Arquitectura MVC (con PHP)

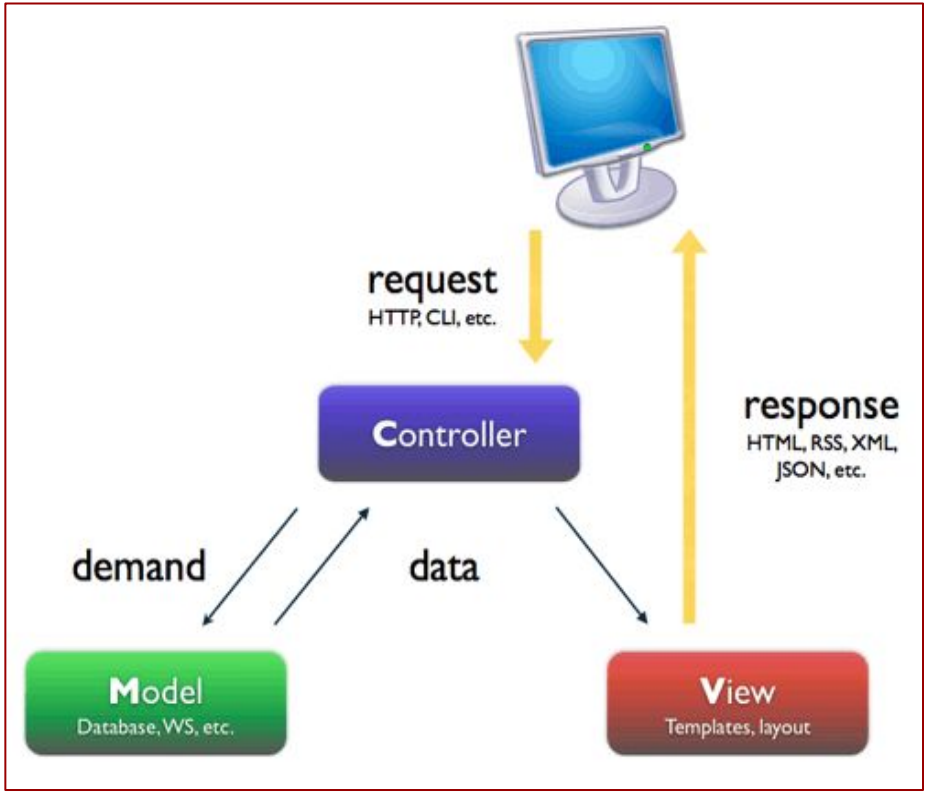
Realizado por A.Garay (dpto. de Informática)

Conceptos

- La arquitectura MVC se refiere a una forma o estilo de programación en la que prima la separación de las ***responsabilidades*** de una aplicación en tres áreas diferenciadas
  - **Modelo** (M ó Model)
    - Se encarga de la gestión de la lógica de negocio y la persistencia de datos
  - **Vista** (V ó View)
    - Se encarga de la presentación de los datos (interfaz de usuario) y la gestión de eventos para desencadenar acciones en el controlador (listeners en UI java, javascript en web)
  - **Controlador** (C ó Controller)
    - Se encarga de gestionar todas las peticiones que vienen desde la interfaz, de acceder al modelo, y transmitir estos datos a la vista

Conceptos

- El modelo y la vista son elementos “pasivos”. Actúan a demanda por parte del controlador.
- Ventajas
  - Modularidad.
  - Independencia del mecanismo de persistencia (cambios en el modelo sin miedo)
  - Independencia de la representación de datos (cambios en las vistas sin miedo, varias interfaces, etc.)
  - Facilidad de mantenimiento
  - Escalabilidad
  - Inteligibilidad.
- En algunos casos (no en ap.web), el modelo podría comunicar datos directamente a la vista utilizando un patrón Observer.



