# ZF - Session 4

# Acceso a bases de datos

Trabajaremos con bases de datos desde 2 enfoques diferentes:

- Usando directamente el Adapter PDO
- Usando la clase Zend\Db\TableGateway

Nuestra intención debe ser reutilizar el controller y las vistas en las diferentes implementaciones. Para ello utilizaremos las interfaces necesarias.

La base de datos con la que vamos a trabajar es la siguiente:

```
CREATE SCHEMA `zf2` DEFAULT CHARACTER SET utf8 COLLATE
utf8_general_ci ;
CREATE TABLE `zf2`.`users` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `email` VARCHAR(255) NOT NULL,
  `password` VARCHAR(80) NOT NULL,
  `role` VARCHAR(45) NOT NULL DEFAULT 'user',
  `date` TIMESTAMP NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `email_UNIQUE` (`email` ASC));

INSERT INTO `zf2`.`users` (`id`, `email`, `password`, `role`, `date`) VALUES
(NULL, 'user1@email.com', '1234', 'admin', CURRENT_TIMESTAMP),
(NULL, 'user2@email.com', '1234', 'user', CURRENT_TIMESTAMP),
(NULL, 'user3@email.com', '1234', 'user', CURRENT_TIMESTAMP),
(NULL, 'user4@email.com', '1234', 'user', CURRENT_TIMESTAMP),
(NULL, 'user5@email.com', '1234', 'user', CURRENT_TIMESTAMP);
```

## 1. Configuración DB

Lo primero que vamos a hacer es configurar la conexión con nuestra base de datos. Creamos el fichero 'database.local.php' dentro del directorio 'config/autoload':

```
<?php

return array(
    'db' => array(
        'driver'         => 'Pdo',
        'username'       => 'mvc',
        'password'       => '1234',
        'dsn'            => 'mysql:dbname=zf2;host=localhost',
        'driver_options' => array(
            PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES \'UTF8\''
        ),
    ),
);
```

Ahora vamos a registrar el recurso 'database' que será el adapter que utilizaremos. Para ello utilizaremos un Factory que nos proporciona ZF2, 'Zend\Db\Adapter\AdapterServiceFactory '. Este Factory espera encontrar la key 'db' en la configuración, por este motivo es obligatorio usar el nombre 'db'. El fichero module.config.php tiene ahora como servicio el adapter:

```
'service_manager' => array(
    'factories' => array(
        'database'  => 'Zend\Db\Adapter\AdapterServiceFactory',
    ),
),
```

Podemos repartir los anteriores valores de configuración en diferentes ficheros dentro de config/autoload. Por ejemplo podemos dejar en database.local.php solo los datos de las credenciales y el resto podría ir en otro fichero llamado database.global.php:

```
<?php

return array(
    'db' => array(
        'driver'            => 'Pdo',
        'dsn'               => 'mysql:dbname=zf2;host=localhost',
        'driver_options'    => array(
            PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES \'UTF8\'"
        ),
    ),
    'service_manager' => array(
        'factories' => array(
            'database'  => 'Zend\Db\Adapter\AdapterServiceFactory',
        ),
    ),
);
```

quedando el fichero database.local.php:

```
<?php

return array(
    'db' => array(
        'username'          => 'mvc',
        'password'          => '1234',
    ),
);
```

Registrar un servicio como hemos hecho con 'database' no significa que se cree una instancia. La instancia se crea cuando se solicita el servicio.

## 2. PDO

Debemos tener en cuenta que el adapter no es una instancia de PDO y que por tanto tiene sus propios métodos (next, current, ...).

Vamos a realizar las siguientes tareas:

- Crear la clase entidad User para almacenar filas de la tabla User.
- Crear la clase UserDao para implementar las operaciones sobre la tabla User. Esta clase es la que hará uso del adapter y por tanto lo requiere como dependencia.
- El controller AccountController utilizará una instancia de UserDao y actualizará las vistas. Este controller por tanto, depende de UserDao.
- Las vistas para cada una de las acciones y en las que tendremos que iterar sobre los valores devueltos por el modelo.

## 3. Entity User

```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;


class User
{
    private $id;
    private $email;
    private $password;
    private $role;
    private $date;

    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;
}

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

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

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

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

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

/**
 * @param mixed $password
```

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

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

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

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

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

## 4. UserDao

La clase UserDao utiliza el adapter y espera que le sea inyectado en el constructor. Vamos a crear el Factory para ello:

```
<?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\UserDao;
use Zend\ServiceManager\FactoryInterface;
use Zend\ServiceManager\ServiceLocatorInterface;

class UserDaoFactory implements FactoryInterface
{

    /**
     * Create service
     *
     * @param ServiceLocatorInterface $serviceLocator
     *
     * @return UserDao
     */
    public function createService(ServiceLocatorInterface $serviceLocator)
    {
        $database = $serviceLocator->get('database');
        return new UserDao($database);
    }
}
```

y lo damos de alta en el module.config.php:

```
'service_manager' => array(
    'factories' => array(
        'database'  => 'Zend\Db\Adapter\AdapterServiceFactory',
        'userDao'   => 'User\Model\Factory\UserDaoFactory',
    ),
),
```

