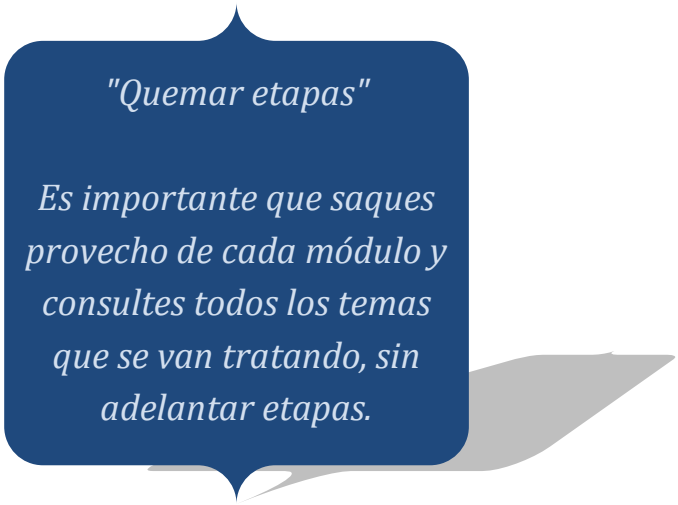
"Presentación de Zend Framework"

Módulo 1

Objetivos

El objetivo de este módulo semanal es hacer **la primera presentación formal** de Zend Framework como herramienta de desarrollo basada en *PHP5*. La intención es comprender y discutir todo el alcance y las posibilidades de la herramienta y culminar la primera etapa con **la instalación de un sistema base** y realizar la infalible y nunca bien valorada prueba clásica de *"hola mundo"* ☺



Introducción

Zend Framework es un proyecto open source para desarrollar aplicaciones web usando puramente en PHP5 y Programación Orientada a Objetos bajo licencia New BSD License.

Zend Framework comúnmente abreviado como **ZF**, es un producto desarrollado por la empresa **Zend Technologies [*]** y está diseñado con el objetivo de simplificar el desarrollo de sistemas web implementando las mejores prácticas y patrones de diseños (*design patterns*) de ingeniería de software, enfocado en la construcción de seguras, fiables y modernas aplicaciones Web 2.0.

[*] erróneamente se cree que la empresa creó PHP y que el autor original trabaja en ella (una locura, odia los frameworks ;-)); todo lo contrario, pero es verdad que la empresa fue formada por los principales desarrolladores del lenguaje y que actualmente son quienes más aportan a su desarrollo.

Nota importante: el curso estará basado en **la última versión disponible (1.10.x)** la cual incorpora muchas funcionalidades que automatizan procesos de creación de componentes de nuestro sistema, pero por temas didácticos **desde el principio del curso haremos "todo a mano"** y recién al final del curso abordaremos las últimas mejoras.

¿Por qué ZF?

Extendiendo el arte y el espíritu de PHP, **ZF se basa en la simplicidad**, las mejores prácticas de programación y ofrece flexibilidad y reducción de costos (licencias) a través de un producto de calidad y rigurosamente testeado .

"Demasiadas ruedas redondas"

Una de las cosas que como desarrolladores hay que tratar de evitar es "**reinventar la rueda**", ya que existen demasiados Frameworks para usar y API's para reusar, y **nuestra meta debería ser construir *"plataformas de desarrollo"[*]*** que deberían apoyarse –en lo posible- sobre herramientas sólidas y muy probadas, así no perder el objetivo primario y esencial que es *"desarrollar sistemas"*, no *"frameworks"* (y nuestros clientes agradecidos).

