ZF - Session 1

1. Qué es Zend Framework

PHP es un lenguaje del lado del servidor muy fácil de aprender. El 'Time to Market' es mínimo, llevar una idea a producción lleva muy poco tiempo. Esto que en principio puede parecer una ventaja, puede acabar no siéndolo. Cada programador puede tener su propio estilo, lo que implica que es muy difícil trabajar en equipo y que probablemente no se siguen las buenas prácticas de programación en PHP. Esto nos conduce a aplicaciones muy difíciles de mantener en el tiempo y que además, probablemente tendrán fallos de seguridad.

Cuando iniciamos un nuevo proyecto lo primero que nos planteamos es qué estructura de directorios utilizar. Es probable que cada programador a medida que va realizando proyectos, se vaya construyendo su propio marco de trabajo. Intentando reutilizar al máximo el trabajo ya realizado en proyectos anteriores. Tales como la estructura de carpetas o la división del código en componentes separados, siguiendo el patrón de diseño MVC por ejemplo. Por muy bueno que sea nuestro framework, tiene un problema inherente: es nuestro y nadie más lo conoce. Tanto si cambiamos de empresa como si viene alguien nuevo a nuestra empresa, debemos emplear un tiempo en aprender dicho framework.

Afortunadamente existen en PHP una gran cantidad de frameworks que podemos considerar como estándar. Zend Framework, Symfony, Laravel, yiiframework, PhalconPHP, ... y un largo etcétera (perdón por los que se quedan en el tintero). Con cualquiera de ellos podemos estar seguro de estar utilizando las mejores prácticas de programación en PHP. Algunos sin embargo, son más potentes que otros y están más indicados para proyectos de más envergadura. En este punto destacamos Zend Framework.

En este curso utilizaremos el que probablemente sea considerado el estándar de los frameworks: Zend Framework. Zend Framework, en adelante ZF, está desarrollado por la empresa que desarrolla PHP: Zend Technologies.

ZF es de código abierto y 100% orientado a objetos. ZF está débilmente acoplado, por lo que sus componentes pueden ser utilizados fuera del framework. Es seguro y extensible al estar basado en módulos.

ZF está actualmente en su versión 2. Nada o muy poco que ver con su predecesor ZF1 ya que prácticamente ha sido rediseñado desde cero. Sus principales características son:

- Uso de Namespaces
- MVC
- Orientado a eventos
- Basado en módulos
- Inyección de dependencias
- Service Locator

Durante el curso haremos un recorrido por todas ellas a medida que vayamos introduciendo nuevos componentes.

Antes de proseguir, debe quedar claro que este curso no es para principiantes. Son necesarias unas buenas bases en:

- Programación Orientada a Objetos
- MVC
- Namespaces
- A ser posible haber tenido contacto con algún framework

En alfa9 disponemos de un curso previo que abarca todos los puntos anteriores: "Introducción a los frameworks PHP".

De ahora en adelante, nos marcaremos un objetivo claro:

DO NOT REPEAT YOURSELF

No repetir código. Todo está enfocado a eso. El uso de ZF nos permitirá alcanzar nuestro objetivo. Si lo conseguimos tendremos un código muy fácil de mantener, y eso es lo que queremos ya que ahorraremos tiempo y dinero.

2. Instalando el entorno de desarrollo

Para poder seguir el curso solamente es necesario tener instalado PHP en nuestra máquina. Usaremos el servidor web que trae PHP incorporado.

http://php.net/manual/es/features.commandline.webserver.php

Los siguientes puntos son opcionales, hacemos referencia a ellos para todos aquellos que preferís tener vuestra pila LAMP instalada. Pero repetimos que solo teniendo PHP ya es suficiente tal y como se indica en los puntos: 3.4.2 y 3.4.2.1 para el caso de Windows.

1.1. Windows

Aquí tienes un vídeo tutorial para instalar PHP, git, composer y MySql en windows:

http://youtu.be/fIPC2bJAIIo

Si prefieres instalar un todo en uno:

http://www.wampserver.com/en/
http://www.easyphp.org/

1.2. Mac

1.2.1. Si quieres montarlo todo tu mismo

http://akrabat.com/php/setting-up-php-mysql-on-os-x-yosemite/
http://akrabat.com/computing/setting-up-php-mysql-on-os-x-mavericks/

1.2.2. Otra opción es instalar un todo en uno

http://www.mamp.info/en/
http://www.zend.com/en/products/server (de pago)

1.3. Linux

Usando el gestor de paquetes la instalación resulta muy sencilla.

1.3.1. Móntalo tu mismo

https://www.digitalocean.com/community/tutorials/como-instalar-linux-apache-mysql-php-lamp-en-ubuntu-14-04-es

1.3.2. O bien instala un todo en uno

https://www.apachefriends.org/download.html

1.4. Máquina virtual

Otra opción muy recomendable en cualquiera de los 3 sistemas es utilizar máquinas virtuales. Podemos usar 'vagrant' para configurar nuestra máquina virtual para que sea creada como un servidor LAMP:

https://www.virtualbox.org/wiki/Downloads
https://www.vagrantup.com/downloads
https://puphpet.com/

El primer enlace es para instalar 'VirtualBox', el segundo es para instalar 'Vagrant' y el tercero es para generar un fichero de configuración para 'Vagrant'.

1.5. IDE

Es recomendable el uso de un IDE para aumentar nuestra productividad. Existe una gran variedad de IDE's que puedes utilizar. A modo orientativo te proponemos dos de los más usados:

PhpStorm (de pago)
Netbeans

1.6. xDebug

xDebug nos permitirá ejecutar nuestro código paso a paso y ver el valor que van tomando las variables durante la ejecución. Para su instalación puedes seguir este wizard:

http://xdebug.org/wizard.php

1.7. GitHub

Usaremos GitHub como sistema de control de versiones y como plataforma para compartir nuestro código.

Todo el código del curso puede ser descargado desde el siguiente repositorio en GitHub:

https://github.com/itrascastro/zf2-tutorial

El código está organizado en diferentes 'branches' o ramas de manera que podamos seguir la evolución del curso sin perder nada de lo que vayamos aprendiendo.

3. Instalando Zend Framework

Una vez tenemos listo nuestro entorno de desarrollo, ya podemos proceder con la instalación de ZF.

Tenemos varías alternativas dependiendo de nuestras necesidades.

