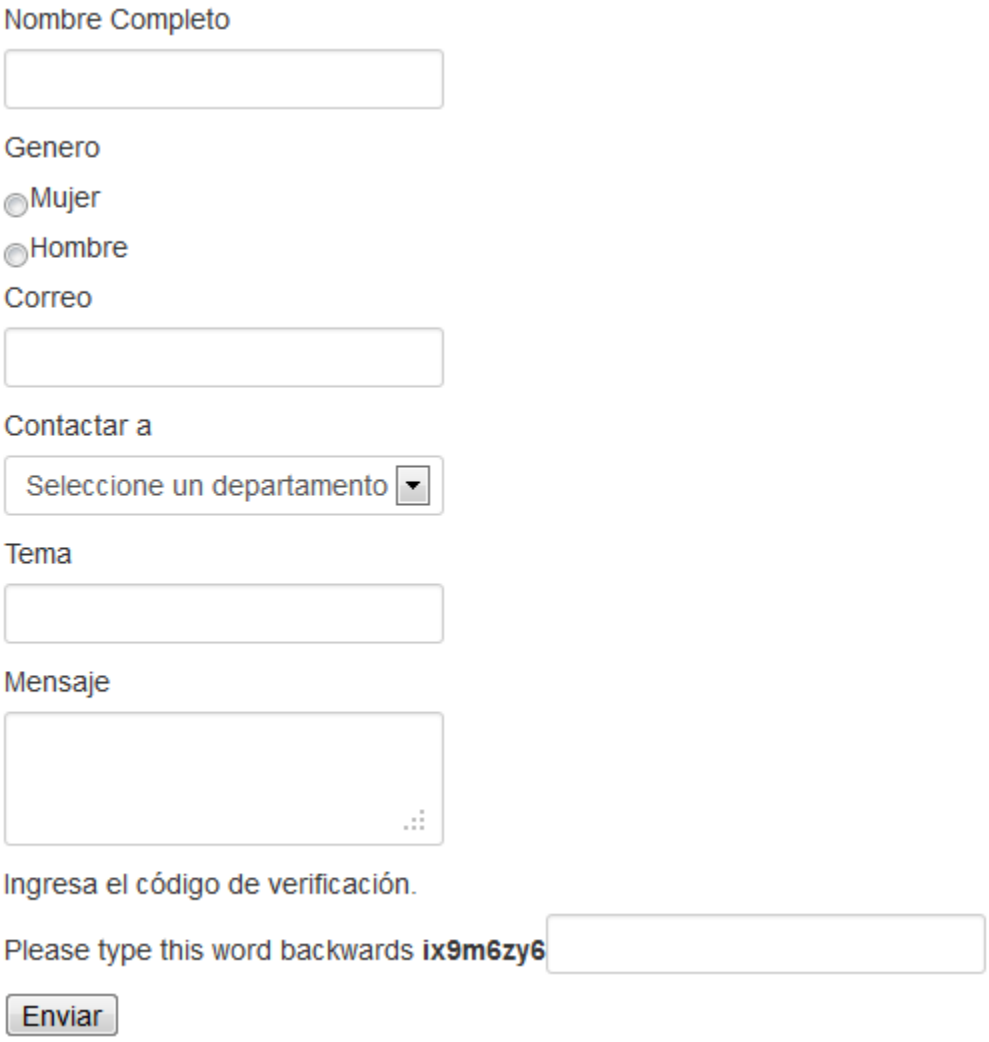
Desde alguna acción en el Controlador:

```
    public function fooAction()
    {
        $form = new Contacto("contacto");
        return array('form' => $form);
    }

    public function barAction()
    {
        //En esta acción procesamos el formulario, validaciones, etc…
        $form = new Contacto("contacto");
        $data = $this->request->getPost();
        $form->setData($data);

             // Validamos el Form
        if ($form->isValid()) {
            // Hacer algo si es valido
        } else {
            // Manejamos los errores y retornamos al form
        }
    }
```

