ZF - Session 5

1. TableGateway

Zend\Db\TableGateway es un componente que nos proporciona un objeto que representa una tabla en la base de datos.

Este objeto tiene ya implementadas las operaciones básicas sobre una tabla (listar, insertar, borrar, modificar, getById) con el ahorro que esto nos supone.

Otra de las ventajas de trabajar con TableGateway es que el conjunto de datos que devuelve una consulta ya son entidades que hemos definido en nuestro modelo, no necesitamos convertir desde un array asociativo como sucedía en la anterior implementación de la clase UserDao.

En muchos casos puede ser suficiente trabajar directamente con la instancia de TableGateway en el controller porque no vayamos a realizar ninguna consulta compleja y con los métodos predefinidos ya nos basta. No obstante siempre es aconsejable crear un model que realice todas esas llamadas al objeto de tipo TableGateway. De esta manera no repetiremos código en diferentes controllers que hagan uso del mismo TableGateway. Nosotros trabajaremos siempre de esta manera. Recuerda:

EN UN CONTROLLER NO HAY CÓDIGO REUTILIZABLE

2. Implementando UserDao con TableGateway

Primero tenemos que añadir el método exchangeArray() a la entidad User para que el objeto TableGateway pueda llenar (populate) la instancia de la entidad User:

```php
/**
 * 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;
}
```

Lo siguiente será crear un factory para UserDao en el que crearemos la instancia de TableGateway y se la inyectaremos vía contructor:

```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\Model\Factory;


use User\Model\User;
use User\Model\UserDaoTableGateWay;
use Zend\Db\ResultSet\ResultSet;
use Zend\Db\TableGateway\TableGateway;
use Zend\ServiceManager\FactoryInterface;
use Zend\ServiceManager\ServiceLocatorInterface;

class UserDaoTableGateWayFactory implements FactoryInterface
{

    /**
     * Create service
     *
     * @param ServiceLocatorInterface $serviceLocator
     *
     * @return mixed
     */
    public function createService(ServiceLocatorInterface $serviceLocator)
    {
        $adapter = $serviceLocator->get('database');
        $resultSetPrototype = new ResultSet();

        // Here is where we connect our Entity with the TableGateway
        $resultSetPrototype->setArrayObjectPrototype(new User());

        $tableGateWay = new TableGateway('users', $adapter, null, $resultSetPrototype);

        return new UserDaoTableGateWay($tableGateWay);
    }
}
```

Donde podemos ver que para crear la instancia de TableGateway necesitamos el nombre de la tabla con la que vamos a trabajar, el conector a la base de datos (adapter) y la entidad que usaremos para representar las filas de la tabla (Entidad User).

Damos de alta la clase UserDaoTableGateWay (así hemos llamado al Dao que usa TableGateway para diferenciarlo de la anterior implementación) en el module.config:

```
'service_manager' => array(
    'factories' => array(
        'database'  => 'Zend\Db\Adapter\AdapterServiceFactory', // needs a db key in
config
        'userDao'   => 'User\Model\Factory\UserDaoFactory',
        'userDaoTableGateWay' => 'User\Model\Factory\UserDaoTableGateWayFactory',
    ),
),
```

Ya solo nos queda la clase UserDaoTableGateWay:

```
<?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 User\Model\Interfaces\UserDaoInterface;
use Zend\Db\TableGateway\TableGateway;

class UserDaoTableGateWay implements UserDaoInterface
{
    /**
     * @var TableGateWay
     */
    private $tableGateWay;

    function __construct(TableGateway $tableGateway)
```

```
    {
        $this->tableGateWay = $tableGateway;
    }

    public function findAll()
    {
        return $this->tableGateWay->select();
    }

    public function getById($id)
    {
        $rows = $this->tableGateWay->select(['id' => $id]);

        return $rows->current();
    }

    public function save($data)
    {
        $data['date'] = date('Y-m-d H:i:s');
        $this->tableGateWay->insert($data);
    }

    public function delete($id)
    {
        $this->tableGateWay->delete(['id' => $id]);
    }

    public function update($data)
    {
        $this->tableGateWay->update($data, ['id' => $data['id']]);
    }
}
```

que como vemos implementa la interfaz UserDaoInterface. De esta manera el controller permanece inalterable y es independiente de la implementación que le pasemos de UserDao.