3.1. Instalación común para varios proyectos

Podemos colocar la librería de ZF en un directorio accesible por varios proyectos. Con lo que si una nueva versión de ZF es liberada, todos nuestros proyectos se actualizan a la vez.

Esto lo podemos conseguir de dos maneras:

3.1.1. Creando enlaces simbólicos

Desde nuestros proyectos podemos crear enlaces simbólicos a la carpeta vendor. Sería algo así:

```
ln -s path_to_vendor_folder/vendor vendor
```

3.1.2. Añadiendo la ruta al path de PHP

Otra opción es hacer la librería de ZF accesible a través del Path de PHP. Esto lo haríamos editando el fichero php.ini.

Debemos buscar y descomentar la entrada 'include_path' y ahí colocar la ruta de la librería de ZF.

3.2. Instalación única para cada proyecto

Cada proyecto tiene una versión de ZF. Pudiera ser que algunas actualizaciones de la librería fueran incompatibles con algunos de nuestros proyectos. No es lo normal, pero si alguien quiere cubrirse las espaldas puede optar por esta opción perfectamente. Aunque tiene el coste de mantener diversas versiones.

3.3. Instalación

Una vez tenemos claro como vamos a mantener las librerías de ZF, vamos a instalarlo. El proceso es el mismo independientemente de la opción que hayamos elegido.

Aunque ZF puede ser descargado directamente desde los repositorios de GitHub, vamos a optar por utilizar el gestor de dependencias para PHP Composer.

Siguiendo las indicaciones de la ayuda oficial de Zend:

Lo primero será tener instalado 'composer' y posteriormente utilizarlo para crear un proyecto con sus dependencias.

Asegúrate de que puedes ejecutar PHP desde la consola:

```
#php -v
```

En esta guía se detallan los pasos a seguir para instalar 'composer' en Windows:

https://getcomposer.org/doc/00-intro.md - installation-windows

Para linux y MAC se puede hacer directamente:

```
curl -s https://getcomposer.org/installer | php --

php composer.phar create-project -sdev --repository-
```

```
url="https://packages.zendframework.com" zendframework/skeleton-application myproject
```

Son dos comandos.

Puedes encontrar otras alternativas de instalación en la ayuda oficial:

http://framework.zend.com/manual/current/en/ref/installation.html

3.4. Configurando el servidor web

En este punto también tenemos dos opciones:

3.4.1. Virtual Host

Ya sea en xampp o en nuestra propia instalación de Apache, tendríamos que crear un Virtual Host con un contenido similar al siguiente:

```
<VirtualHost *:80>
                    ServerName zf2-tutorial.localhost
                    DocumentRoot /path/to/zf2-tutorial/public

                    <Directory /path/to/zf2-tutorial/public>
                    AllowOverride All
                    Order allow,deny
                    Allow from all
                    </Directory>
                </VirtualHost>
```

Si nuestra version de Apache es igual o superior a la 2.4 tendríamos que poner:

```
<VirtualHost *:80>
                    ServerName zf2-tutorial.localhost
                    DocumentRoot /path/to/zf2-tutorial/public

                    <Directory /path/to/zf2-tutorial/public>
                    AllowOverride All
                    Require all granted
                    </Directory>
                </VirtualHost>
```

3.4.2. Usar el servidor web que PHP trae incorporado

Para el proceso de desarrollo podemos utilizar el servidor que en las últimas versiones de PHP viene ya incorporado y que nos evita tener que estar creando y manteniendo Virtual Hosts para cada proyecto.

Para lanzar el 'PHP Buit-In Web Server':

Nos movemos a la carpeta de nuestro proyecto y desde la consola ejecutamos:

```
php -S 0.0.0.0:8080 -t public/ public/index.php
```

Donde la ip y el puerto pueden ser cualquier valor que no interfiera con la instalación de Apache que podamos tener instalada.

3.4.2.1. En Windows

Buscar la ruta del fichero 'php.exe' y ejecutar desde la carpeta del proyecto:

```
php.exe -S localhost:8080 -t public public\index.php
```

anteponiendo la ruta si no está en la variable de entorno PATH.

En realidad es suficiente con tener instalado PHP, no es necesario ningún todo en uno ni tan siquiera tener Apache.

Windows: http://windows.php.net/download

3.5. Mod_Rewrite

ZF utiliza un módulo de PHP para permitir la creación de rutas amigables. Asegúrate de tener instalado el módulo 'mod_rewrite' en tu instalación de Apache. Si usas el servidor web de PHP ya funciona la redirección con ZF2.

3.6. Configurando nuestro IDE

Ahora podemos crear un nuevo proyecto a partir de la instalación que acabamos de realizar de ZF.

Para ejecutar nuestro proyecto desde nuestro IDE, debemos configurar las opciones de ejecución y debugging.

3.5.1. PhpStorm

https://www.jetbrains.com/phpstorm/help/creating-and-editing-run-debug-configurations.html

Donde tenemos que poner los datos de nuestro Virtual Host o de nuestro 'PHP Buit-In Web Server'. Por ejemplo, en el segundo caso quedaría:

```
+ Php Web Application
                    Name: myproject
                Server:
                    + name: localhost
                        Host: 0.0.0.0
                        Port: 8080
```