Finalmente, cómo se genera el formulario en la vista:

```
<?php
// Dentro de un view script
$form = $this->form;
$form->prepare();

// Asumiendo que existe la ruta "usuarios/index/foo" ...
$form->setAttribute('action', $this->url('usuarios', array('controller'=>'index',
'action' =>'bar')));

$form->setAttribute('method', 'post');

// Get the form label plugin
$formLabel = $this->plugin('formLabel');

// Imprime la etiqueta HTML del for: <form>
echo $this->form()->openTag($form);
?>
<div class="form_element">
    <?php
    $name = $form->get('nombre');
    echo $formLabel->openTag() . $name->getOption('label');
    echo $formLabel->closeTag();
    echo $this->formInput($name);
    echo $this->formElementErrors($name);
    ?></div>

<div class="form_element">
    <?php
    $genero = $form->get('genero');
    echo $formLabel->openTag() . $genero->getLabel();
    echo $formLabel->closeTag();
    echo $this->formRadio($genero);
    echo $this->formElementErrors($genero);
    ?></div>

<div class="form_element">
    <?php
    $email = $form->get('email');
    echo $formLabel->openTag() . $email->getOption('label');
    echo $formLabel->closeTag();
    echo $this->formInput($email);
    echo $this->formElementErrors($email);
    ?></div>


<div class="form_element">
    <?php
    $area = $form->get('area');
    echo $formLabel->openTag() . $area->getLabel();
    echo $formLabel->closeTag();
    echo $this->formSelect($area);
```

```
    echo $this->formElementErrors($area);
    ?></div>

<div class="form_element">
    <?php
    $subject = $form->get('tema');
    echo $formLabel->openTag() . $subject->getOption('label');
    echo $formLabel->closeTag();
    echo $this->formInput($subject);
    echo $this->formElementErrors($subject);
    ?></div>

<div class="form_element">
    <?php
    $message = $form->get('mensaje');
    echo $formLabel->openTag() . $message->getOption('label');
    echo $formLabel->closeTag();
    echo $this->formTextarea($message);
    echo $this->formElementErrors($message);
    ?></div>

<div class="form_element">
    <?php
    $captcha = $form->get('captcha');
    echo $formLabel->openTag() . $captcha->getOption('label');
    echo $formLabel->closeTag();
    echo $this->formCaptcha($captcha);
    echo $this->formElementErrors($captcha);
    ?></div>

<?php echo $this->formElement($form->get('security')) ?>
<?php echo $this->formElement($form->get('send')) ?>

<?php echo $this->form()->closeTag() ?>
```



Antes de empezar con los validadores de ZF2, es necesario tener configurado y activado el Locale nativo de PHP 5.3.

Por lo que hay que activar la extensión ***php_intl.dll*** en **php.ini**, eliminando el comentario (;), entonces cambiamos o modificamos:

```
;extension=php_intl.dll
```

Por:

```
extension=php_intl.dll
```

Y volvemos a reiniciar apache!

Validadores

Por defecto los elementos no incluyen ningún tipo de validador ni de filtro. Esto significa que necesitamos configurar nuestros propios validadores y filtros por cada elemento. Podemos hacer esto de diferentes formas:

1. Heredar de la clase InputFilter e implementar el constructor para las reglas de filtros y validaciones de cada campo.
2. Implementar la interfaz InputFilterAwareInterface e implementar el método getInputFilter() para que retorne el objeto InputFilter con las reglas de validaciones.
3. Directamente instanciar el objeto InputFilter e Input en la clase Form y crear las validaciones.

En el ejemplo veremos cómo crear el objeto InputFilter e InputFactory para declarar e implementar las reglas de validaciones de nuestros campos:

```
use Zend\InputFilter\Factory as InputFactory;
use Zend\InputFilter\InputFilter;

... ETC...

$inputFilter = new InputFilter();
$factory     = new InputFactory();

$inputFilter->add($factory->createInput(array(
    'name'     => 'id',
    'required' => true,
    'filters'  => array(
        array('name' => 'Int'),
    ),
)));

$inputFilter->add($factory->createInput(array(
    'name'     => 'nombre',
    'required' => true,
    'filters'  => array(
        array('name' => 'StripTags'),
        array('name' => 'StringTrim'),
    ),
    'validators' => array(
        array('name'    => 'StringLength',
              'options' => array(
```

```
                    'encoding' => 'UTF-8',
                    'min'      => 4,
                    'max'      => 14,
                ),
            ),
        ),
));
```