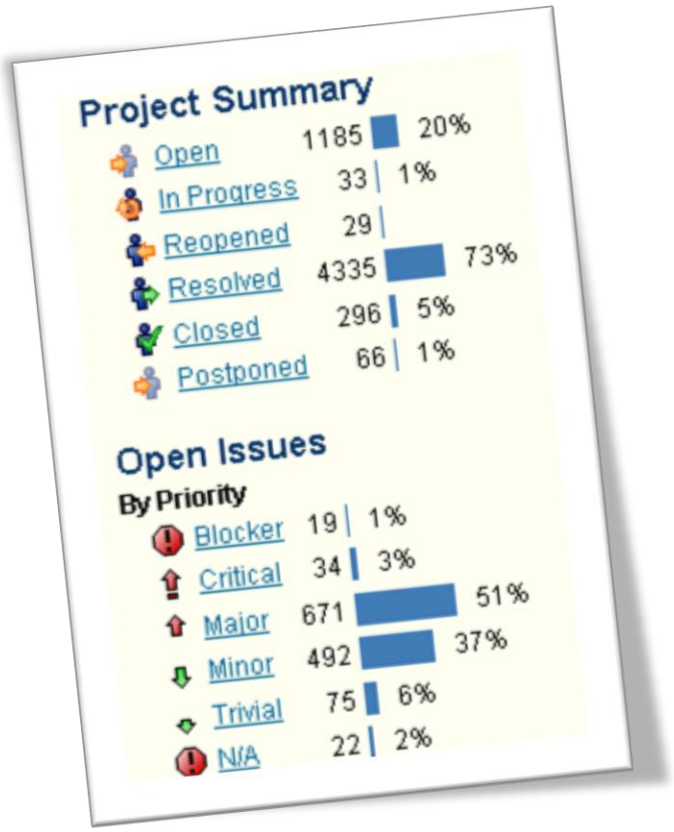
[*] Por "**plataforma de desarrollo"** nos referimos a lo opuesto de hacer todo "artesanalmente", todo lo que se hace desde la primera vez con "sudor y lágrimas", como un carpintero que trabaja sobre el trozo de madera hasta llegar a la pieza de arte que luego puede vender. **Deberíamos poner el foco en reducir todo lo posible cualquier actividad que no sea necesaria hacer dos veces**, reusar componentes a tal extremo que lo único que se hace "artesanalmente" son las funcionalidades nuevas que aún no están contempladas en nuestra "plataforma", pero que una vez hecho, no volveremos a desarrollar de cero, solo trabajar en su evolución.

Por ejemplo, ¿Cuántas veces repetimos código de persistencia de datos contra una base de datos? ¿Cuántas veces hicimos una clase de persistencia? ¿todos nuestros sistemas usan una única clase de persistencia o existen distintas versiones de distintas clases que hacen lo mismo o similar?

Una ventaja de ZF es que nos permite contar con componentes que se pueden usar dentro o fuera del sistema "MVC" permitiendo reusar el mismo código funcional (y conocimientos) en todos nuestros sistemas.

No tendremos sistemas aislados, tendremos sistemas que comparten la misma plataforma de desarrollo.

ZF es una herramienta que está extensamente probada y testeada, desarrollada y certificada por Zend Technologies y siendo usada en millones de aplicaciones web. Si desarrollamos "otra rueda" a través de mecanismos típicos de **"programación artesanal"**, ¿quién se hará cargo de los costos de desarrollar de cero funcionalidades completamente triviales?... sin contar que muy probablemente **por cada paso que avancemos iremos generando bugs** que convivarán mucho tiempo con nosotros? (hasta les tomaremos cariño y les pondremos apodos)



Entonces…

¿Para qué reinventar algo que existe (como una "rueda redonda") si se puede reusar partes para construir un vehículo que permita transportar pasajeros (y en menos tiempo)?

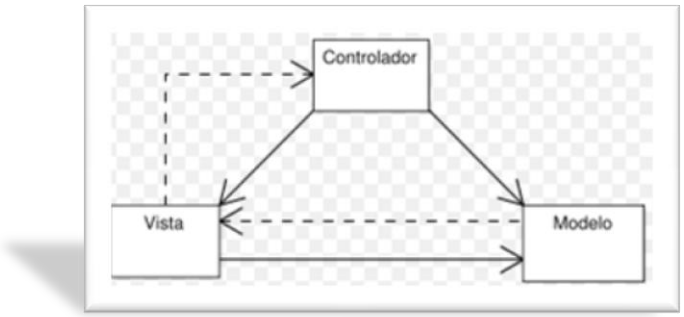
Anexo:

"¿La empresa cuenta con framework propio?"