3.5.2. Netbeans

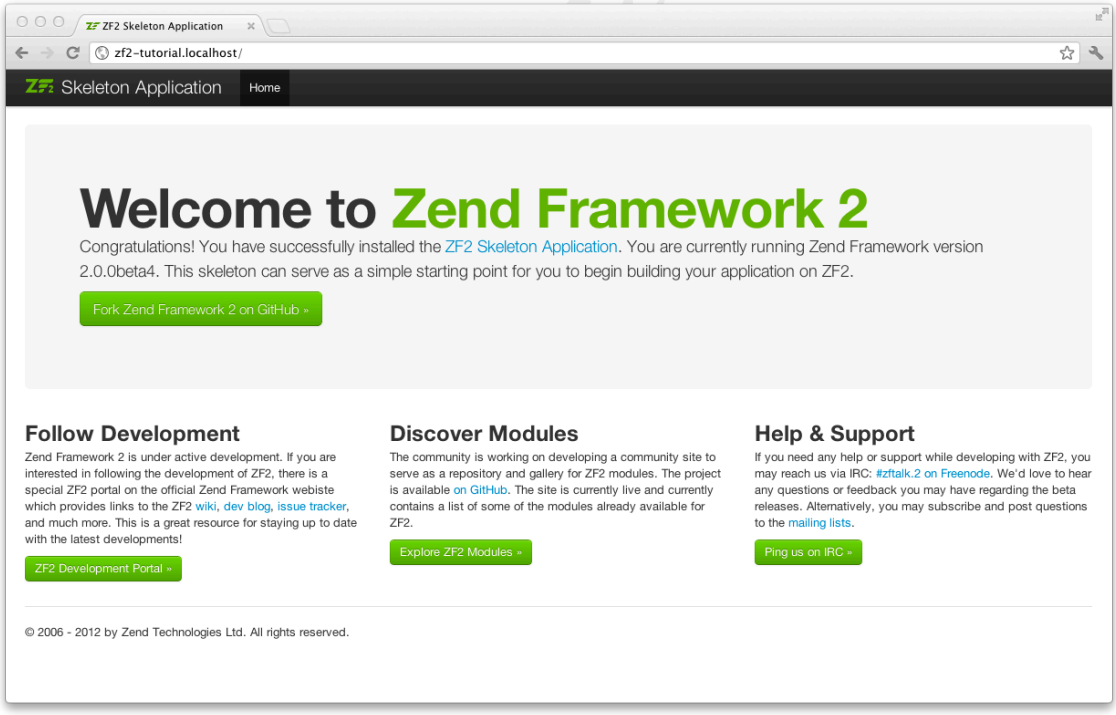
https://netbeans.org/kb/docs/php/project-setup.html

3.6. Probando nuestra instalación

Ahora ya podemos ir a nuestro navegador o ejecutar desde nuestro IDE y probar la instalación:

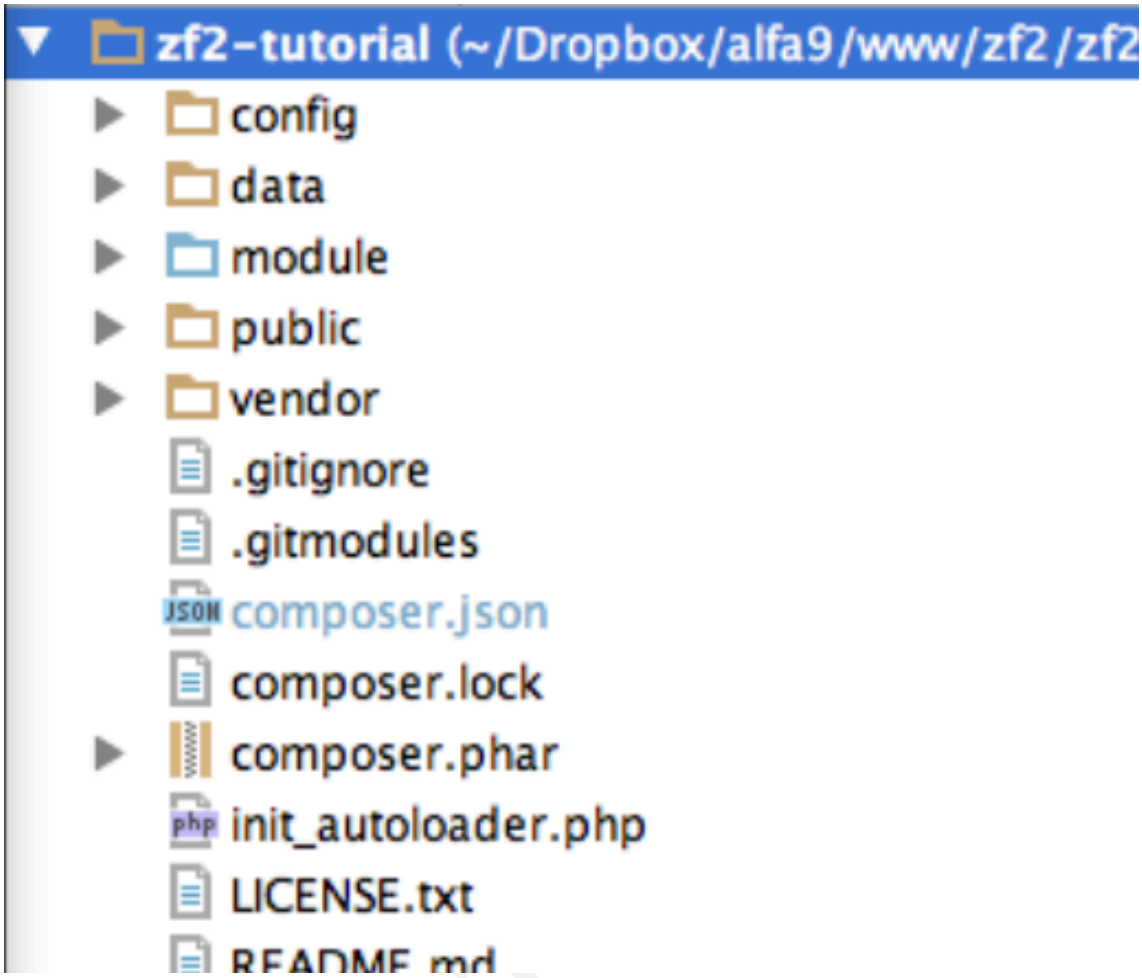
http://0.0.0.0:8080

Si todo ha ido bien deberíamos ver la página por defecto que ZF incorpora:



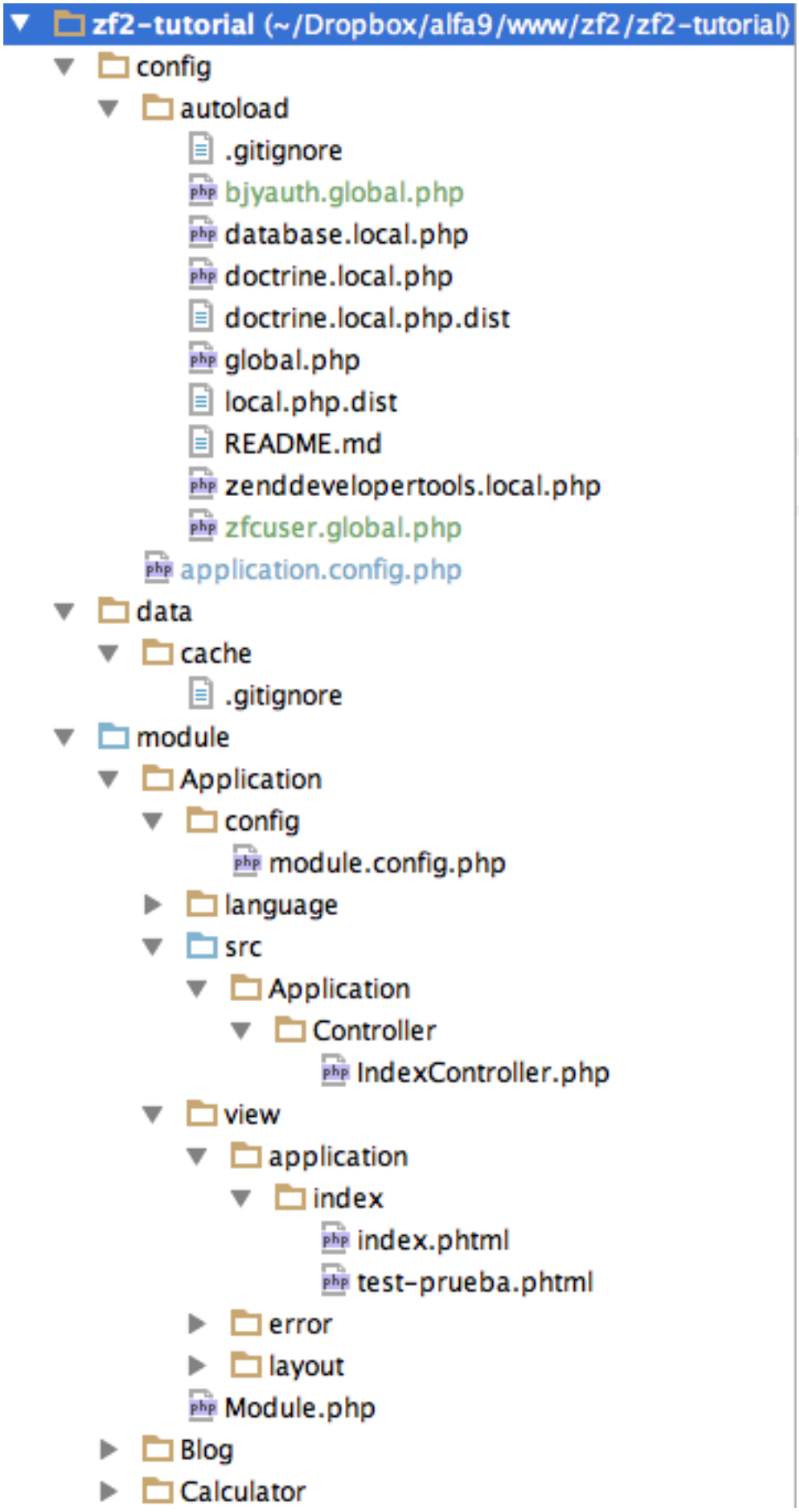
4. Estructura de directorios

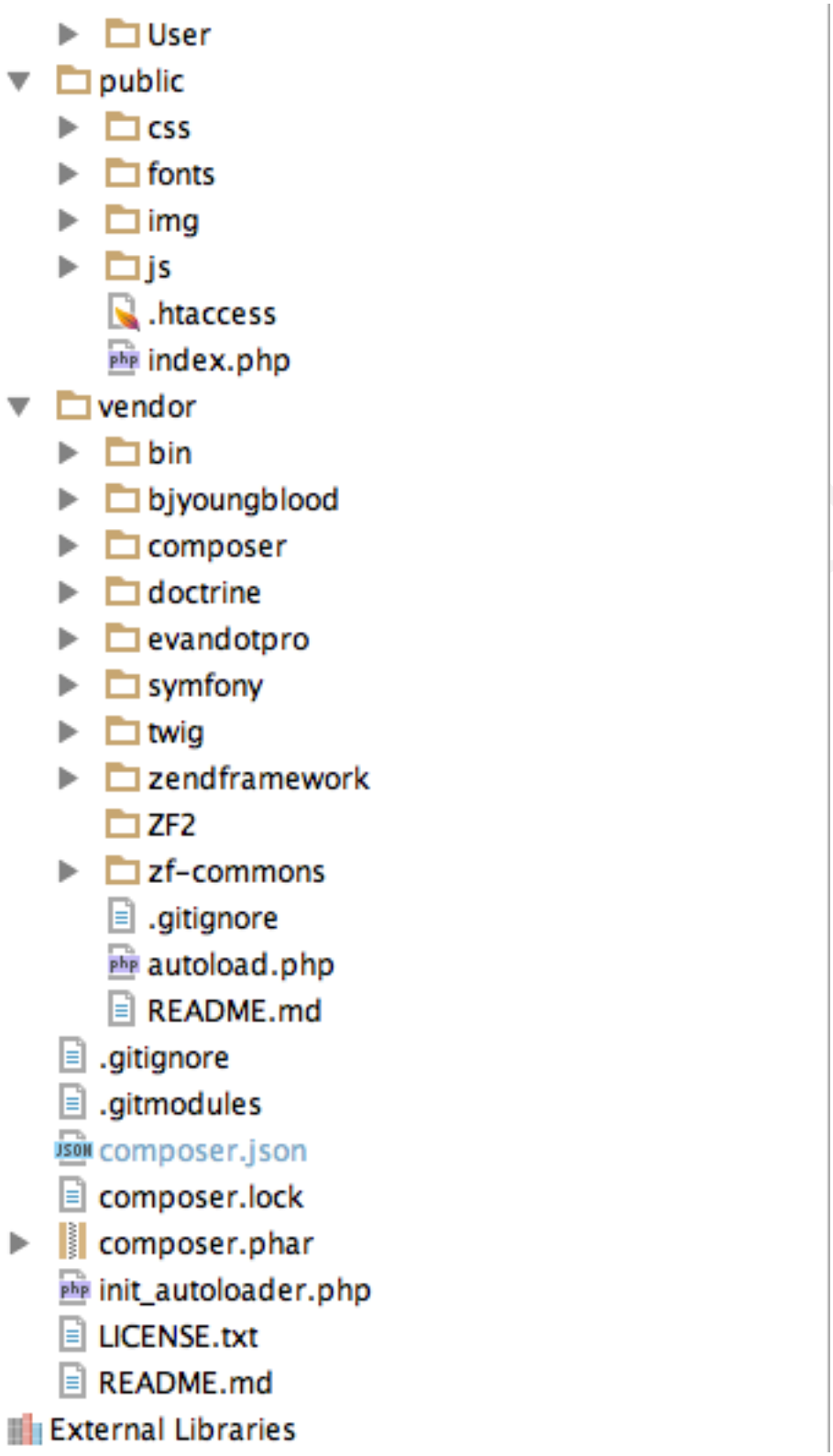
Cuando instalamos ZF nos encontramos con la siguiente estructura de carpetas:



config: Ficheros de configuración aplicables a toda la aplicación
data: Carpeta destinada a almacenar datos utilizados por la aplicación como por ejemplo la caché.
module: Esta será la carpeta donde residirá nuestro código. ZF organiza las aplicaciones en módulos. Partimos con un módulo inicial llamado 'Application'. Los módulos nos permitirán reutilizar código en diferentes aplicaciones.
public: Dentro de la carpeta public se ubican los ficheros estáticos de nuestra aplicación (.js, imágenes, css) junto con el único punto de acceso a la app: el fichero 'index.php'
vendor: Aquí se sitúan los módulos de terceros y todas las dependencias de nuestra app como por ejemplo, la propia librería de Zend.

A modo de ejemplo mostramos el contenido típico de las carpetas:


zf2-tutorial (~/Dropbox/alfa9/www/zf2/zf2-tutorial)
config
autoload
.gitignore
bjyauth.global.php
database.local.php
doctrine.local.php
doctrine.local.php.dist
global.php
local.php.dist
README.md
zenddevelopertools.local.php
zfcuser.global.php
application.config.php
data
cache
.gitignore
module
Application
config
module.config.php
language
src
Application
Controller
IndexController.php
view
application
index
index.phtml
test-prueba.phtml
error
layout
Module.php
Blog
Calculator



Donde podemos apreciar que dentro de la carpeta 'module/Application' es donde residen los 'controllers', 'views' y 'models' de dicho módulo.

Entraremos en detalle a medida que avancemos en el curso. En este punto es suficiente con tener una visión global.

5. Ficheros de configuración

En ZF2 encontraremos ficheros de configuración en diferentes ubicaciones.

- En el directorio 'config'.
- En el directorio 'config/autoload'.
- En el directorio 'config' de cada módulo.

5.1. application.config.php

Lo primero que ZF carga es el fichero 'config/application.config.php'. En este fichero es donde activamos los módulos que se utilizarán en la aplicación.

5.1.1. modules

```php
<?php
/**
 * Configuration file generated by ZFTool
 * The previous configuration file is stored in application.config.old
 *
 * @see https://github.com/zendframework/ZFTool
 */
return array(
    'modules' => array(
        'Application',
        'Calculator',
        'User',
        'EdpModuleLayouts',
        'ZendDeveloperTools',
        'DoctrineModule',
        'DoctrineORMModule',
        'Blog',
        'ZfcBase',
        'ZfcUser',
        'ZfcUserDoctrineORM',
        'BjyAuthorize',
    ),
);
```

Si un módulo no aparece en esta entrada, no se cargará y por tanto, no podrá utilizarse. Insistimos en ello ya que suele ser un olvido habitual.

El orden en el que los módulos son añadidos es importante como veremos más adelante.

5.1.2. module_listener_options

En este fichero también nos encontramos con la entrada 'module_listener_options':

```
'module_listener_options' => array(
    'module_paths' => array(
        './module',
        './vendor'
    ),
    'config_glob_paths' => array(
        'config/autoload/{,*.}{global,local}.php'
    )
)
```

Normalmente no tocaremos nada aquí, ya que los valores definidos son suficiente.

5.1.2.1. module_paths

En 'module_paths' se indica donde ZF puede encontrar módulos. En este caso, en dos sitios:

1. Bajo el directorio 'module'

Serán los módulos implementados por nosotros mismos.

2. Bajo el directorio 'vendor'

Serán los módulos implementados por terceros.

5.1.2.2. config_glob_paths

Aquí indicamos rutas donde ZF puede encontrar ficheros de configuración que sobrescribirán a los de los módulos. Normalmente ubicados bajo el directorio 'config/autoload'.

Primero cargará los ficheros 'global.php' y 'local.php', a continuación todos aquellos con la terminación '*.global.php' y '*.local.php'.

5.2. config/autoload

Como hemos visto en el punto anterior, en este directorio se guardan los ficheros '*.global.php' y '*.local.php'.

Éstos ficheros de configuración prevalecen sobre los de los módulos.

Los '*.global.php' contienen valores que son aplicables en cualquier entorno, mientras que los '*.local.php' hacen referencia a un entorno específico. Parámetros de la base de datos o de la caché son buenos ejemplos de ficheros '*.local.php'.

Debemos indicar en este punto que por defecto los ficheros '*.local.php' son ignorados por el sistema de control de versiones git. Esto es así porque en estos ficheros puede haber información sensible como por ejemplo las credenciales de nuestra base de datos.

Es posible que algunos módulos de terceros incluyan ficheros con la terminación 'dist': '*.global.php.dist', '*.local.php.dist'. Éstos ficheros deben ser movidos al directorio 'config/autoload' eliminando la terminación 'dist'.

5.3. Ficheros de configuración de módulo

5.3.1. Module.php

Dentro de cada módulo encontraremos un fichero llamado 'Module.php'. Se trata del único fichero que es obligatorio cuando creamos un módulo. Podemos crear un módulo únicamente con este fichero, veremos que es algo habitual. En este fichero encontramos dos métodos que nos sirven para indicar la ruta del fichero de configuración del módulo, y otro para indicar la ruta donde estarán las clases asociadas al NAMESPACE del módulo. En principio no tenemos por qué editar dichos métodos, pero sirva para indicar que en ZF2 todo es configurable, incluida la estructura de directorios del módulo.

```php
<?php
namespace Blog;

class Module
{
    public function getConfig()
    {
        return include __DIR__ . '/config/module.config.php';
    }

    public function getAutoloaderConfig()
    {
        return array(
            'Zend\Loader\StandardAutoloader' => array(
                'namespaces' => array(
                    __NAMESPACE__ => __DIR__ . '/src/' . __NAMESPACE__,
                ),
            ),
        );
    }
}
```

Este fichero puede contener también un método para realizar tareas de Bootstrap del módulo:

```php
public function onBootstrap(MvcEvent $e)
{
    $eventManager        = $e->getApplication()->getEventManager();
    $moduleRouteListener = new ModuleRouteListener();
    $moduleRouteListener->attach($eventManager);
}
```

Este método es llamado una vez todos los módulos han sido cargados y la configuración ya ha sido generada a partir de la mezcla de todos los ficheros de configuración.

