ZF - Session 6

1. Forms

Hasta ahora hemos venido implementando nuestros formularios de manera manual ya que nos interesaba centrarnos más en otros aspectos de ZF2. Entre otras cosas hemos omitido la validación de los datos introducidos por el usuario.

Zend\Form es un componente que nos permite reutilizar formularios en nuestro código. Con Zend\Form tenemos la posibilidad de usar nuestros formularios en diferentes puntos de la aplicación:

- En las vistas mediante el uso de view helpers específicos para formularios
- En el módelo a la hora de filtrar y validad los datos a través de InputFilters
- En los controllers para parametrizar formularios y rellenar sus campos a partir de entidades

Utilizaremos el módulo User para ilustrar su uso. Reimplementaremos los formularios del CRUD de usuarios.

Los pasos a seguir serán:

1. Crear la clase UserForm
2. Crear el InputFilter dentro de la entidad User
3. Instanciar el formulario en el action create y pasarlo a la vista
4. Mostrar el formulario en la vista mediante el uso de view helpers
5. Utilizar el formulario para validar los datos introducidos por el usuario

2. Creando el formulario

```php
<?php
/**
 * xenFramework (http://xenframework.com/)
 *
 * This file is part of the xenframework package.
 *
 * (c) Ismael Trascastro <itrascastro@xenframework.com>
 *
 * @link        http://github.com/xenframework for the canonical source repository
 * @copyright   Copyright (c) xenFramework. (http://xenframework.com)
 * @license     MIT License - http://en.wikipedia.org/wiki/MIT_License
 *
 * For the full copyright and license information, please view the LICENSE
 * file that was distributed with this source code.
 */

namespace User\Form;
```

```php
use Zend\Form\Form;

class User extends Form
{
    function __construct($name = null)
    {
        parent::__construct();

        $this->setName('User');
        $this->setAttribute('method', 'post');

        $this->add(array(
            'name'  => 'id',
            'type'  => 'Hidden',
        ));

        $this->add(array(
            'name'  => 'date',
            'type'  => 'Hidden',
        ));

        $this->add(array(
            'name'       => 'email',
            'type'       => 'Email',
            'attributes' => array(
                'required' => 'required',
            ),
        ));

        $this->add(array(
            'name'  => 'password',
            'type'  => 'Text',
            'attributes' => array(
                'required' => 'required',
            ),
        ));

        $this->add(array(
            'name'  => 'role',
            'type'  => 'Text',
            'attributes' => array(
                'required' => 'required',
            ),
        ));

        $this->add(array(
            'name' => 'submit',
            'type' => 'Submit',
            'attributes' => array(
                'id' => 'submitbutton',
```

```
            ),
        ));

    }
}
```

Se hereda de la clase Form y se van añadiendo los controles. Para ver de qué controles disponemos y su uso consultamos la documentación oficial:

http://framework.zend.com/manual/current/en/modules/zend.form.elements.html

3. Filtrado y validación

Lo siguiente que haremos será filtrar y validar los datos. Lo haremos en la entidad User ya que es quien mejor conoce la naturaleza de los datos:

```
<?php
/**
 * xenFramework (http://xenframework.com/)
 *
 * This file is part of the xenframework package.
 *
 * (c) Ismael Trascastro <itrascastro@xenframework.com>
 *
 * @link        http://github.com/xenframework for the canonical source repository
 * @copyright   Copyright (c) xenFramework. (http://xenframework.com)
 * @license     MIT License - http://en.wikipedia.org/wiki/MIT_License
 *
 * For the full copyright and license information, please view the LICENSE
 * file that was distributed with this source code.
 */

namespace User\Model;


use Zend\InputFilter\InputFilter;
use Zend\InputFilter\InputFilterAwareInterface;
use Zend\InputFilter\InputFilterInterface;
use Zend\Validator\InArray;

class User implements InputFilterAwareInterface
{
    // for binding to work with the form this variables have the same name as the form fields
    // not use here '_' to denote private property

    private $id;
```

```
    private $email;
    private $password;
    private $role;
    private $date;

    /**
     * @var InputFilterInterface
     *
     * This variable is needed for the input filter
     */
    private $inputFilter;

    function __construct($id = null, $email = null, $password = null, $role = null, $date
= null)
    {
        $this->id          = $id;
        $this->email       = $email;
        $this->password    = $password;
        $this->role        = $role;
        $this->date        = $date;
    }

    /**
     * exchangeArray
     *
     * This method is required to work with TableGateWay
     *
     * @param $data
     */
    public function exchangeArray($data)
    {
        $this->id          = (!empty($data['id'])) ? $data['id'] : null;
        $this->email       = (!empty($data['email'])) ? $data['email'] : null;
        $this->password    = (!empty($data['password'])) ? $data['password'] : null;
        $this->role        = (!empty($data['role'])) ? $data['role'] : null;
        $this->date        = (!empty($data['date'])) ? $data['date'] : null;
    }

    /**
     * getArrayCopy
     *
     * Needed for use in form binding
     *
     * @return array
     */
    public function getArrayCopy()
    {
        return get_object_vars($this);
    }
```

```php
/**
 * @return null
 */
public function getId()
{
    return $this->id;
}

/**
 * @param null $id
 */
public function setId($id)
{
    $this->id = $id;
}

/**
 * @return null
 */
public function getEmail()
{
    return $this->email;
}

/**
 * @param null $email
 */
public function setEmail($email)
{
    $this->email = $email;
}

/**
 * @return null
 */
public function getPassword()
{
    return $this->password;
}

/**
 * @param null $password
 */
public function setPassword($password)
{
    $this->password = $password;
}

/**
 * @return null
```

```
     */
    public function getRole()
    {
        return $this->role;
    }

    /**
     * @param null $role
     */
    public function setRole($role)
    {
        $this->role = $role;
    }

    /**
     * @return null
     */
    public function getDate()
    {
        return $this->date;
    }

    /**
     * @param null $date
     */
    public function setDate($date)
    {
        $this->date = $date;
    }

    /**
     * Set input filter
     *
     * @param  InputFilterInterface $inputFilter
     *
     * @return InputFilterAwareInterface
     */
    public function setInputFilter(InputFilterInterface $inputFilter)
    {
        throw new \Exception('Not used');
    }

    /**
     * Retrieve input filter
     *
     * @return InputFilterInterface
     */
    public function getInputFilter()
    {
        if (!$this->inputFilter) {
```

```
        $inputFilter = new InputFilter();

        $inputFilter->add(array(
            'name' => 'id',
            'continue_if_empty' => true,
        ));

        $inputFilter->add(array(
            'name' => 'email',
            'required' => true,
            'filters' => array(
                array('name' => 'StringTrim'), // clean blank spaces
                array('name' => 'StripTags'), // clean malicious code
                array('name' => 'StringToLower'),
            ),
            'validators' => array(
                array(
                    'name' => 'EmailAddress',
                    'options' => array(
                        'messages' => array(
                            'emailAddressInvalidFormat' => 'You entered an invalid email
address',
                        ),
                    ),
                ),
                array(
                    'name' => 'NotEmpty',
                    'options' => array(
                        'messages' => array(
                            'isEmpty' => 'Email address is required',
                        ),
                    ),
                ),
            ),
        ));

        $inputFilter->add(array(
            'name' => 'password',
            'required' => true,
            'filters' => array(
                array('name' => 'Alnum'),
            ),
        ));

        $inputFilter->add(array(
            'name' => 'role',
            'required' => true,
            'filters' => array(
                array('name' => 'Alpha'), // only letters
            ),
```

```
            'validators' => array(
                array(
                    'name'    => 'InArray',
                    'options' => array(
                        'haystack' => array('user', 'admin'),
                        'strict'   => InArray::COMPARE_STRICT
                    ),
                ),
            ),
        ));

        $inputFilter->add(array(
            'name' => 'date',
            'continue_if_empty' => true,
        ));

        $this->inputFilter = $inputFilter;
    }

    return $this->inputFilter;
  }

}
```

Los cambios realizados en la entidad User son la implementación de la interfaz InputFilterAwareInterface (nos obliga a implementar los métodos getter y setter para la propiedad InputFilter) y la creación del método getArrayCopy (utilizado para poder rellenar un formulario a partir de una entidad User).

El método getInputFilter es el utilizado para crear los filtros y validadores del formulario.