Simplicidad

Zend Framework ha sido desarrollado pensando en la extrema simplicidad. Provee un ligero y de **bajo acoplamiento** conjunto de componentes simplificado para proporcionar las funcionalidades más comunes de los desarrolladores, entre estas podemos destacar las siguientes:

- **Separar las distintas capas con sus roles bien definidos**, implementando el patrón de arquitectura de software **MVC (Modelo, Vista y Controlador)**. Donde el **Modelo** es la representación específica de los datos con la cual el sistema opera (generalmente bases de datos, pero pueden ser otras fuentes de datos), la **Vista** presenta los datos en un formato adecuado para interactuar, usualmente cumpliendo con la "interfaz de usuario" y finalmente el **Controlador** que responde a eventos, generalmente peticiones de los usuarios (presionando un link, ejecutando un formulario, etc), e interactúa a su vez con el Modelo y la Vista (para pedirle información o para representar la información, respectivamente).



- **Trabajar con formularios y validaciones de datos enviados**: a través de componentes como Zend_Form, Zend_Filter y Zend_Filter_Input.
- **Abstraer la conexión y las operaciones en la bases de datos** a través de Zend_Db
- **Autenticación y seguridad**, como Zend_Acl y Zend_Auth
- **Emails**, Logs, etc, muchos componentes que nos resuelven todos los problemas habituales de cualquier sistema que intentemos desarrollar.

Lo último en tecnología y funcionalidades de la Web 2.0

- **AJAX implementado con Dojo (framework de javascript)** y apoyado con componentes Json (alternativamente cuenta con soporte para jQuery, menos compleja que Dojo)
- **Zend_Search_Lucene** - Motor Estándar de búsqueda Lucene
- **Zend_Feed (Sindicación)** - formatos de datos y fácil acceso a ellos en nuestras aplicaciones Web 2.0
- **Zend_Soap (web services)** y una larga lista de componentes para acceder a populares servicios como Twitter, Yahoo, Flickr, Delicious, Amazon, etc.
- **100% Orientado a objetos con PHP5**– siguiendo estándares y las mejores prácticas, patrones de diseño, pruebas unitarias, y un largo etcétera.

