INTRODUCCIÓN A MVC CON PHP

PRIMERA PARTE

Esta es la primera parte de una serie de artículos introductorios al patrón de diseño MVC. En este artículo en particular podrás encontrar algunos ejemplos básicos en PHP con programación estructurada/funcional.

¿Qué es MVC?

MVC viene de **M**odel, **V**iew, **C**ontroller, o bien: **M**odelo, **V**ista y **C**ontrolador. Es un patrón de diseño que empecé a utilizar hace algún tiempo y la verdad es que me dió muy buenos resultados en los sistemas donde lo pude aplicar. La idea básica de éste patrón es separar nuestros sistemas en 3 capas, El Modelo, La Vista y el Controlador.

El Modelo se encarga de todo lo que tiene que ver con la persistencia de datos. Guarda y recupera la información del medio persistente que utilicemos, ya sea una base de datos, ficheros de texto, XML, etc.

La Vista presenta la información obtenida con el modelo de manera que el usuario la pueda visualizar.

El Controlador, dependiendo de la acción solicitada por el usuario, es el que pide al modelo la información necesaria e invoca a la plantilla(de la vista) que corresponda para que la información sea presentada.

Un pequeño ejemplo

1. Marcos entra a nuestro sitio mediante la URL **www.example.com/items/listar**.
2. Se carga el **Controlador** Items para ejecutar la **acción** de Listar.
3. El **controlador** solicita al **modelo** que le entregue un arreglo con todos los items que hay almacenados en la base de datos.
4. Una vez que posee dicha información le indica a la **vista** que va a utilizar la plantilla correspondiente al listado de items y le provee el arreglo con todos los usuarios.
5. La **vista**, por su parte, toma el arreglo de items y los muestra uno a uno en la plantilla que le indico el **controlador**.
6. Finalmente Marcos recibe el listado de items; lo observa un instante y decide que quiere agregar un nuevo item por lo que hace click en un enlace que lo lleva a la URL**www.example.com/items/agregar**.
7. Se repite el proceso desde el paso 1 pero con la nueva URL

Vamos al codigo

Para ir de a poco tomaré un ejemplo sencillo similar a los que utilice cuando hable de PHP Data Objects y lo iré separando en capas paso a paso. El ejemplo que voy a utilizar es el siguiente:

```
1  < ?php
2  require 'conexion.php';
3  $db = new PDO('mysql:host=' . $servidor . ';dbname=' . $bd, $usuario, $contrasenia);
4  $consulta = $db->prepare('SELECT * FROM items WHERE id_item = ? OR id_item = ?');
5  $consulta->execute(array(2, 4));
6  $items = $consulta->fetchAll();
7  $db = null;
8  ?>
9  < !DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
10         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
11
12 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
13 <head>
14         <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
15         <title>PDO - Jourmoly</title>
16 </head>
```

```
<body>
<table>
        <tr>
                <th>ID
                </th><th>Item
        </th></tr>
        < ?php
        foreach($items as $item)
        {
        ?>
        <tr>
                <td>< ?php echo $item['id_item']?></td>
                <td>< ?php echo $item['item']?></td>
        </tr>
        < ?php
        }
        ?>
</table>
<a href="index.php">Men&uacute;</a>
</body>
</html>
```

Ver Ejemplo

Nada del otro mundo, es un simple listado común presentado en una tabla HTML. Separaremos dicho ejemplo, por el momento, en 3 ficheros. Uno corresponderá al modelo, otro a la vista y el tercero será el controlador.