Utilizaremos el método 'onBootstrap' para:

- Código que queremos que se ejecute en cada petición.
- Asignar manejadores de eventos a eventos.
- Definir nuevas rutas

5.3.2. module.config.php

En el directorio 'config' de cada módulo se encuentra el fichero 'module.config.php'.

Este fichero se usa para la configuración del módulo. En él podemos especificar:

- routes
- services
- controllers
- views
- ...

Entraremos en detalle cuando creemos nuestro primer módulo, pero a modo de introducción este podría ser el contenido típico del fichero:

```php
<?php
/**
 * Zend Framework (http://framework.zend.com/)
 *
 * @link      http://github.com/zendframework/ZendSkeletonApplication for the
canonical source repository
 * @copyright Copyright (c) 2005-2014 Zend Technologies USA Inc.
(http://www.zend.com)
 * @license   http://framework.zend.com/license/new-bsd New BSD License
 */

return array(
    'router' => array(
        'routes' => array(
            'home' => array(
                'type' => 'Zend\Mvc\Router\Http\Literal',
```

```
            'options' => array(
                'route'    => '/',
                'defaults' => array(
                    'controller' => 'Application\Controller\Index',
                    'action'     => 'index',
                ),
            ),
        ),
        // The following is a route to simplify getting started creating
        // new controllers and actions without needing to create a new
        // module. Simply drop new controllers in, and you can access them
        // using the path /application/:controller/:action
        'application' => array(
            'type'    => 'Literal',
            'options' => array(
                'route'    => '/application',
                'defaults' => array(
                    '__NAMESPACE__' => 'Application\Controller',
                    'controller'    => 'Index',
                    'action'        => 'index',
                ),
            ),
            'may_terminate' => true,
            'child_routes' => array(
                'default' => array(
                    'type'    => 'Segment',
                    'options' => array(
                        'route'    => '/[:controller[/:action]]',
                        'constraints' => array(
                            'controller' => '[a-zA-Z][a-zA-Z0-9_-]*',
                            'action'     => '[a-zA-Z][a-zA-Z0-9_-]*',
                        ),
                        'defaults' => array(
                        ),
                    ),
                ),
            ),
        ),
    ),
),
'service_manager' => array(
    'abstract_factories' => array(
        'Zend\Cache\Service\StorageCacheAbstractServiceFactory',
        'Zend\Log\LoggerAbstractServiceFactory',
    ),
    'aliases' => array(
        'translator' => 'MvcTranslator',
    ),
),
'translator' => array(
```

```
        'locale' => 'en_US',
        'translation_file_patterns' => array(
            array(
                'type'     => 'gettext',
                'base_dir' => __DIR__ . '/../language',
                'pattern'  => '%s.mo',
            ),
        ),
    ),
    'controllers' => array(
        'invokables' => array(
            'Application\Controller\Index' => 'Application\Controller\IndexController'
        ),
    ),
    'view_manager' => array(
        'display_not_found_reason' => true,
        'display_exceptions'       => true,
        'doctype'                  => 'HTML5',
        'not_found_template'       => 'error/404',
        'exception_template'       => 'error/index',
        'template_map' => array(
            'Application/layout'      => __DIR__ . '/../view/layout/layout.phtml',
            'application/index/index' => __DIR__ . '/../view/application/index/index.phtml',
            'error/404'               =>    DIR    . '/../view/error/404.phtml',
            'error/index'             => __DIR__ . '/../view/error/index.phtml',
        ),
        'template_path_stack' => array(
            __DIR__ . '/../view',
        ),
    ),
    // Placeholder for console routes
    'console' => array(
        'router' => array(
            'routes' => array(
            ),
        ),
    ),
);
```

5.4. Ejemplo de módulo para configurar los valores de PHP para nuestro proyecto

En este ejemplo vamos a crear un fichero de configuración para definir los parámetros de PHP para nuestro proyecto. A continuación crearemos un módulo en el que, a través del método 'onBootstrap' del 'Module.php', setearemos esos valores en nuestro entorno de ejecución.

El fichero de configuración se llama 'module.itrascastroPhpSettings.global.php':

```php
<?php
/**
 * My Global Configuration
 * /config/autoload/module.itrascastroPhpSettings.global.php
 */

return array(
    'phpSettings'   => array(
        'display_startup_errors'        => false,
        'display_errors'                => false,
        'max_execution_time'            => 60,
        'date.timezone'                 => 'Europe/Madrid',
        'mbstring.internal_encoding'    => 'UTF-8',
    ),
);
```

El fichero 'Module.config':

```php
<?php

class Module
{
    public function onBootstrap(MvcEvent $e)
    {
        $config = $e->getApplication()->getServiceManager()->get('config');
        $phpSettings = $config['phpSettings'];

        if ($phpSettings) {
            foreach($phpSettings as $key => $value) {
                ini_set($key, $value);
            }
        }
    }
}
```

5.5. Cómo acceder a la configuración

Podemos acceder a la configuración desde diferentes puntos de nuestra aplicación. ZF crea un servicio llamado 'config' en el contenedor de servicios y que está accesible a través del componente ServiceLocator.

Por ejemplo, podríamos definir la siguiente variable para almacenar el nombre de nuestra aplicación en el fichero 'global.php' dentro de la carpeta 'autoload':

```php
'application' => array(
    'name' => 'MyProject',
),
```

En un action de un controller podríamos acceder a dicho valor de la siguiente manera:

```
public function indexAction()
{
    $config = $this->serviceLocator->get('config');
    $appName = $config['application']['name'];

    return array('appName' => $appName);
}
```

Para posteriormente usarlo en una vista:

```
<p><?php echo $appName; ?></p>
```

En el fichero 'Module.php' podríamos hacer uso del servicio 'config' en el método 'onBootstrap':

```
public function onBootstrap(MvcEvent $e)
{
    $layout = $e->getViewModel();
    $config = $e->getApplication()->getServiceManager()->get('config');
    $layout->appName = $config['application']['name'];
}
```

6. Calculator - Una aplicación tutorial

Para poder explicar los diferentes componentes de ZF2 que vamos a utilizar utilizaremos una sencilla aplicación.