El código de la clase UserDao:

```
<?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\Adapter\Adapter;

/**
 * Class UserDao
 *
 * Data Access Object for User
 *
 */
class UserDao implements UserDaoInterface
{
    /**
     * @var Adapter
     */
    private $db;

    /**
     * @param Adapter $db
     */
    function __construct(Adapter $db)
    {
        $this->db = $db;
    }

    public function findAll()
    {
        $resultSet = $this->db->query('SELECT * FROM users',
Adapter::QUERY_MODE_EXECUTE);
        $users = new \ArrayObject();
        $count = $resultSet->count();

        for ($i = 0; $i < $count; $i++) {
            $row = $resultSet->current();
            $user = new User($row->id, $row->email, $row->password, $row->role, $row-
>date);
```

```
            $users->append($user);
            $resultSet->next();
        }

        return $users;
    }

    /**
     * getById
     *
     * current() is not using FETCH_ASSOC as it is supposed, it is using
FETCH_ARRAY !!!
     *
     * @param $id
     *
     * @return User
     */
    public function getById($id)
    {
        $stmt = $this->db->createStatement('SELECT * FROM users WHERE id = ?');
        $resultSet = $stmt->execute([$id]);
        $row = $resultSet->current();

        return new User($row['id'], $row['email'], $row['password'], $row['role'],
$row['date']);
    }

    public function save($data)
    {
        $stmt = $this->db->createStatement('INSERT INTO users VALUES (NULL, ?, ?,
?, NULL)');
        $stmt->execute([$data['email'], $data['password'], $data['role']]);
    }

    public function delete($id)
    {
        $stmt = $this->db->createStatement('DELETE FROM users WHERE id = ?');
        $stmt->execute([$id]);
    }

    public function update($data)
    {
        $stmt = $this->db->createStatement('UPDATE users SET email = ?, password = ?,
role = ? WHERE id = ?');
        $stmt->execute([$data['email'], $data['password'], $data['role'], $data['id']]);
    }
}
```

Siempre que una consulta tenga parámetros usaremos statements para optimizarla.

Es importante que cuando no usamos statements pongamos el siguiente parámetro para que la consulta sea ejecutada:

```
Adapter::QUERY_MODE_EXECUTE
```

Hemos creado la interfaz UserDaoInterface para las dos diferentes implementaciones que haremos (ésta y en la que usaremos TableGateway):

```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\Interfaces;


interface UserDaoInterface
{
    public function findAll();
    public function getById($id);
    public function save($data);
    public function delete($id);
    public function update($data);
}
```

## 5. AccountController

El controller AccountController tendrá como dependencia una instancia de UserDao:

```php
<?php

namespace User\Controller;

use User\Model\Interfaces\UserDaoInterface;
```

```php
use Zend\Mvc\Controller\AbstractActionController;

class AccountController extends AbstractActionController
{
    /**
     * @var UserDaoInterface
     */
    private $model;

    function __construct(UserDaoInterface $model)
    {
        $this->model = $model;
    }

    public function indexAction()
    {
        $this->layout()->title = 'List Users';
        $users = $this->model->findAll();

        return ['users' => $users];
    }

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

        return [];
    }

    public function doCreateAction()
    {
        $this->model->save($this->params()->fromPost());
        $this->redirect()->toRoute('account');
    }

    public function viewAction()
    {
        $this->layout()->title = 'User Details';

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

        return ['user' => $user];
    }

    public function deleteAction()
    {
        $this->model->delete($this->params()->fromRoute('id'));

        $this->redirect()->toRoute('account');
```

```
    }

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

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

        return ['user' => $user];
    }

    public function doUpdateAction()
    {
        $this->model->update($this->params()->fromPost());

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

Creamos el Factory para inyectar la dependencia:

```
<?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 AccountControllerFactory implements FactoryInterface
{

    /**
     * Create service
     *
```

```
     * @param ServiceLocatorInterface $serviceLocator
     *
     * @return AccountController
     */
    public function createService(ServiceLocatorInterface $serviceLocator)
    {
        $sm = $serviceLocator->getServiceLocator();
        $userDao = $sm->get('userDao');
        return new AccountController($userDao);
    }
}
```

Y lo registramos en el fichero module.config.php en la sección controllers:

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

## 6. Rutas

Creamos las rutas para cada action:

```
<?php
return array(
    'router' => array(
        'routes' => array(
            'account' => array(
                'type' => 'Literal',
                'options' => array(
                    'route'    => '/account/',
                    'defaults' => array(
                        'controller' => 'account',
                        'action'     => 'index',
                    ),
                ),
            ),
            'account_view' => array(
                'type'              => 'Segment',
                'options'           => array(
                    'route'         => '/account/view/id/[:id]/',
                    'constraints'   => array(
                        'id' => '[0-9]+',
                    ),
                    'defaults' => array(
                        'controller' => 'account',
                        'action'     => 'view',
```

```
            ),
        ),
    ),
    'account_create' => array(
        'type' => 'Literal',
        'options' => array(
            'route'    => '/account/create/',
            'defaults' => array(
                'controller' => 'account',
                'action'     => 'create',
            ),
        ),
    ),
    'account_doCreate' => array(
        'type' => 'Literal',
        'options' => array(
            'route'    => '/account/do-create/',
            'defaults' => array(
                'controller' => 'account',
                'action'     => 'doCreate',
            ),
        ),
    ),
    'account_delete' => array(
        'type' => 'Segment',
        'options' => array(
            'route'    => '/account/delete/id/[:id]/',
            'constraints' => array(
                'id' => '[0-9]+',
            ),
            'defaults' => array(
                'controller' => 'account',
                'action'     => 'delete',
            ),
        ),
    ),
    'account_update' => array(
        'type' => 'Segment',
        'options' => array(
            'route'    => '/account/update/id/[:id]/',
            'constraints' => array(
                'id' => '[0-9]+',
            ),
            'defaults' => array(
                'controller' => 'account',
                'action'     => 'update',
            ),
        ),
    ),
    'account_doUpdate' => array(
```

```
            'type' => 'Literal',
            'options' => array(
                'route'    => '/account/do-update/',
                'defaults' => array(
                    'controller' => 'account',
                    'action'     => 'doUpdate',
                ),
            ),
        ),
    ),
),
```

## 7. Las vistas

La vista principal es la que lista los usuarios y crea enlaces para su borrado y edición:

```
<div>
    <table>
        <tr>
            <th>Id</th>
            <th>Email</th>
            <th>Password</th>
            <th>Role</th>
            <th>Date</th>
        </tr>
        <?php foreach($this->users as $user): ?>
        <tr>
            <td><?php echo $this->escapeHtml($user->getId()); ?></td>
            <td><?php echo $this->escapeHtml($user->getEmail()); ?></td>
            <td><?php echo $this->escapeHtml($user->getPassword()); ?></td>
            <td><?php echo $this->escapeHtml($user->getRole()); ?></td>
            <td><?php echo $this->escapeHtml($user->getDate()); ?></td>
            <td><a href="<?php echo $this->url('account_view', ['id' => $user->getId()])
?>">View</a></td>
            <td><a href="<?php echo $this->url('account_update', ['id' => $user->getId()])
?>">Update</a></td>
            <td><a href="<?php echo $this->url('account_delete', ['id' => $user->getId()])
?>">Delete</a></td>
        </tr>
        <?php endforeach; ?>
    </table>
</div>

<div>
    <p><a href="<?php echo $this->url('account_create') ?>">Create a new
Account</a> </p>
</div>
```

Para el formulario que usaremos tanto en el alta como en la modificación de usuarios utilizaremos un partial:

```
<?php
    $id         = (isset($this->id) ? $this->id : '');
    $email      = (isset($this->email) ? $this->email : '');
    $password   = (isset($this->password) ? $this->password : '');
    $role       = (isset($this->role) ? $this->role : '');
    $date       = (isset($this->date) ? $this->date : '');
?>

<div>
    <form id="form1" action="<?php echo $this->action ?>" method="post">
        <!-- Disabled fields will not be sent in $POST so we have to create hidden fields --
>
        <input type="hidden" name="id" value="<?php echo $id ?>">
        <input type="hidden" name="date" value="<?php echo $date ?>">
        <table>
            <tr>
                <td><strong>Id:</strong></td>
                <td><input type="number" name="id_disabled" value="<?php echo $id ?>"
disabled></td>
            </tr>
            <tr>
                <td><strong>Email:</strong></td>
                <td><input type="email" name="email" value="<?php echo $email
?>"></td>
            </tr>
            <tr>
                <td><strong>Password:</strong></td>
                <td><input type="text" name="password" value="<?php echo $password
?>"></td>
            </tr>
            <tr>
                <td><strong>Role:</strong></td>
                <td><input type="text" name="role" value="<?php echo $role ?>"></td>
            </tr>
            <tr>
                <td><strong>Date:</strong></td>
                <td><input type="date" name="date_disabled" value="<?php echo $date ?>"
disabled></td>
            </tr>
            <tr align="right">
                <td colspan="2"><input type="submit"></td>
            </tr>
        </table>
    </form>
</div>
```