¿Cual es el modelo en este ejemplo?
Como mencione mas arriba, el modelo es el que se ocupa, básicamente, de todo lo que tiene que ver con el acceso a la información. Sin dudarlo, en este ejemplo PDO es quien cumple el papel de Modelo.

modelo.php

```
< ?php
$db = new PDO('mysql:host=' . $servidor . ';dbname=' . $bd, $usuario, $contrasenia);
$consulta = $db->prepare('SELECT * FROM items');
$consulta->execute();
$items = $consulta->fetchAll();
?>
```

¿Y cual es la vista?
La vista es quien representa la información para que el usuario la pueda entender, en este caso, el HTML, la tabla y todo lo usado para mostrar la información forma parte de la vista.

vista.php

```
< !DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
        <title>PDO - Jourmoly</title>
</head>
<body>
<table>
        <tr>
                <th>ID
                </th><th>Item
        </th></tr>
        < ?php
        foreach($items as $item)
        {
        ?>
```

```
        <tr>
                <td>< ?php echo $item['id_item']?></td>
                <td>< ?php echo $item['item']?></td>
        </tr>
        < ?php
        }
        ?>
</table>
</body>
</html>
```

¿Y el controlador?
El controlador es el que permite que todo funcione.

controlador.php

```
< ?php
//Se incluye el modelo
require 'modelo.php';

//En $items tenemos un arreglo con todos los items gracias al modelo

//Ahora la vista recibe dicho arreglo para mostrarlo por pantalla
require 'vista.php';
?>
```

Por último, tendremos un fichero mas *index.php* que lo único que hará es incluir algunas variables de configuración y nuestro controlador. Es decir, para ver el resultado del script entraremos por *index.php*

Ver ejemplo

Afinando nuestro ejemplo

El ejemplo anterior esta bien para un primer acercamiento, pero cuando trabajamos a diario las cosas no son tan sencillas como en este caso, una sola sección o elemento(items), una sola acción(listar), etc. Lo mas normal es que necesitemos de varios controladores y que cada controlador tenga varias acciones. A su vez, cada controlador puede utilizar uno o mas modelos como así también plantillas. Para lograr todo esto, es necesario que automaticemos un poco el primer ejemplo para que admita, en principio, varios controladores y acciones.

Como primera medida vamos a crear una estructura de ficheros para que que todo quede mas o menos ordenado, sencillo:

```
controladores/
.....itemsControlador.php
modelos/
.....itemsModelo.php
vistas/
.....listar.php
index.php
```

Donde listar.php equivale a vista.php de nuestro primer ejemplo. itemsModelo.php equivale a modelo.php con algunos cambios:

itemsModelo.php

```
< ?php
global $servidor, $bd, $usuario, $contrasenia;
$db = new PDO('mysql:host=' . $servidor . ';dbname=' . $bd, $usuario, $contrasenia);

function buscarTodosLosItems($db)
{
```

```
        $consulta = $db->prepare('SELECT * FROM items');
        $consulta->execute();
        return $consulta->fetchAll();
}
?>
```

e itemsControlador.php equivale a controlador.php también con algunos cambios:

itemsControlador.php

```
< ?php
function listar()
{
        //Incluye el modelo que corresponde
        require 'modelos/itemsModelo.php';

        //Le pide al modelo todos los items
        $items = buscarTodosLosItems($db);

        //Pasa a la vista toda la información que se desea representar
        require 'vistas/listar.php';
}
?>
```

Como verán los únicos cambios han sido armar los scripts con funciones, de modo que cada fichero pueda tener mas de una de ellas y puedan ser llamadas en cualquier momento e independientemente.

De ahora en mas, nuestro fichero index.php será quien se encargue de *averiguar* cual es el controlador y acción que busca el usuario, incluirá los archivos que sean necesarios y ejecutara la acción solicitada. Todos los accesos a nuestro sistema serán por medio de index.php y las URL serán similares a las siguientes:

www.example.com/index.php?controlador=**items**&accion=**listar**
www.example.com/index.php?controlador=**items**&accion=**agregar**
www.example.com/index.php?controlador=**items**&accion=**eliminar**

www.example.com/index.php?controlador=**usuarios**&accion=**listar**

Ahora solo nos queda hacer un pequeño script que interprete nuestra URL y llame al controlador y la acción que corresponda.