El objeto InputFactory nos crea cada Input con sus reglas de filtros y validaciones y las agregamos al InputFilter. En el ejemplo observamos cómo declaramos el filtro **Int** para el campo id, permite convertir el valor ingresado a un tipo **integer** de PHP. En el campo nombre tenemos dos filtros, el primero para remover caracteres HTML y PHP del string, el segundo para remover los espacios en blanco. Luego tenemos un validador para el campo nombre para un mínimo de 4 y máximo de 14 caracteres.

Una vez creado el objeto InputFilter se lo pasamos al objeto form usando el método setInputFilter(…), suponiendo que estamos dentro de la misma clase Form :

```
$this->setInputFilter($inputFilter);
```

O bien fuera de la clase form, ejemplo en el controlador usando la instancia del form:

```
$form = new MiForm();
$form->setInputFilter($inputFilter);
```

Veamos otra forma de implementar la validación, creando nuestra propia clase validadora (InputFilter) que hereda de la clase InputFilter, es la forma más típica y recomendada:

```
<?php
namespace Usuarios\Form;

use Zend\Validator\StringLength;
use Zend\InputFilter\InputFilter;
/**
 * Description of ContactoValidator
 *
 * @author Andres
 */
class ContactoValidator extends InputFilter {

    public function __construct() {

        $this->add(
                array(
                    'name' => 'nombre',
                    'required' => true,
                    'filters' => array(
                        array('name' => 'StripTags'),
                        array('name' => 'StringTrim'),
                    ),
                    'validators' => array(
                        array(
                            'name' => 'StringLength',
                            'options' => array(
                                'min' => 4,
                                'max' => 14,
                                'messages' => array(
            StringLength::TOO_SHORT => 'El nombre debe tener minimo 4 caracteres',
            StringLength::TOO_LONG => 'El nombre debe tener maximo 12 caracteres',
                                )
                            ),
                        ),
                        array(
                            'name' => 'Alnum',
                            'options' => array(
                                'allowWhiteSpace' => true,
                            ),
                        ),
                    ),
                )
        );

        /* ETC... por cada campo del formulario */
    }
}
```

Esta es la forma más típica y recomendada por ZF2 ya que se crea una clase separada y reutilizable, completamente desacoplada del formulario, que contiene todas las reglas de validación, por lo general es una clase InputFilter por Formulario.

Otra alternativa, aunque es prácticamente lo mismo sólo que de una manera más programática:

```
<?php

namespace Usuarios\Form;

use Zend\Validator\StringLength;
use Zend\Validator\NotEmpty;
use Zend\InputFilter\InputFilter;
use Zend\InputFilter\Input;
use Zend\I18n\Validator\Alnum;
/**
 * Description of ContactoValidator
 *
 * @author Andres
 */
class ContactoValidator extends InputFilter {

    public function __construct() {

        $nombre = new Input('nombre');
        $nombre->setRequired(true);
        $nombre->getFilterChain()
                ->attachByName('StripTags')
                ->attachByName('StringTrim');

        $nombre->getValidatorChain()
                ->addValidator(new StringLength(array('min' => 4, 'max' => 14,
                        'messages' => array(
           StringLength::TOO_SHORT => "El nombre debe tener mínimo 4 caracteres",
           StringLength::TOO_LONG => "El nombre debe tener máximo 14 caracteres"))))
                ->addValidator(new Alnum(array('allowWhiteSpace' => true)));

        $this->add($nombre);

        /* ETC... por cada campo del formulario */
    }

}
```

Luego la usamos en alguna acción del controlador (la que procesa el formulario y recibe los datos), asignándola al objeto form:

```
use Usuarios\Form\Contacto;
use Usuarios\Form\ContactoValidator;

... ETC...

    public function barAction() {

        $form = new Contacto("contacto");
        $form->setInputFilter(new ContactoValidator());

        $data = $this->request->getPost();  // for POST data
        $form->setData($data);

        if ($form->isValid()) {
        } else {
        }
    }
```