Vemos que en este caso la clase UserDao no hace apenas nada más que delegar en los métodos de TableGateway. Por eso encontraremos muchos artículos en los que directamente se utiliza la instancia de TableGateway en el controller.

Lo único que nos queda es crear un factory para el controller en el que le inyectamos la nueva implementación:

```
<?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\Controller\Factory;


use User\Controller\AccountController;
use Zend\ServiceManager\FactoryInterface;
use Zend\ServiceManager\ServiceLocatorInterface;

class AccountTableGateWayControllerFactory implements FactoryInterface
{

    /**
     * Create service
     *
     * @param ServiceLocatorInterface $serviceLocator
     *
     * @return mixed
     */
    public function createService(ServiceLocatorInterface $serviceLocator)
    {
        $sm = $serviceLocator->getServiceLocator();
        $model = $sm->get('userDaoTableGateWay');

        return new AccountController($model);
    }
}
```

y lo reflejamos en el module.config:

```
'controllers' => array(
    'invokables' => array(
        //'account' => 'User\Controller\AccountController',
    ),
    'factories' => array(
        //'account' => 'User\Controller\Factory\AccountControllerFactory',
        'account' => 'User\Controller\Factory\AccountTableGateWayControllerFactory',
    ),
),
```

3. TableGateway Features

Hay un argumento que no hemos utilizado a la hora de crear la instancia de TableGateway. Se trata de 'features'. Mediante este argumento se puede extender la funcionalidad de TableGateway sin necesidad de implementar una nueva clase que herede de la clase base.

El argumento feature puede ser:

- Un objeto Feature
- Un objeto FeatureSet
- Un array de objetos Feature

Por defecto tenemos algunas features predefinidas. Veamos por ejemplo la feature RowGatewayFeature que nos permitirá tener objetos RowGateway como resultados de un select. Lo cual facilita la edición de los resultados de un select:

```php
<?php

use Zend\Db\TableGateway\TableGateway;
use Zend\Db\TableGateway\Feature\RowGatewayFeature;

$table = new TableGateway('users', $adapter, new RowGatewayFeature('id'));
$rowset = $table->select(array('name' => 'larry'));

$user = $rowset->current();

//updating...
if ($user) {
    $user->name = 'John';
    $user->save();
}

//deleting...
$user = $table->select(array('name' => 'John'))->current();

if ($user) {
    $user->delete();
}
```

Tenemos más ejemplos de features predefinidos en la documentación oficial:

http://framework.zend.com/manual/current/en/modules/zend.db.table-gateway.html - tablegateway-features

4. Zend\Db\Sql

Dado que muchas bases de datos tienen sus propias variaciones del lenguaje SQL, sería bueno utilizar un lenguaje intermedio a modo de capa de abstracción, haciendo nuestro código independiente del motor de base de datos. Esto nos permitiría cambiar de motor de base de datos sin necesidad de rehacer nuestro código.

En ZF2 tenemos un componente que realiza esta tarea: Zend\Db\Sql, que nos proporciona los siguientes objetos para poder trabajar con nuestra base de datos:

```php
<?php

use Zend\Db\Sql\Sql;

$sql = new Sql($adapter);
$select = $sql->select(); // @return Zend\Db\Sql\Select
$insert = $sql->insert(); // @return Zend\Db\Sql\Insert
$update = $sql->update(); // @return Zend\Db\Sql\Update
$delete = $sql->delete(); // @return Zend\Db\Sql\Delete
```

Creando prepared statements:

```php
<?php

use Zend\Db\Sql\Sql;

$sql = new Sql($adapter);
$select = $sql->select();
$select->from('foo');
$select->where(array('id' => 2));

$statement = $sql->prepareStatementForSqlObject($select);
$results = $statement->execute();
```

Sin crear un statement:

```php
<?php

use Zend\Db\Sql\Sql;

$sql = new Sql($adapter);
$select = $sql->select();
$select->from('foo');
$select->where(array('id' => 2));

$selectString = $sql->getSqlStringForSqlObject($select);
$results = $adapter->query($selectString, $adapter::QUERY_MODE_EXECUTE);
```

Podemos también vincular los objetos Sql a una tabla en particular:

```php
<?php

use Zend\Db\Sql\Sql;

$sql = new Sql($adapter, 'foo');
$select = $sql->select();
$select->where(array('id' => 2)); // $select already has the from('foo') applied
```

En la documentación oficial puedes seguir leyendo sobre los usos de este componente:

http://framework.zend.com/manual/current/en/modules/zend.db.sql.html

Crearemos una consulta en la que intervengan dos tablas para ver un ejemplo completo. Se trata de la tabla 'Tags' y de la relación 'Tags_Bookmarks', en la que cada enlace queda relacionado con sus tags.