Se trata de una calculadora. Implementaremos las 4 operaciones básicas (suma, resta, producto y división).

Esta sencilla aplicación nos servirá para ilustrar:

- Módulos
- Controllers y Actions
- Views
- Models
- Layouts
- Configuración
    - Rutas
    - Servicios
    - Controllers
    - Views
- Variables de vista y de layout
- View Helpers

- ServiceManager
- ControllerManager
- Dependency Injection
    - Constructor injection
    - Setter injection
- Zend Tool

7. Creación de módulos

Lo primero que necesitamos para implementar nuestra calculadora es crear un nuevo módulo. Tenemos diferentes opciones para hacerlo:

7.1. Utilizar el módulo 'Application' como plantilla

Cuando instalamos ZF2 por defecto se nos crea un modulo llamado 'Application'. Podemos utilizar este módulo a modo de plantilla.

Lo que tendremos que hacer será copiarlo y pegarlo con otro nombre dentro de la carpeta modules.

Lo siguiente será actualizar el NAMESPACE por el nuevo. Para ello buscaremos las apariciones del NAMESPACE 'Application' y las sustituiremos por 'Calculator'. Tendremos que buscar en 3 archivos:

- Calculator/Module.php
- Calculator/config/module.config.php
- Calculator/src/Calculator/Controller/IndexController.php

Por último debemos añadir el nuevo módulo al archivo de configuración 'config/application.config.php'.

7.2. ModuleSkeleton

Podemos descargar desde el repositorio de Zend un esqueleto de módulo:

https://github.com/zendframework/ZendSkeletonModule

A continuación tendríamos que aplicar los mismos cambios explicados en el apartado 7.1

7.3. Utilizar Zend Tool

Esta opción es tratada en detalle en el punto 9. Sirva como adelanto el siguiente ejemplo:

```
php vendor/bin/zf.php create module Calculator
```

7.4. Creando el módulo Calculator

Aunque la opción preferida será utilizar Zend Tool, ya que perderemos menos tiempo editando ficheros, es recomendable que en nuestro primer proyecto lo hagamos a partir del módulo por defecto 'Application'. Hacerlo así nos permitirá familiarizarnos con el esqueleto de un módulo y los diferentes archivos de configuración del módulo.

Si quisiéramos crear un nuevo proyecto procederíamos así:

```
composer create-project -sdev --repository-url="https://packages.zendframework.com"
zendframework/skeleton-application Calculator
```

ahora lanzamos el servidor web que trae PHP incorporado:

```
cd Calculator
php -S localhost:8080 -t public public/index.php
```

Y ahora vamos a crear el módulo 'Calculator' a partir del módulo por defecto 'Application'.

Copiamos y pegamos cambiando el nombre del módulo.

Cambiamos todas las apariciones del NAMESPACE 'Application' por 'Calculator'. Recordamos que un módulo es en esencia un nuevo NAMESPACE.

Cambiamos el nombre de la carpeta 'src/Application' por 'src/Calculator'.

Cambiamos el nombre de la carpeta 'view/application' por 'view/calculator'. En las vistas no hay clases, y los nombres de carpetas no formarán parte de ningún NAMESPACE. Es por eso que van en minúsculas.

Empezaremos por el fichero más importante del módulo: Module.php

Cambiamos el NAMESPACE, ahora será 'Calculator'. En 'PhpStorm' a veces no reconoce el nuevo NAMESPACE y es necesario incluir una '\' al principio y luego la quitamos. Debe ser un bug.

En Module.config nos encontramos con 3 métodos:

```
<?php
/**
 * Zend Framework (http://framework.zend.com/)
 *
 * @link      http://github.com/zendframework/ZendSkeletonApplication for the
canonical source repository
 * @copyright Copyright (c) 2005-2014 Zend Technologies USA Inc.
```

```
(http://www.zend.com)
 * @license   http://framework.zend.com/license/new-bsd New BSD License
 */

namespace Calculator;

use Zend\Mvc\ModuleRouteListener;
use Zend\Mvc\MvcEvent;

class Module
{
    public function onBootstrap(MvcEvent $e)
    {
        $eventManager        = $e->getApplication()->getEventManager();
        $moduleRouteListener = new ModuleRouteListener();
        $moduleRouteListener->attach($eventManager);
    }

    public function getConfig()
    {
        return include __DIR__ . '/config/module.config.php';
    }

    public function getAutoloaderConfig()
    {
        return array(
            'Zend\Loader\StandardAutoloader' => array(
                'namespaces' => array(
                    __NAMESPACE__ => __DIR__ . '/src/' . __NAMESPACE__,
                ),
            ),
        );
    }
}
```

El método getConfig

Este método se utiliza para indicar a ZF dónde encontrar nuestro fichero de configuración del módulo. No es necesario cambiarlo a no ser que queramos especificar otro nombre de fichero u otra ubicación. ZF nos permite eso, podemos decidir la estructura de nuestro proyecto.

El método getAutoloaderConfig

El segundo método se utiliza para indicar en qué carpeta se van a alojar todas las clases que pertenezcan al NAMESPACE que acabamos de crear. Como siempre, todo configurable. Lo dejamos como viene por defecto.

El método onBoostrap

Lo utilizaremos para ejecutar código en cada petición. De momento no lo necesitamos e incluso podríamos borrarlo. En cambio en módulos que solo contienen el fichero Module.php, sería en este método donde pondríamos nuestro código.

Lo siguiente sería modificar el NAMESPACE de la clase 'IndexController' dentro de 'src/Calculator/Controller':

```php
<?php
/**
 * Zend Framework (http://framework.zend.com/)
 *
 * @link      http://github.com/zendframework/ZendSkeletonApplication for the
canonical source repository
 * @copyright Copyright (c) 2005-2014 Zend Technologies USA Inc.
(http://www.zend.com)
 * @license   http://framework.zend.com/license/new-bsd New BSD License
 */

namespace Calculator\Controller;

use Zend\Mvc\Controller\AbstractActionController;
use Zend\View\Model\ViewModel;

class IndexController extends AbstractActionController
{
    public function indexAction()
    {
        return new ViewModel();
    }
}
```

Por último editamos el fichero 'config/module.config.php'.