Validador Alnum o Alpha y espacios en blanco

En campos, como por ejemplo **nombres, descripción o texto en general** (alfanumérico), podrían contener espacios en blanco entre palabras, entonces usando validadores **Alnum** o **Alpha**, ya que ambos permiten espacios, dentro del InputFilter quedaría:

```
'validators' => array(
    array(
        'name' => 'Alnum',
        'options' => array(
            'allowWhiteSpace' => true,
        ),
    ),
),
```

Para que funcionen bien los validadores Alnum y Alpha es necesario tener activada la extensión **php_intl** en el archivo de configuración de PHP **php.ini**:

```
extension=php_intl.dll
```

Requeridos

También podemos especificar un elemento Input como "requerido" ("obligatorio") dentro del InputFilter, de la siguiente forma:

```
        $this->add(
                array(
                    'name' => 'nombre',
                    'required' => true,
                    'filters' => array(
                        array('name' => 'StripTags'),
                        array('name' => 'StringTrim'),
                    ),
```

O bien de la forma programática:

```
        use Zend\InputFilter\Input;

        $nombre = new Input('nombre');
        $nombre->setRequired(true);
```

Cuando un elemento es "obligatorio", internamente Zend Framework agrega un validador 'NotEmpty' al comienzo de la cadena de validadores, asegurándose de esta forma que el elemento siempre tenga un valor.

Opciones de Validación

Los validadores más importantes son (el nombre de validador es el que está entre paréntesis):

- **Zend\I18n\Validator\Alnum (alnum)**
  Nos permite verificar que la entrada solo contenga caracteres alfanuméricos (letras y dígitos), opcionalmente se le puede pasar como opción un booleano para definir si se deben aceptar espacios en blanco (predeterminado en falso)

- **Zend\I18n\Validator\Alpha (alpha)**
  Nos permite verificar que la entrada sea sólo letras, al igual que en el anterior, tiene una opción que permite aceptar espacios en blanco

- **Zend\I18n\Validator\Int (int)**
  Permite verificar que la entrada sea sólo números enteros

- **Zend\I18n\Validator\Float (float)**
  Permite verificar que la entrada sea sólo decimales

- **Zend\Validator\Digits (digits)**
  Permite verificar que la entrada sea sólo números

- **Zend\Validator\StringLength (stringLength)**
  Nos permite definir que la entrada sólo contenga un número mínimo o máximo de valores.

- **Zend\Validator\EmailAddress (emailAddress)**
  Es un validador muy poderoso que nos permite verificar que un correo sea válido (al menos en cuanto a formato, pero también incluye que el servidor sea válido)

- **Zend\Validator\NotEmpty (notEmpty)**
  Este validador nos permite verificar que el usuario ha introducido un valor, pero lamentablemente no permite eliminar espacios en blanco (trim).

Los Filtros

Básicamente son registrados de la misma manera que los validadores (validators), dentro de las reglas del InputFilter. Por ejemplo, si queremos agregar un filtro que agrega "lowercase" (minúsculas) al valor del elemento:

```
'filters' => array(
   array('name' => 'StringToLower'),
   array('name' => 'StripTags'),
   array('name' => 'StringTrim'),
),
```

Validando el Formulario

Después de haber "renderizado" y enviado un formulario (haciendo "submit"), necesitamos comprobar y ver si los datos enviados por el usuario han pasado correctamente la validación.

Para esto debemos:

1. crear nuevamente el formulario y verificar si se han enviado datos (porque ZF solo envía valores y no información sobre el formulario),
2. si se han enviado datos, entonces podemos utilizar la función isValid($_POST); que regresará un valor de verdadero en caso de que todos los validadores fueron satisfechos o falso en caso de que alguno no lo fuera.

Cada elemento es verificado contra el valor suministrado; si el identificador (key) del nombre del elemento no está presente, y el ítem es marcado como "requerido", las validaciones se ejecutan con un valor "nulo" (null).