3.1. Filtros

Los filtros dejan los valores de los campos tal y como los necesitamos (sin espacios en blanco, en minúsculas, sin código malicioso, ...).

http://framework.zend.com/manual/current/en/modules/zend.filter.set.html

3.2. Validadores

Los validadores son preguntas que hacemos sobre los campos (¿es un email?, ¿es un valor incluido en una colección?, ...)

Los validadores pueden tener mensajes de error que luego se mostrarán en las vistas.

http://framework.zend.com/manual/current/en/modules/zend.validator.set.html

4. Instanciación y personalización en un controller

```php
public function createAction()
{
    $this->layout()->title = 'Create User';

    $form = new UserForm();
    $form->get('submit')->setValue('Create New User');
    $form->setAttribute('action', $this->url()->fromRoute('user\account\doCreate'));

    return ['form' => $form, 'isUpdate' => false];
}
```

Donde personalizamos el formulario con un valor concreto para el botón submit y el action al cual enviaremos los datos.

5. Form View Helpers

```php
<div>
    <?php echo $this->form()->openTag($form); ?>
        <?php if ($isUpdate): ?>
            <!-- Disabled fields will not be sent in $_POST so we have to create hidden fields -->
            <?php echo $this->formHidden($form->get('id')); ?>
            <?php echo $this->formHidden($form->get('date')); ?>
        <?php endif; ?>
        <table>
            <tr>
                <td colspan="2"><?php echo $this->formElementErrors($form->get('email')); ?></td>
            </tr>
            <tr>
                <td><strong>Email:</strong></td>
                <td><?php echo $this->formElement($form->get('email')) ?></td>
            </tr>
            <tr>
                <td colspan="2"><?php echo $this->formElementErrors($form->get('password')); ?></td>
            </tr>
            <tr>
                <td><strong>Password:</strong></td>
                <td><?php echo $this->formElement($form->get('password')) ?></td>
            </tr>
            <tr>
                <td colspan="2"><?php echo $this->formElementErrors($form->get('role'));
```

```
?></td>
        </tr>
        <tr>
            <td><strong>Role:</strong></td>
            <td><?php echo $this->formElement($form->get('role')) ?></td>
        </tr>
        <tr align="right">
            <td colspan="2"><?php echo $this->formElement($form->get('submit'))
?></td>
        </tr>
    </table>
  <?php echo $this->form()->closeTag(); ?>
</div>
```

Hemos mostrado cada una de las partes del formulario por separado para tener un mayor control sobre su visualización. Haciendo uso de los diferentes view helpers podemos tener más o menos control sobre el renderizado del formulario.

http://framework.zend.com/manual/current/en/modules/zend.form.view.helpers.html

6. Validación en un controller

```
public function doCreateAction()
{
    $request = $this->getRequest();

    if ($request->isPost()) {
        $form = new UserForm();
        $userEntity = new User();
        $form->setInputFilter($userEntity->getInputFilter());
        $form->setData($request->getPost());

        if ($form->isValid()) {
            $formData = $form->getData();

            $data['email']      = $formData['email'];
            $data['password']   = $formData['password'];
            $data['role']       = $formData['role'];
            $data['date']       = date('Y-m-d H:i:s');

            $this->model->save($data);

            $this->redirect()->toRoute('user\account\index');
        }

        $form->prepare();
```

```
        $this->layout()->title = 'Create User - Error - Review your data';

        // we reuse the create view
        $view = new ViewModel(['form' => $form, 'isUpdate' => false]);
        $view->setTemplate('user/account/create.phtml');

        return $view;
    }

    $this->redirect()->toRoute('user\account\create');
}
```

Creamos el formulario y le asignamos el filtro y los datos. Utilizamos el método isValid() en caso de no serlo volvemos a mostrarlo con errores.

7. Rellenando un formulario a partir de una entidad

```
public function updateAction()
{
    $this->layout()->title = 'Update User';

    $user = $this->model->getById($this->params()->fromRoute('id'));

    $form = new UserForm();
    $form->setAttribute('action', $this->url()->fromRoute('user\account\doUpdate'));
    $form->bind($user);
    $form->get('submit')->setAttribute('value', 'Edit User');

    // we reuse the create view
    $view = new ViewModel(['form' => $form, 'isUpdate' => true]);
    $view->setTemplate('user/account/create.phtml');

    return $view;
}
```

Recuperamos un usuario de la base de datos y mediante el método bind() rellenamos el formulario con dicho usuario.