index.php

```
< ?php
//Primero algunas variables de configuracion
require 'conexion.php';

//La carpeta donde buscaremos los controladores
$carpetaControladores = "controladores/";

//Si no se indica un controlador, este es el controlador que se usará
$controladorPredefinido = "items";

//Si no se indica una accion, esta accion es la que se usará
$accionPredefinida = "listar";

if(! empty($_GET['controlador']))
      $controlador = $_GET['controlador'];
else
      $controlador = $controladorPredefinido;

if(! empty($_GET['accion']))
      $accion = $_GET['accion'];
else
      $accion = $accionPredefinida;
```

```php

//Ya tenemos el controlador y la accion

//Formamos el nombre del fichero que contiene nuestro controlador
$controlador = $carpetaControladores . $controlador . 'Controlador.php';

//Incluimos el controlador o detenemos todo si no existe
if(is_file($controlador))
	require_once $controlador;
else
	die('El controlador no existe - 404 not found');

//Llamamos la accion o detenemos todo si no existe
if(is_callable($accion))
	$accion();
else
	die('La accion no existe - 404 not found');
?>
```

Y ya lo podemos probar:

index.php?controlador=items&accion=listar

¿Y si ahora quiero insertar items?

Es muy sencillo, solo debemos agregar la accion de *agregar* a nuestro controlador.

itemsControlador.php

```php
function listar()
{
	//Incluye el modelo que corresponde
	require 'modelos/itemsModelo.php';

	//Le pide al modelo todos los items
	$items = buscarTodosLosItems($db);

	//Pasa a la vista toda la información que se desea representar
	require 'vistas/listar.php';
}

function agregar()
{
	echo 'Aqui incluiremos nuestro formulario para insertar items';

	require 'modelos/itemsModelo.php';

	if($_POST)
	{
		insertar();
	}

	  require 'vistas/agregar.php';
}
```

Desde luego que el modelo ahora también debería incluir una función insertar() y debería existir una plantilla agregar.php. Para ver nuestro formulario solo deberiamos ingresar por:

index.php?controlador=items&accion=agregar

No olvides que el action del formulario debe apuntar a www.tusitio.com/index.php?controlador=items&accion=agregar

¿Y si quiero agregar un listado de usuarios?

Para ello, solo debes crear un controlador usuariosControlador.php con una funcion listar() similar a la de itemsControlador, y obviamente, también debes crear las plantillas que creas necesarias o, por que no, reutilizar alguna que ya tengas.

index.php?controlador=usuarios&accion=listar

Notas finales

Y hasta aquí llega esta primera parte. Logramos implementar un script separado en 3 capas y dimos el primer paso con MVC usando programación estructurada/funcional. En el articulo que sigue mostraré esto mismo pero con programación orientada a objetos y algunas funcionalidades extras como aplicar URL amigables a un sistema de este tipo.

Bajar los ejemplos

Si querés retocar un poco el código, practicar, etc... podés bajarte todos los ejemplos desde aqui. Incluye un fichero .sql para que crees la tabla utilizada.

Lectura complementaria

Si mi explicación no te basto, podés leer la explicación de MVC en la wikipedia. Desde luego, Google también sabe algo sobre el tema.

SEGUNDA PARTE

Al fin aquí está la segunda parte del artículo que empecé hace algunos meses. Espero que les sea de utilidad.

En esta parte, como había comentado, mostraré una mini implementación de mvc basada en el ejemplo de la primera parte del artículo pero con programación orientada a objetos. No explicaré la utilización de clases, descuento que saben herencia y demás, solo me limitaré al mvc y poco mas.

Lecturas recomendadas antes de leer este artículo

Clases y objetos en PHP5
Singleton en wikipedia

Empecemos, ingredientes

La estructura de archivos se mantiene bastante con respecto a nuestro ejemplo anterior, pero ahora cada archivo es una Clase, salvo el index.php y el config.php.