Los datos puede provenir desde $_POST o $_GET, o cualquier otra fuente de datos que podríamos tener a mano (peticiones de servicio web, etc.):

En el siguiente ejemplo obtenemos los parámetros del formulario, muy similar a obtenerlos mediante la variable de entorno $_POST.

```
$data = $this->request->getPost();
if($form->isValid($data)) {
    // success!
} else {
    // failure!
}
```

Asumiendo que nuestra validación fue pasada con éxito, podemos obtener los valores que fueron filtrados:

```
$values = $form->getData();
```

Individualmente por cada elemento del formulario:

```
$values = $form->getData();
$nombre = $values['nombre'];
$apellido = $values['apellido'];
```

Como vemos en el ejemplo, podemos obtener los valores del formulario con getData().

Contrariamente, si necesitamos obtener los valores no filtrados, en cualquier momento podemos recurrir a lo siguiente:

```
$filter = $form->getInputFilter();

$rawValues    = $filter->getRawValues();
$nameRawValue = $filter->getRawValue('nombre');
```

¿Qué sucede si falla la validación?

En la mayoría de los casos podemos simplemente imprimir nuevamente el formulario y los errores serán mostrados:

```
if (!$form->isValid($formData)) {
        //Retornamos e imprimimos el formulario en pantalla con los mensajes   de errrores
        return array('form' => $form);
}
```

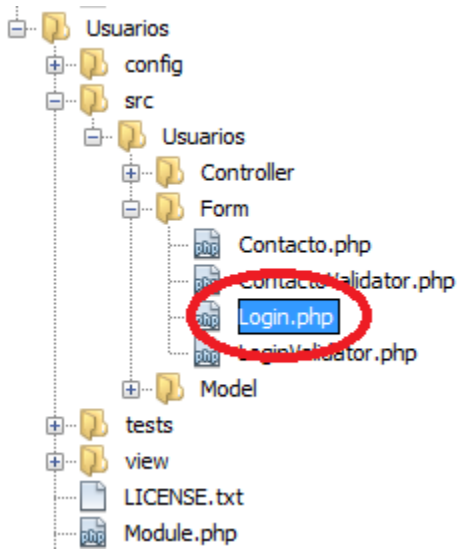
Y si la vista del formulario se encuentra en otra acción:

```
if (!$form->isValid($formData)) {
        //Retornamos e imprimimos el formulario en pantalla con los mensajes   de errrores
        $modelView = new ViewModel(array('form' => $form));
        $modelView->setTemplate('usuarios/index/foo');
        return $modelView;
}
```

La ventaja de ZF es que automáticamente agregará un mensaje de error al renderizar el elemento que lo generó.

Un ejemplo de formulario de Login

Revisamos la estructura de directorios de nuestro proyecto, debemos tener creado una carpeta **Form** dentro del módulo **Usuarios** en **src**:



Dentro del directorio **Form** creamos la clase de formulario **Login**, que extiende de **Zend\Form\Form**, impecable:

```
<?php
namespace Usuarios\Form;

use Zend\Form\Form;

class Login extends Form
{

    public function __construct($name = null)
    {
        /* Definición de los elementos del Form aquí */
    }

}
```

Ahora implementamos el constructor que es donde se debe escribir el código relacionado al form y sus elementos, es decir implementamos el formulario.

Para poner todo lo anterior en orden, vamos a implementar el __construct() del form. Este necesitará de los siguientes elementos:

1. email
2. password
3. submit

Para nuestro propósito vamos a asumir que un email válido con su respectivo validador EmailAddress. Para el caso del password debe tener un mínimo de 6 caracteres y alfanumérico.

Nuestra clase formulario Login tendrá la siguiente forma:

```
<?php
namespace Usuarios\Form;
use Zend\Form\Form;

/**
 * Description of Contacto
 *
 * @author Andres
 */
class Login extends Form {

    public function __construct($name = null) {
        parent::__construct($name);

        $this->add(array(
            'type' => 'Zend\Form\Element\Email',
            'name' => 'email',
            'options' => array(
                'label' => 'Correo',
            ),
        ));

        $this->add(array(
            'type' => 'Zend\Form\Element\Password',
            'name' => 'clave',
            'options' => array(
                'label' => 'Clave',
            ),
        ));

        $this->add(array(
            'name' => 'send',
            'attributes' => array(
                'type' => 'submit',
                'value' => 'Enviar',
            ),
        ));
    }

}
```