El problema (1/2)

```php
<?php
// Conexión BD
$db = new PDO ( "mysql:host=localhost;dbname=test", "root", "" );
// Query SQL
$resultado = $db->query ( "select id,nom from empleados" );

// Mostrando el resultado
echo <<<HTML
<h1>Lista de empleados</h1>
<table><tr><th>Id</th><th>Nombre</th></tr>
HTML;
foreach ( $resultado as $fila ) {
    echo "<tr><td> {$fila['id']} </td> <td> {$fila['nom']} </td> </tr>";
}
echo '</table>';

// Cierre conexión BD
$db = null;
?>
```

El problema (2/2)

- El código anterior es compacto, rápido y funciona, pero adolece de varios problemas
  - Hay mezclados muchos tipos de responsabilidades distintas: conexión a la BD, ejecución de queries, lógica de negocio, presentación de la página, y porque es un ejemplo muy sencillo no hay más (control de formato de datos, control de lógica de workflow, posibles redirecciones, cuáles son las pantallas siguientes, gestión de errores y excepciones, logging, etc.)
  - Hay mezclados muchos lenguajes informáticos (código “macarroni”): PHP, HTML, SQL, y podría ser incluso peor (JavaScript, XML, jquery, json...)
- MVC viene a poner un poco de orden en el diseño de arquitecturas y responsabilidades para los componentes de aplicaciones informáticas en general, y de aplicaciones web en particular. En este sencillo ejemplo, lo iríamos arreglando de la siguiente manera...

La solución (1/2): separar la vista

- Crear un script **view.php** en el que extraeremos todo lo que tiene que ver con presentación
  - HTML
  - javascript
  - PHP (escueto)
  - etc.
- El resto del código quedará en **resto.php**
- Como es lógico habrá que pasarle a **view.php** datos desde **resto.php** que es el que sabe qué datos hay que "pintar".
- Intentaremos quitar (en **view.php**) el máximo de lógica y código PHP, utilizando marcas más escuetas y evitando "echos"

resto.php

```php
<?php
// Conexión BD
$db = new PDO ( "mysql:host=localhost;dbname=test", "root", "" );
// Query SQL
$resultado = $db->query("select id,nom from empleados");

// Cierre conexión
$db = null;

//Mostrando el resultado
require_once ('view.php');
?>
```

view.php

```php
<h1>Lista de empleados</h1>
<table>
    <tr>
        <th>Id</th>
        <th>Nombre</th>
    </tr>
<?php foreach ( $resultado as $fila ) : ?>
    <tr>
        <td><?= $fila['id'] ?></td>
        <td><?= $fila['nom'] ?></td>
    </tr>
<?php endforeach;?>
</table>
```

La solución (2/2) separar el modelo

- Crear un script **model.php** en el que extraeremos todo lo que tiene que ver con acceso a datos y lógica de negocio.
- El código que nos queda, sería el del **controller.php**, que en este sencillo ejemplo lo único que haría es invocar al ***model*** para obtener los datos y pasárselos al ***view*** para que los “pinte”,

controller.php

```php
<?php
//Obtener datos del modelo
require_once ('model.php');

//Preparar los datos para la vista
$resultado=getEmpleados();

//Mostrar el resultado con la vista
require_once ('view.php');
?>
```

model.php

```php
<?php
function getEmpleados() {
// Conexión BD
$db = new PDO ( "mysql:host=localhost;dbname=test", "root", "" );
// Query SQL
$resultado = $db->query("select id,nom from empleados");

// Cierre conexión
$db = null;

return $resultado;
}
?>
```

Implementación de MVC

- Las ideas anteriormente expuestas se pueden implementar con más o menos complejidad.
- Las implementaciones más sencillas harán que ejemplos sencillos se hagan bien, pero que sea difícil escalar y reutilizar código.
- Las más complejas, hacen que sea más difícil la programación inicial, pero luego mucho más sencillo crear nuevos controladores, vistas y modelos.
- Otra buena opción es no realizar esta labor de “fontanería” previa y dejar que otros la programen por nosotros en los llamados “frameworks MVC” (Symfony, CodeIgniter, Laravel, CakePHP, etc.).
    - ventaja: desarrollos rápidos, más robustos, más metódicos.
    - inconveniente: hay que aprender a manejar el framework (archivos XML de configuración, plantillas, etc.) para sacarle partido, y la curva de aprendizaje inicial es más lenta.