index.php
Será la única entrada de nuestro sistema como en el ejemplo anterior, pero en este caso no realiza otra tarea mas que incluir e iniciar el *FrontController*.

```
1 < ?php
2 //Incluimos el FrontController
3 require 'libs/FrontController.php';
4 //Lo iniciamos con su método estático main.
5 FrontController::main();
6 ?>
```

libs/FrontController.php

El *FrontController* es el que recibe todas las peticiones, incluye algunos ficheros, busca el controlador y llama a la acción que corresponde.

```php
< ?php
class FrontController
{
	static function main()
	{
		//Incluimos algunas clases:

		require 'libs/Config.php'; //de configuracion
		require 'libs/SPDO.php'; //PDO con singleton
		require 'libs/View.php'; //Mini motor de plantillas

		require 'config.php'; //Archivo con configuraciones.

		//Con el objetivo de no repetir nombre de clases, nuestros controladores
		//terminaran todos en Controller. Por ej, la clase controladora Items, será
ItemsController

		//Formamos el nombre del Controlador o en su defecto, tomamos que es el
IndexController
		if(! empty($_GET['controlador']))
			$controllerName = $_GET['controlador'] . 'Controller';
		else
			$controllerName = "IndexController";

		//Lo mismo sucede con las acciones, si no hay accion, tomamos index como accion
		if(! empty($_GET['accion']))
			$actionName = $_GET['accion'];
		else
			$actionName = "index";

		$controllerPath = $config->get('controllersFolder') . $controllerName . '.php';

		//Incluimos el fichero que contiene nuestra clase controladora solicitada
		if(is_file($controllerPath))
			require $controllerPath;
		else
			die('El controlador no existe - 404 not found');

		//Si no existe la clase que buscamos y su acción, tiramos un error 404
		if (is_callable(array($controllerName, $actionName)) == false)
		{
			trigger_error ($controllerName . '->' . $actionName . '` no existe',
E_USER_NOTICE);
			return false;
		}
		//Si todo esta bien, creamos una instancia del controlador y llamamos a la accion
		$controller = new $controllerName();
		$controller->$actionName();
	}
}
?>
```

libs/View.php

Es una pequeña clase que hace de motor de plantilla, aunque con poquitas funcionalidades. Solo nos permite incluir una plantilla y asignarle variables.

```php
< ?php
class View
{
	function __construct()
	{
	}

	public function show($name, $vars = array())
	{
```

```
                //$name es el nombre de nuestra plantilla, por ej, listado.php
                //$vars es el contenedor de nuestras variables, es un arreglo del tipo llave =>
valor, opcional.

                //Traemos una instancia de nuestra clase de configuracion.
                $config = Config::singleton();

                //Armamos la ruta a la plantilla
                $path = $config->get('viewsFolder') . $name;

                //Si no existe el fichero en cuestion, tiramos un 404
                if (file_exists($path) == false)
                {
                        trigger_error ('Template `' . $path . '` does not exist.', E_USER_NOTICE);
                        return false;
                }

                //Si hay variables para asignar, las pasamos una a una.
                if(is_array($vars))
                {
                   foreach ($vars as $key => $value)
                   {
                $$key = $value;
                   }
                }

                //Finalmente, incluimos la plantilla.
                include($path);
        }
}
/*
 El uso es bastante sencillo:
 $vista = new View();
 $vista->show('listado.php', array("nombre" => "Juan"));
*/
?>
```

libs/SPDO.php

SPDO es una clase que extiende de *PDO*, su única ventaja es que nos permite aplicar el patron*Singleton* para mantener una única instancia de *PDO*.

```
< ?php
class SPDO extends PDO
{
        private static $instance = null;

        public function __construct()
        {
                $config = Config::singleton();
                parent::__construct('mysql:host=' . $config->get('dbhost') . ';dbname=' . $config-
>get('dbname'),
$config->get('dbuser'), $config->get('dbpass'));
        }

        public static function singleton()
        {
                if( self::$instance == null )
                {
                        self::$instance = new self();
                }
                return self::$instance;
        }
}
?>
```