Nuevamente es importante notar que al igual que los modelos, se encuentran dentro del namespace del módulo, en el ejemplo **Usuarios**

Seguido del sub-namespace Form: **namespace Usuarios\Form;**

Usando el formulario en el controlador

El controlador que usa el formulario, en nuestro caso LoginController del módulo **Usuarios**, quedaría más o menos de la siguiente forma:

```
<?php

namespace Usuarios\Controller;

use Zend\Mvc\Controller\AbstractActionController;
use Zend\View\Model\ViewModel;
use Usuarios\Form\Login;
use Usuarios\Form\LoginValidator;

use Usuarios\Model\Dao\IUsuarioDao;

class LoginController extends AbstractActionController {

    private $usuarioDao;

    public function setUsuarioDao(IUsuarioDao $usuarioDao) {
        $this->usuarioDao = $usuarioDao;
    }

    public function indexAction() {
        $form = new Login("login");
        return array('form' => $form);
    }

    public function autenticarAction() {

        if (!$this->request->isPost()) {
            $this->redirect()->toRoute('usuarios', array('controller' => 'login'));
        }

        $form = new Login("login");

        $form->setInputFilter(new LoginValidator());

        // Obtenemos los datos desde el Formulario con POST data:
        $data = $this->request->getPost();


        $form->setData($data);

        // Validando el form
        if (!$form->isValid()) {

            $modelView = new ViewModel(array('form' => $form));
            $modelView->setTemplate('usuarios/login/index');
            return $modelView;
        }
```

```
        $values = $form->getData();

        $email = $values['email'];
        $clave = $values['clave'];

        $cuentaUsuario = $this->usuarioDao->obtenerCuenta($email, $clave);

        if ($cuentaUsuario !== null) {
            $this->layout()->mensaje = "Login Correcto!!!";
            $this->layout()->colorMensaje = "green";

            return $this->forward()->dispatch('Usuarios\Controller\Login',
array('action' => 'success'));
        } else {
            $this->layout()->mensaje = "Login Incorrecto";
            return $this->forward()->dispatch('Usuarios\Controller\Login',
array('action' => 'index'));
        }
    }

    public function successAction() {
        return array('titulo' => 'Página de exito');
    }

}
```

Cuando hacemos un **return** dentro de una acción es para salirnos del método, en el ejemplo anterior si la validación falla retornamos y se generará la vista que hemos llamado con el objeto ViewModel, todo lo que esté más abajo del return no se ejecutará.

Usando el ServiceManager para inyectar el UsuarioDao dentro del Controller

El objeto UsuarioDao lo inyectamos en la clase Module, implementando el método getControllerConfig() que nos permite crear con factories el controlador y de esta forma inyectar el servicio:

```
public function getServiceConfig() {
    return array(
        'factories' => array(
            'UsuarioDao' => function($sm) {
                return new UsuarioDao();
            },
        ),
    );
}

public function getControllerConfig() {
    return array(
        'factories' => array(
            'Usuarios\Controller\Login' => function ($sm) {
                $controller = new \Usuarios\Controller\LoginController();
                $locator = $sm->getServiceLocator();
                $controller->setUsuarioDao($locator->get('UsuarioDao'));
                return $controller;
            }
        )
    );
}
```