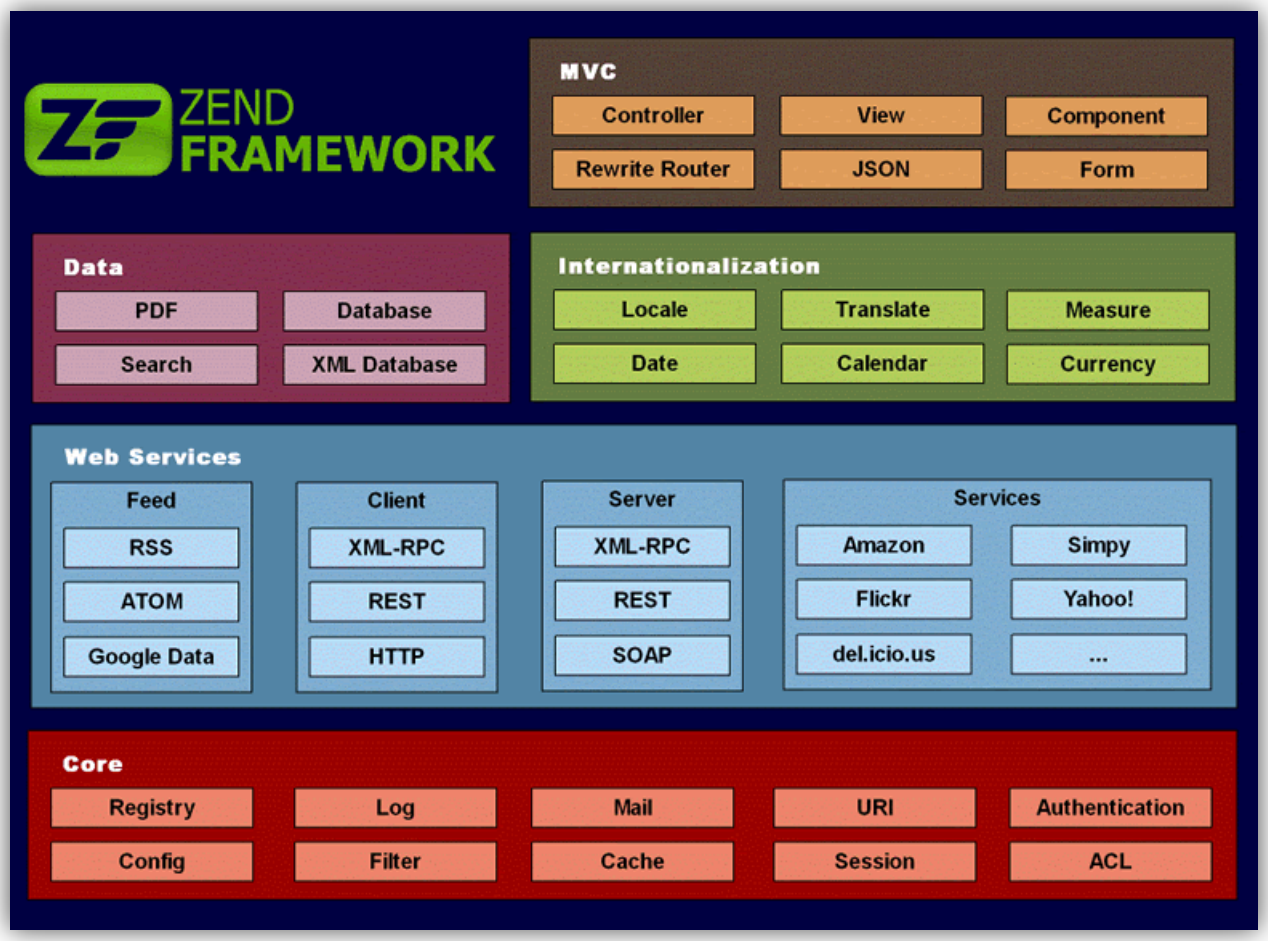
Aunque suene obvio y repetitivo, cabe destacar que **aún existen frameworks que usan PHP4**, no todos aprovechan correctamente la POO, y menos sus componentes son lo suficiente independientes como para evitar **una forma de trabajo rígida**, donde si queremos hacer algo que no estaba previsto, toda la productividad del framework se viene abajo.

Arquitectura

ZF tiene una **arquitectura flexible** que **permite a los desarrolladores usar tanto la estructura MVC y todos sus componentes (como originalmente está pensado), pero también permite tomar un subconjunto del Framework e incluirlo como una librería aislada**. Esto permite tener la posibilidad de armar una "plataforma homogénea" de desarrollo, donde **se podrá aplicar el patrón MVC para sistemas web tradicionales**, o usar –por ejemplo- las clases de persistencia para simples scripts que corran procesos contra la base de datos, evitando tener que crear código de persistencia cuando ya existe uno y puede aprovecharse sin problemas.

Con esto demuestra estar **perfectamente diseñado para que sus componentes estén altamente desacoplados** (si es importante destacarlo, no nos cansaremos de repetirlo ☺), es decir, hay poca o nada de dependencias entres los componentes.