Models/*Model.php

En el ejemplo anterior los modelos eran ficheros comunes con algunas funciones sueltas, en este caso son clases, y lo que antes eran funciones ahora son métodos. Al igual que en el primer ejemplo, usamos *PDO* (esta vez a traves de *SPDO*) para el acceso a datos.

```php
< ?php
class ItemsModel
{
        protected $db;

        public function __construct()
        {
                //Traemos la unica instancia de PDO
                $this->db = SPDO::singleton();
        }

        public function listadoTotal()
        {
                //realizamos la consulta de todos los items
                $consulta = $this->db->prepare('SELECT * FROM items');
                $consulta->execute();
                //devolvemos la colección para que la vista la presente.
                return $consulta;
        }
}
?>
```

Controllers/*Controller.php

Sucede lo mismo que con los modelos, ahora los controladores son Clases y las acciones son los métodos. La única diferencia con respecto al ejemplo anterior es el uso de la clase *View* para asignar variables y presentar la plantilla.

```php
< ?php
class ItemsController
{
        function __construct()
        {
            //Creamos una instancia de nuestro mini motor de plantillas
            $this->view = new View();
        }

        public function listar()
        {
                //Incluye el modelo que corresponde
                require 'models/ItemsModel.php';

                //Creamos una instancia de nuestro "modelo"
                $items = new ItemsModel();

                //Le pedimos al modelo todos los items
                $listado = $items->listadoTotal();

                //Pasamos a la vista toda la información que se desea representar
                $data['listado'] = $listado;

                //Finalmente presentamos nuestra plantilla
                $this->view->show("listar.php", $data);
        }

        public function agregar()
        {
                echo 'Aquí incluiremos nuestro formulario para insertar items';
        }
}
?>
```

Views/listar.php

Poco tengo para decir sobre las plantillas, son archivos .php comunes.

```php
< !DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
```

```
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
        <title>MVC - Modelo, Vista, Controlador - Jourmoly</title>
</head>
<body>
<table>
        <tr>
                <th>ID
                </th><th>Item
        </th></tr>
        < ?php
        // $listado es una variable asignada desde el controlador ItemsController.
        while($item = $listado->fetch())
        {
        ?>
        <tr>
                <td>< ?php echo $item['id_item']?></td>
                <td>< ?php echo $item['item']?></td>
        </tr>
        < ?php
        }
        ?>
</table>
</body>
</html>
```

libs/Config.php

Es una pequeña clase de configuración con un funcionamiento muy sencillo, implementa el patron singleton para mantener una única instancia y poder acceder a sus valores desde cualquier sitio.

```
< ?php
class Config
{
    private $vars;
    private static $instance;

    private function __construct()
    {
        $this->vars = array();
    }

    //Con set vamos guardando nuestras variables.
    public function set($name, $value)
    {
        if(!isset($this->vars[$name]))
        {
            $this->vars[$name] = $value;
        }
    }

    //Con get('nombre_de_la_variable') recuperamos un valor.
    public function get($name)
    {
        if(isset($this->vars[$name]))
        {
            return $this->vars[$name];
        }
    }

    public static function singleton()
    {
        if (!isset(self::$instance)) {
            $c = __CLASS__;
            self::$instance = new $c;
        }

        return self::$instance;
    }
}
```

```php
/*
 Uso:

 $config = Config::singleton();
 $config->set('nombre', 'Federico');
 echo $config->get('nombre');

 $config2 = Config::singleton();
 echo $config2->get('nombre');

*/
?>
```

config.php
Es el archivo de configuración, hace uso de una instancia de la clase *Config*.

```php
< ?php
$config = Config::singleton();

$config->set('controllersFolder', 'controllers/');
$config->set('modelsFolder', 'models/');
$config->set('viewsFolder', 'views/');

$config->set('dbhost', 'localhost');
$config->set('dbname', 'pruebas');
$config->set('dbuser', 'root');
$config->set('dbpass', '');
?>
```

¿Y como funciona todo esto?

Tomando como ejemplo la siguiente URL:

http://www.jourmoly.com.ar/ejemplos/mvc/ejemplo-poo/?controlador=Items&accion=listar

El recorrido detallado es el siguiente:

1. El usuario ingresa por el index.php, aqui se incluye el FrontController y se inicia nuestro sistema.

2. El FrontController incluye los ficheros basicos, averigua el controlador y la acción, incluye el controlador, crea una instancia del mismo y llama a la accion correspondiente. En este caso el controlador es ItemsController y la acción (método) es listar().