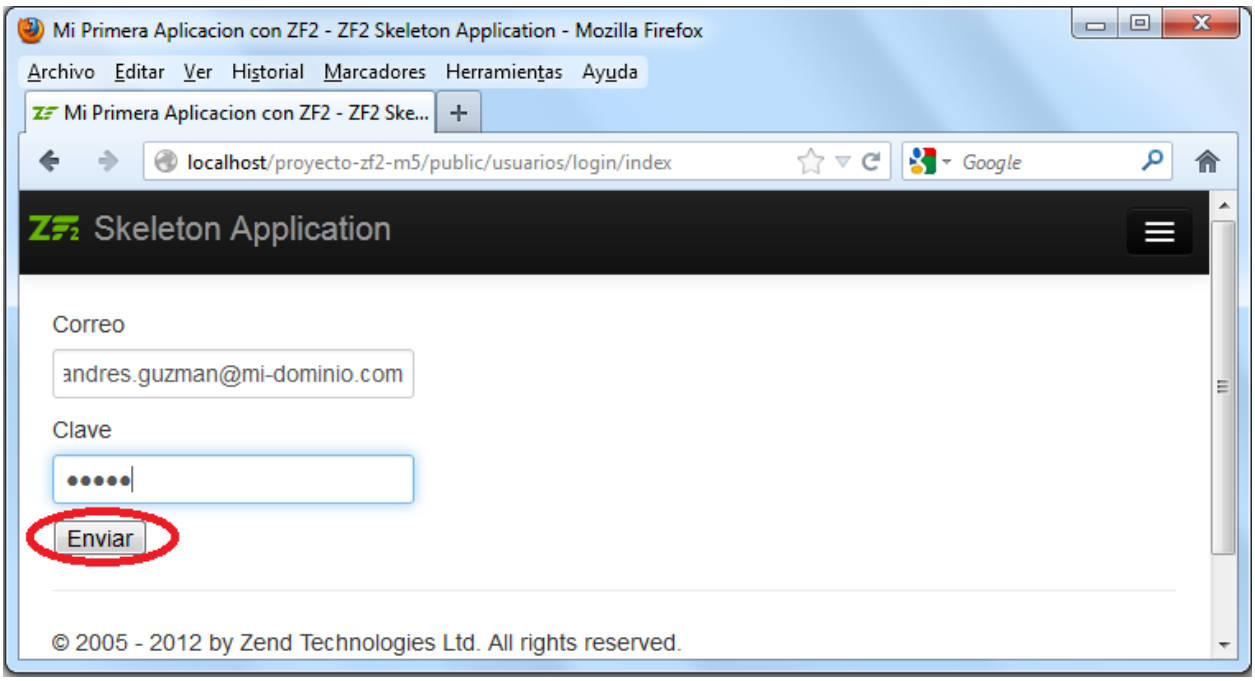
Las reglas de validación y filtros la implementamos en la clase que heredamos de InputFilter:

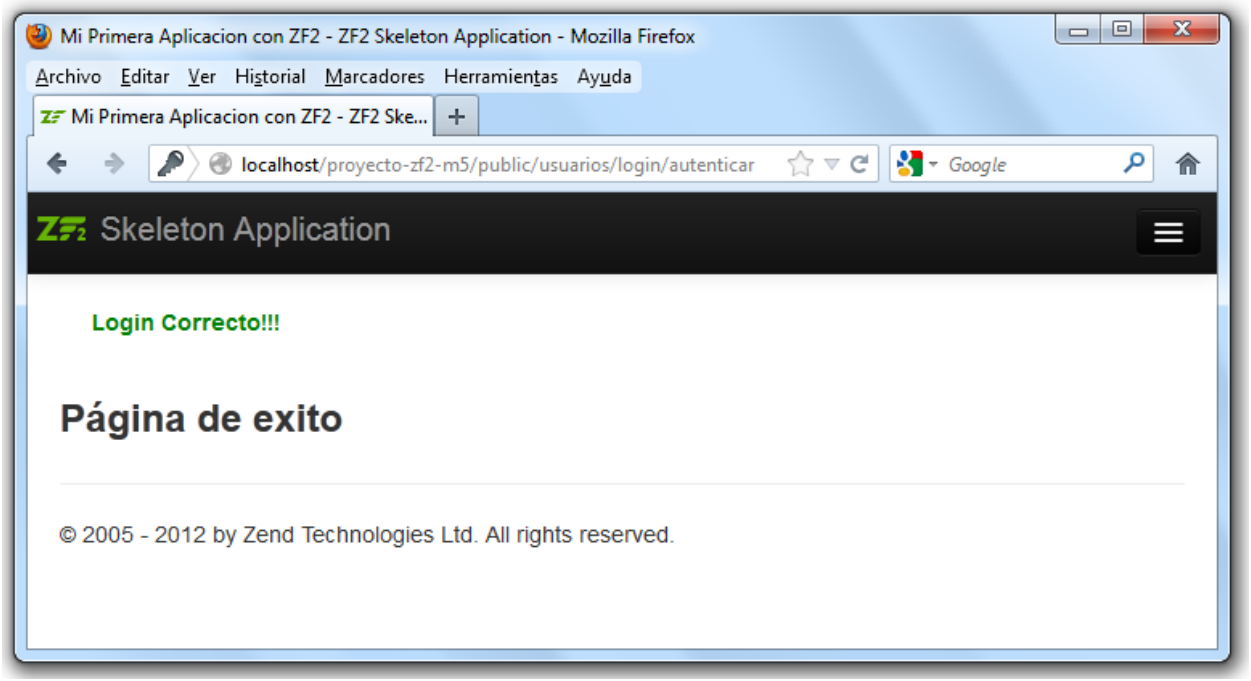
```
<?php

namespace Usuarios\Form;

use Zend\Validator\StringLength;
use Zend\InputFilter\InputFilter;

/**
 * Description of ContactoValidator
 *
 * @author Andres
 */
class LoginValidator extends InputFilter {

    public function __construct() {

        $this->add(
                array(
                    'name' => 'email',
                    'required' => true,
                    'validators' => array(
                        array(
                            'name' => 'EmailAddress',
                        ),
                    ),
                )
        );

        $this->add(
                array(
                    'name' => 'clave',
                    'required' => true,
                    'filters' => array(
                        array('name' => 'StripTags'),
                        array('name' => 'StringTrim'),
                    ),
                    'validators' => array(
                        array(
                            'name' => 'StringLength',
                            'options' => array(
                                'min' => 4,
                                'max' => 8,
                                'messages' => array(
            StringLength::TOO_SHORT => 'La clave debe tener minimo 4 caracteres.',
            StringLength::TOO_LONG => 'La clave debe tener maximo 8 caracteres.',
                                )
                            ),
                        ),
```

```
                        array(
                            'name' => 'Alnum',
                        ),
                    ),
                )
            );
        }

}
```