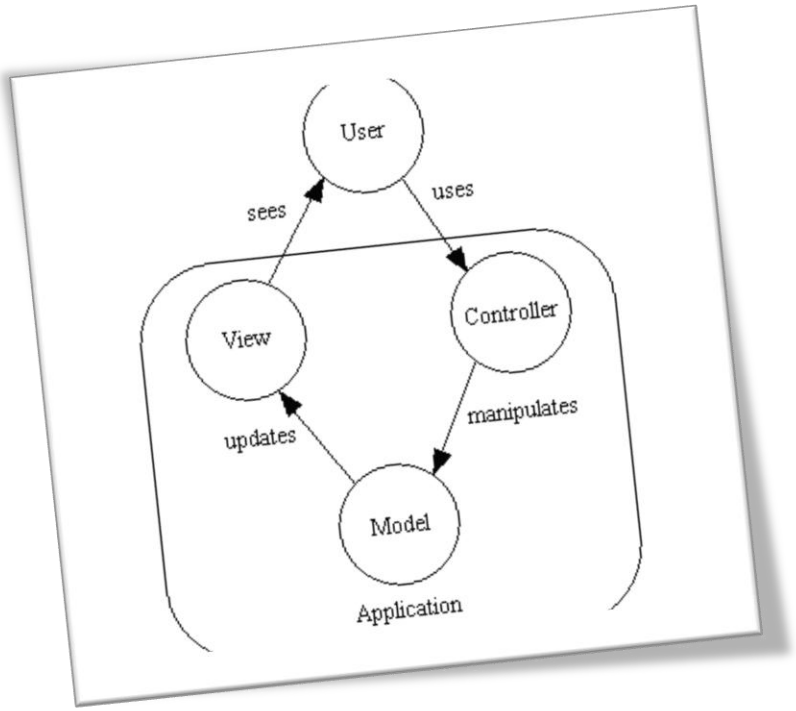
Uno de los primeros diagramas presentados por Zend donde muestran de forma general los componentes principales de la arquitectura del framework



Enlaces relacionados: gráfico con la dependencia entre componentes [1] [2] [3]

Diseño Interno

Entre los paradigmas o patrones de diseño implementados en ZF se destacan sin duda MVC, Registry y Table Gateway, los cuales son ampliamente usados en la mayoría de los proyectos.



- **MVC** como se mencionó anteriormente, nos permite separar las distintas capas de nuestra aplicación (no necesariamente es sinónimo de "3 capas", MVC es otra forma distinta de hacer una separación de responsabilidades),
- El patrón de diseño **Registry** nos permite almacenar objetos dentro de una contenedor y después contar con ellos en cualquier parte/momento dentro de nuestro sistema,
- Finalmente el patrón **Table Gateway** por cada tabla de la base de datos nos permite disponer de las funcionalidades y operaciones básicas como actualizar, crear, eliminar, listar y ver detalle.

Plenamente probado, seguro y confiable

ZF se prueba constantemente mediante técnicas de test unitario desde el principio, con estrictos requisitos en materia de calidad de código para asegurarse de que todo el código contribuido no sólo ha sido objeto testeado, además de ser estable y fácil de extender y de mantener. Con todo esto **buscan garantizar que podemos crear nuestras propias librerías o componentes** a partir de los existentes en Zend Framework y que no es por resultado de la improvisación.

La siguiente es **una lista de los componentes actuales** de Zend Framework, que si seguimos a través de actualizaciones a través de SVN veremos que todos los días se mejoran, corrigen o se agregan nuevas funcionalidades:

- Zend_Acl
- Zend_Amf
- Zend_Application
- Zend_Auth
- Zend_Cache
- Zend_Captcha
- Zend_CodeGenerator
- Zend_Config
- Zend_Config_Writer
- Zend_Console_Getopt
- Zend_Controller
- Zend_Currency
- Zend_Date
- Zend_Db
- Zend_Debug
- Zend_Dojo
- Zend_Dom
- Zend_Exception
- Zend_Feed
- Zend_File
- Zend_Filter
- Zend_Form
- Zend_Gdata
- Zend_Http
- Zend_InfoCard
- Zend_Json
- Zend_Layout
- Zend_Ldap
- Zend_Loader
- Zend_Locale
- Zend_Log
- Zend_Mail
- Zend_Measure
- Zend_Memory
- Zend_Mime
- Zend_Navigation
- Zend_OpenId
- Zend_Paginator
- Zend_Pdf
- Zend_ProgressBar
- Zend_Queue
- Zend_Reflection
- Zend_Registry
- Zend_Rest
- Zend_Search_Lucene
- Zend_Server
- Zend_Service
- Zend_Session
- Zend_Soap
- Zend_Tag
- Zend_Test
- Zend_Text
- Zend_TimeSync
- Zend_Tool_Framework
- Zend_Tool_Project
- Zend_Translate
- Zend_Uri
- Zend_Validate
- Zend_Version
- Zend_View
- Zend_Wildfire
- Zend_XmlRpc
- ZendX_Console_Process_Unix
- ZendX_JQuery