Para recuperar todos los tags de un Bookmark dado procedemos de la siguiente manera:

```php
/*
  * SELECT
*  t.name
  * FROM
*  Tags_Bookmark tb
*  INNER JOIN Tags t
*      ON tb.tagsId = t.id
  * WHERE
*  bookmarkId = ?
  */
 public function getTagsByBookmarkId($id)
 {
    $select = new Select();
    $select->from(['t' => 'Tags']);
    $select->columns(['name']);
    $select->join(['tb' => 'Tags_Bookmark'], 'tb.tagsId = t.id');
    $select->where('tb.bookmarkId = ' . $id);

    return $this->table->selectWith($select);
 }
```

En este ejemplo utilizamos el TableGateway para ejecutar una consulta, de esta manera nos aseguramos que el resultSet devuelto sea de instancias de la clase 'Tag'.

Ahora para recuperar los tags de todos los bookmarks para mostrarlos en la vista:

```php
/**
 * indexAction
```

```
     *
     * We need to copy $bookmarks ResultSet into an Array because we cannot iterate a
ResultSet twice
     * ResulSet does not allow rewind
     *
     * @return array
     */
    public function indexAction()
    {
        $bookmarks = $this->bookmarkDao->findAll();

        foreach ($bookmarks as $b) {
            $tags = $this->tagDao->getTagsByBookmarkId($b->getId());
            $b->tags = $tags;
            $bookmarksArray[] = $b;
        }

        return [
            'bookmarks' => $bookmarksArray,
        ];
    }
```

Donde hemos tenido que crear un array copia del resultSet Bookmarks ya que dicho resultSet no es iterable más de una vez y por tanto nos daría error en caso de intentar iterarlo en la vista.

Ahora sí en la vista podemos mostrar todos los tags de cada Bookmark:

```
<div>
    <table class="table table-bordered">
        <thead>
            <th>id</th>
            <th>url</th>
            <th>title</th>
            <th>description</th>
            <th>tags</th>
            <th>date</th>
            <th>votes</th>
            <th>idUser</th>
        </thead>
        <?php foreach ($bookmarks as $bookmark): ?>
        <tr>
            <td><a href="<?php echo $this->url('application\bookmark\info', ['id' =>
$bookmark->getId()]) ?>"><?php echo $this->escapeHtml($bookmark->getId());
?></a></td>
            <td><?php echo $this->escapeHtml($bookmark->getUrl()); ?></td>
            <td><?php echo $this->escapeHtml($bookmark->getTitle()); ?></td>
            <td><?php echo $this->escapeHtml($bookmark->getDescription()); ?></td>
            <td>
                <table>
```

```
                <?php foreach($bookmark->tags as $tag): ?>
                    <tr>
                        <td><?php echo $tag->getName(); ?></td>
                    </tr>
                <?php endforeach; ?>
            </table>
        </td>
        <td><?php echo $this->escapeHtml($bookmark->getDate()); ?></td>
        <td><?php echo $this->escapeHtml($bookmark->getVotes()); ?></td>
        <td><?php echo $this->escapeHtml($bookmark->getIdUser()); ?></td>
        <td><a class="btn btn-success" href="<?php echo $this-
>url('application\bookmark\update', ['id' => $bookmark->getId()])
?>">Update</a></td>
        <td><a class="btn btn-danger" href="<?php echo $this-
>url('application\bookmark\delete', ['id' => $bookmark->getId()]) ?>">Delete</a></td>
    </tr>
    <?php endforeach; ?>
</table>
</div>

<div>
    <a href="<?php echo $this->url('application\bookmark\addBookmark') ?>">Insert
New Bookmark</a>
</div>
```

Encontrarás más ejemplos de uso de Zend\Db\Sql en los siguientes enlaces:

https://github.com/ralphschindler/Zend_Db-Examples
http://avnpc.com/pages/advanced-database-select-usage-in-zf2
http://eltonminetto.net/blog/2013/03/21/subqueries-no-zend-framework-2/