El controlador

- Habrá normalmente un ligero controlador de fachada y varios subcontroladores: uno por cada entidad, por caso de uso o por funcionalidad.
- Pueden realizar varias acciones, la acción por defecto se suele llamar “index” (no tiene nada que ver con la página index.html o index.php)
- Sólo deben contener PHP (no HTML, no SQL, no JavaScript)
- Son los únicos que manejan sesiones, cookies y cabeceras.
- El controlador de fachada debe ser el punto único de entrada a nuestra aplicación (suele ser el fichero index.php bajo el DocRoot)
- Como el número de operaciones que se pueden hacer en una aplicación es potencialmente muy grande, en la práctica habrá un controlador frontal que derivará las peticiones realizadas a otros subcontroladores.
- NO DEBEN CONTENER LA INSTRUCCIÓN **ECHO**

El modelo

- Suele parecerse a un patrón DAO clásico, aunque conceptualmente es una representación abstracta de unos datos
- Iremos implementando los métodos DAO que vayamos necesitando sobre la marcha.
- Un método de modelo nunca consultará $_POST, $_GET, $_SESSION e intentará devolver los datos empaquetados en un formato que le resulte útil al controlador

La vista

- Debería contener HTML (JavaScript, CSS, etc.) principalmente
- Intercalar sólo PHP, si se puede, en las formas:
  - <?php foreach(...): ?> … <?php endforeach; ?>
  - <?php if(...): ?>...<?php else: ?>...<?php endif; ?>
  - <?= $expresión ?>
- Debería ser inteligible para alguien que supiera HTML/javascript, y poco o nada de PHP
- Recibirá del controlador los datos del modelo “empaquetados” en un array asociativo que contiene toda la información variable a mostrar.

Mapeo URI (1/2)

- Para que el controlador frontal (normalmente implementado en el index.php del DocRoot) sepa qué subcontrolador y qué acción realizar, una primera opción sería proporcionársela vía parámetro GET ó POST,
  - http://sitio.com/docRoot/index.php?tarea=empleadoCrear (peor)
  - http://sitio.com/docRoot/index.php?subcontrolador=empleado&accion=crear (mejor)
- *index.php* delegará en *controladorEmpleadoCrear.php*
  - switch($_REQUEST[‘tarea’]) {

    case ‘empleadoCrear’:header(‘Location:controladorEmpleadoCrear.php’); }
- Inconvenientes
  - El número de acciones de fachada para una aplicación puede ser muy grande y habrán múltiples controladores “colgando” todos del *DocumentRoot* de la aplicación, sin ningún tipo de orden ni estructura, para escalar tendríamos que añadir “case’s” a un “switch”
  - Puede ser desconcertante una redirección y me obliga a pasarle al otro script datos vía sesión o algún otro método.

Mapeo URI (2/2)

- Una forma más elegante de indicar qué controlador/acción queremos es mediante reescritura de URI en lugar de vía GET, dejando esta vía abierta, así como la vía POST para el paso de parámetros habituales.
- Utilizando un sencillo fichero .htaccess ubicado en el documentRoot se pueden indicar los controladores/acciones mediante "/" de directorio.

```
RewriteEngine On
RewriteRule . index.php
```

- La URI que utilizaremos será del estilo http://sitio.com/docRoot/empleado/crear
- A partir de aquí podremos "trocear" fácilmente la URI, valiéndonos de la función "explode()", y de las variables superglobales **$_SERVER ['REQUEST_URI']** y **$_SERVER ['SCRIPT_NAME']**
- Para evitar este comportamiento de reescritura en determinados directorios (p.ej. "img", "css") bastaría con ubicar ahí un **.htaccess** que contenga "RewriteEngine Off"