Lo primero que haremos será eliminar la ruta 'application':

```
// The following is a route to simplify getting started creating
// new controllers and actions without needing to create a new
// module. Simply drop new controllers in, and you can access them
// using the path /application/:controller/:action
'application' => array(
    'type'    => 'Literal',
    'options' => array(
        'route'    => '/application',
        'defaults' => array(
            '__NAMESPACE__' => 'Application\Controller',
            'controller'    => 'Index',
            'action'        => 'index',
        ),
    ),
```

```
    'may_terminate' => true,
    'child_routes' => array(
        'default' => array(
            'type'    => 'Segment',
            'options' => array(
                'route'    => '/[:controller[/:action]]',
                'constraints' => array(
                    'controller' => '[a-zA-Z][a-zA-Z0-9_-]*',
                    'action'     => '[a-zA-Z][a-zA-Z0-9_-]*',
                ),
                'defaults' => array(
                ),
            ),
        ),
    ),
),
```

Esta ruta que acabamos de eliminar se utiliza en la etapa de desarrollo cuando aún no tenemos rutas definidas. De manera que podemos acceder a cualquier action de un controller poniendo en la url el módulo, el controller y el action:

http://localhost:8080/calculator/index/index

Vamos a definir ahora la ruta de entrada a nuestro módulo:

```
'router' => array(
    'routes' => array(
        'calculator' => array(
            'type' => 'Zend\Mvc\Router\Http\Literal',
            'options' => array(
                'route'    => '/calculator/',
                'defaults' => array(
                    'controller' => 'Calculator\Controller\Index',
                    'action'     => 'index',
                ),
            ),
        ),
    ),
),
```

Aquí estamos diciendo que hay una ruta que se llama 'calculator' y que es accesible poniendo la url 'http://localhost:8080/calculator/'. En el parámetro 'defaults' indicamos qué controller y que action serán llamados para responder a la petición.

Ahora es un buen momento para crear el controller Index y el action index:

Creamos una clase llamada 'IndexController' dentro de la carpeta 'src/Calculator/Controller'. El NAMESPACE de esta nueva clase es 'Calculator\Controller'.

El controller que acabamos de crear debe heredar de la clase 'AbstractActionController' ya que es un controller que va a usar vistas. El heredar de esta clase nos va a dar acceso en nuestro controller a multiples componentes y helpers que nos serán de utilidad. Tenemos acceso por ejemplo al ServiceLocator, a través del cual podemos recuperar servicios que están disponibles en un contenedor de servicios. A modo de ejemplo podemos recuperar el servicio 'config'. Servicios en ZF2 pueden ser arrays, instancias de clases como puede ser un modelo, ..., o incluso los controllers.

Nos queda por dar de alta en el fichero 'module.config.php' el identificador de controller que estamos usando en la ruta:

```
'controllers' => array(
    'invokables' => array(
        'Calculator\Controller\Index' => 'Calculator\Controller\IndexController'
    ),
),
```

Aquí estamos diciendo que el identificador 'Calculator\Controller\Index' hace referencia a la clase 'Calculator\Controller\IndexController'. Podríamos haber elegido cualquier otro identificador, pero este es un buen nombre porque es fácil asociarlo visualmente con la clase que lo define.

El código del controller quedaría así:

```
<?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;


use Zend\Mvc\Controller\AbstractActionController;

class IndexController extends AbstractActionController
```

```
{
    public function indexAction()
    {
        $config = $this->serviceLocator->get('config');
        return ['appName' => $config['application']['name']];
    }
}
```

Observa que podemos devolver directamente un array. ZF ya lo convierte por nosotros en una instancia de ViewModel.

Donde estamos recuperando a través del ServiceLocator el servicio 'config'. Y estamos utilizando un valor del array $config para pasarlo a la vista.

Podemos crear ese parámetro en cualquier fichero de configuración. Un buen sitio sería por ejemplo el fichero 'config/autoload/global.php':

```
<?php
/**
 * Global Configuration Override
 *
 * You can use this file for overriding configuration values from modules, etc.
 * You would place values in here that are agnostic to the environment and not
 * sensitive to security.
 *
 * @NOTE: In practice, this file will typically be INCLUDED in your source
 * control, so do not include passwords or other sensitive information in this
 * file.
 */

return array(
    'application' => array(
        'name' => 'Calculator App',
    ),
);
```

Nos queda crear la vista asociada a nuestro action. Para ello creamos la carpeta 'view/calculator/index' y dentro el fichero 'index.phtml' con el siguiente contenido:

```
<?php echo $appName; ?>
```

En ZF2 no es necesario utilizar $this para acceder a las variables de vistas como sí ocurría en ZF1. Aunque si se desea puede seguirse utilizando de la misma manera.

Dentro de la carpeta view hemos creado una carpeta que se llama igual que nuestro módulo. Y dentro de la carpeta del módulo hemos creado otra carpeta que se llama igual que el controller. Necesitaremos una carpeta para cada controller.

Para finalizar, para que no se sobrescriba la configuración de las vistas del módulo 'Application', cambiamos un parámetro del viewManager. Se trata del parámetro que está dentro de 'template_map' y que apunta a la vista para el controller index y el action index:

```
'view_manager' => array(
    'display_not_found_reason' => true,
    'display_exceptions'       => true,
    'doctype'                  => 'HTML5',
    'not_found_template'       => 'error/404',
    'exception_template'       => 'error/index',
    'template_map' => array(
        'layout/layout'           => __DIR__ . '/../view/layout/layout.phtml',
        'calculator/index/index' => __DIR__ . '/../view/calculator/index/index.phtml',
        'error/404'               => __DIR__ . '/../view/error/404.phtml',
        'error/index'             => __DIR__ . '/../view/error/index.phtml',
    ),
    'template_path_stack' => array(
        __DIR__ . '/../view',
    ),
),
```

Como siempre en ZF2 todo es configurable. Aquí podemos ver cómo nuestras vistas podrían estar en cualquier ruta que queramos. Se puede conseguir cambiando el valor o añadiendo más rutas a la key 'template_path_stack'.

Usaremos el apartado 'template_map' de la configuración del 'view_manager' para esto. Aquí se crean unos alias que son utilizados bien por el framework o bien por nosotros mismos en nuestro código. Por defecto ZF cargará el layout configurado como 'layout/layout'. Aquí podríamos especificar nuestros propios identificadores para otros ficheros de vistas, como podría ser un partial. Más adelante hablaremos más en detalle sobre layouts, vistas y partials.