Tips y recordatorios PHP5 / POO

- **Seguir el estándar de codificación Zend y el uso de apertura de tags en las vistas (phtml), sólo en las vistas**, ejemplos: <?php if(condicion == true): ?> <?php foreach($this->usuarios as $usuario):?> <?php while($this->usr->valid()):?>.
- **En general los atributos y métodos private y protected siempre inician con "_"** (underscore).
- **En las clases el tag ?> de cierre no va** (Estándar Zend).
- **Los atributos son siempre protegidos o privados y se acceden/modifican mediante los métodos getter/setter.** Ej getNombre(). Uno de los principios básico de la POO: *"Principio de ocultación".*
- **El constructor en PHP5 se escribe como: __construct** y NO como el nombre de la clase.
- **Como estándar en POO, PHP5 y Zend los nombres de clase siempre comienzan con mayúscula, mientras que los nombre de variables y métodos comienzan en minúscula.** Cuando son nombres compuesto se debe separar con una letra Mayúscula en la primera letra de la palabra compuesta (*estilo de escritura CamelCase*), de esta forma el nombre de una clase podría ser *MiClaseDeUpperCamelCase* y de un método o variable *miMetodoDeLowerCamelCase*().
- **Tag de apertura en Zend y PHP5 se utiliza <?php y NO <?=.** EL Short tag no está permitido en el estándar.
- **El nombre del archivo de la clase se tiene que llamar de la misma manera que el nombre de la misma clase que la contiene**.
- **Siempre debe haber una clase por archivo**, como se dijo en el punto anterior, el nombre de este debe ser el mismo que de la clase.

Como complemento a los temas iníciales y a la tarea que se publicará próximamente, se recomienda leer los siguientes enlaces:

Repaso (si existieran dudas)

- Programación orientada a objetos - Wikipedia
- Zend Coding Standard
- PHP5 POO

Nuevos conceptos / herramientas

- Modelo Vista Controlador (MVC)
- ArrayObject Class
- Standard PHP Library (SPL)

En Resumen

ZF proporciona cada uno de los componentes para muchos otros requisitos comunes en el desarrollo de aplicaciones web, incluyendo la autenticación y autorización a través de listas de control de acceso (ACL), formularios, configuración, caché, filtro y validación de los datos proporcionados por el usuario para la seguridad y la integridad de los mismos, internacionalización, AJAX, correo electrónico, Lucene (formato de indexación y búsqueda de consulta), y todos los API de Google junto con muchos otros populares servicios web para facilitar la creación de proyectos Mashups (aplicación híbrida que integra otras aplicaciones web).

"Framework" significa reducción de costos, menos fallos y más productividad, particularmente **ZF nos ofrece flexibilidad** y un **"Framework de Bajo Nivel"** que nos posibilita a partir de los componentes existentes **construir herramientas de más "Alto Nivel".**

> ***"Si he llegado más lejos ha sido apoyado en los hombros de gigantes"***
>
> Isaac Newton (1642-1727)

Estos son factores importantes a la hora de decidir qué herramienta de adoptar para construir nuestra plataforma de desarrollo.

Fin.

Envía tus consultas a los foros!

Aquí es cuando debes sacarte todas las dudas haciendo consultas en los foros correspondientes