Y por último la vista (view script) para mostrar e imprimir el formulario de login:

```
<?php
// within a view script
$form = $this->form;
$form->prepare();

$form->setAttribute('action', $this->url('usuarios', array('controller'=>'login',
'action' =>'autenticar')));

$form->setAttribute('method', 'post');

// Get the form label plugin
$formLabel = $this->plugin('formLabel');

// Render the opening tag
echo $this->form()->openTag($form);
?>
<div class="form_element">
    <?php
    $email = $form->get('email');
    echo $formLabel->openTag() . $email->getOption('label');
    echo $formLabel->closeTag();
    echo $this->formInput($email);
    echo $this->formElementErrors($email);
    ?></div>

<div class="form_element">
    <?php
    $password = $form->get('clave');
    echo $formLabel->openTag() . $password->getOption('label');
    echo $formLabel->closeTag();
    echo $this->formInput($password);
    echo $this->formElementErrors($password);
    ?></div>

<?php echo $this->formElement($form->get('send')) ?>
<?php echo $this->form()->closeTag() ?>
```


Mi Primera Aplicacion con ZF2 - ZF2 Skeleton Application - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
Mi Primera Aplicacion con ZF2 - ZF2 Ske...
localhost/proyecto-zf2-m5/public/usuarios/login/index
Google
Skeleton Application
Correo
andres.guzman@mi-dominio.com
Clave
Enviar
© 2005 - 2012 by Zend Technologies Ltd. All rights reserved.


Mi Primera Aplicacion con ZF2 - ZF2 Skeleton Application - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
Mi Primera Aplicacion con ZF2 - ZF2 Ske...
localhost/proyecto-zf2-m5/public/usuarios/login/autenticar
Google
Skeleton Application
Login Correcto!!!
Página de exito
© 2005 - 2012 by Zend Technologies Ltd. All rights reserved.

Resumen

En este capítulo vimos cómo funciona el componente Zend\Form, cómo se puede generar un formulario XHTML, la manera que se agregan los elementos de todo formulario y de qué forma se pueden agregar validadores y filtros.

La próxima tarea se concentrará en aplicar todos estos conceptos.

FIN.