3. La acción listar() de ItemsController incluye el modelo que necesita (ItemsModel) y crea una instancia, solicita todos los datos y se los pasa a la instancia de la vista junto con el nombre de la plantilla a presentar (listar.php).

4. La vista incluye la plantilla y asigna las variable $listado.

5. El usuario recibe en pantalla el listado total.

Afinando el mini mvc

Para afinar un poquito nuestro sistema nos convendria hacer que nuestros controladores y modelos no sean clases base, sino que extiendan de otras clases que contengan las funcionalidades básicas. De este modo todas las funcionalidades que agreguemos a nuestro ControladorBase y ModeloBase seran heredadas por los controladores y modelos que utilicemos.

libs/ControllerBase.php

```php
< ?php
```

```php
abstract class ControllerBase {

    protected $view;

    function __construct()
    {
        $this->view = new View();
    }
}
?>
```

Como verán inclui en este controlador base la creación de la instancia de la vista en el constructor, es decir que ya no sera necesario hacer esto en los demas controladores. Deben recordar que si en los controladores sobreescriben el constructor, deben llamar al constructor de la clase base para poder tener la instancia de la vista (parent::__construct()).

Con esta clase base, nuestro controlador Items nos quedaria de la siguiente manera:

controllers/ItemsController.php

```php
< ?php
class ItemsController extends ControllerBase
{
        public function listar()
        {
                //Incluye el modelo que corresponde
                require 'models/ItemsModel.php';

                //Creamos una instancia de nuestro "modelo"
                $items = new ItemsModel();

                //Le pedimos al modelo todos los items
                $listado = $items->listadoTotal();

                //Pasamos a la vista toda la información que se desea representar
                $data['listado'] = $listado;

                //Finalmente presentamos nuestra plantilla
                $this->view->show("listar.php", $data);
        }

        public function agregar()
        {
                echo 'Aqui incluiremos nuestro formulario para insertar items';
        }
}
?>
```

libs/ModelBase.php

```php
< ?php
abstract class ModelBase
{
        protected $db;

        public function __construct()
        {
                $this->db = SPDO::singleton();
        }
}
?>
```

Similar a ControllerBase, nos permitira ahorrarnos el paso de iniciar PDO en cada modelo. Esta clase podria tener muchísimas funcionalidades mas y podria ahorrarnos escribir muchísimo código SQL si implementáramos en ella el patron Active Table por ejemplo, pero eso es otra historia.

libs/ItemsModel.php

```php
< ?php
class ItemsModel extends ModelBase
{
        public function listadoTotal()
        {
                //realizamos la consulta de todos los items
                $consulta = $this->db->prepare('SELECT * FROM items');
                $consulta->execute();
                //devolvemos la coleccion para que la vista la presente.
                return $consulta;
        }
}
?>
```

El último cambio está en el *FrontController* pero es mínimo, solo es la inclusión de los archivos que contienen las clases Base para que puedan ser heredadas.

Aclaraciones finales

1. Esta es una forma sencilla de implementar mvc con poo, no quiere decir que se ala única ni la mejor, hay algunas variantes pero me parecio bastante mas sencilla de comprender de esta manera.

2. Cuando me refiero a motor de plantillas y plantillas no lo tomen tan literal, fijense que las plantillas son simples .php donde igual se puede incluir codigo php de cualquier tipo. Solo es una manera sencilla y liviana, sin tener que usar Smarty u otro motor de plantillas.

3. Abierto a sugerencias :p

Links de interes

1. Building a simple MVC system with PHP5: Implementa una clase Router y otra Registry, aunque en ingles, esta bastante claro.

2. Todos los archivos del primer ejemplo.

3. Todos los archivos del segundo ejemplo.

4. Google

http://www.jourmoly.com.ar/introduccion-a-mvc-con-php-primera-parte/

http://www.jourmoly.com.ar/introduccion-a-mvc-con-php-segunda-parte/