Colegio Nacional de E.M.D. “Asunción Escalada”

Expo CNAE

2MVC Framework

Categoría: Nivel Medio

Área: Ciencias de la Computación

Curso: 2º 3ª B.T.I.

Integrante:

Getulio Valentin Sánchez Ozuna

Tutor:

Lic. Pedro Núñez

Asunción – Paraguay

2013

Índice

Introducción

A nivel profesional sabemos que cuando un cliente requiere nuestros servicios ya sea para crear un sitio o una aplicación web, debemos tener una reunión personal con el cliente que nos permita conocer en profundidad sus necesidades, deseos y esperanzas sobre su proyecto web.

Pero como es de esperarse, el cliente siempre querrá un proyecto con una buena y agradable interfaz gráfica y una funcionalidad excelente que cumpla sus expectativas. Pero, en la realización de un proyecto web, el trabajo se divide en dos secciones importantes: el diseño y la programación.

Los frameworks PHP actuales facilitan la realización de la programación, incluyendo librerias que pueden ser reutilizados en muchos proyectos, pero el problema continua ya que la lógica aún se sigue mezclando con el diseño en un archivo HTML, esto resulta confuso para los diseñadores, y un poco molestoso para los programadores.

2MVC Framework se encarga de solucionar este problema, aportando librerias, que separan estas dos etapas, basándose en un "*diccionario*". 2MVC Framework también incluye otras facilidades como el acceso a la base datos, el manejo de niveles de sesiones, entre otras.

Problemática

El hecho de que las dos etapas de la elaboración web se mezclen en un mismo archivo, hace que la realización del proyecto se relentice ya que el programador debe esperar a que el diseñador termine su trabajo para que el pueda tomar la posta o viceversa, además de que complica que el proyecto pueda ser mantenido y ampliado sin muchas dificultades.

Pregunta

¿Existe alguna forma de hacer que estos trabajos sean completamente independientes y que permita mantener y extender las funcionalidades de un proyecto web?

Hipótesis

El hecho de incorporar un algoritmo que reemplace el código embebido por una linea de texto simple encerrado en llaves y siguiendo ciertos “estándares” solucionaría el problema de tener que mezclar el trabajo del programador y el diseñador y que a la vez permita la ampliación y el mantenimiento del proyecto sin muchas complicaciones.

Objetivos

Objetivo General

- Proveer de un Framework que permita el trabajo por separado del diseñador y el programador

Objetivos Específicos

- Suministrar librerias que facilitan el trabajo del programador
- Establecer clases css genéricos para permitir la maquetación más rápida y con diseño web adaptativo.
- Proponer el desarrollo utilizando un patrón arquitectónica que ayuda a mantener y ampliar un proyecto.

Resumen del Proyecto

2MVC Framework provee librerias que facilitan el trabajo tanto del programador como del diseñador web, además de ayudar a separar estos trabajos y de esta manera utilizar un servicio de SVC (como github) de manera más cómoda.

Metodología

La creación de 2MVC Framework se basó más en los resultados que se obtenían durante su elaboración, haciendo que el proyecto pueda mejorar continuamente hasta llegar a las soluciones que hasta el día de hoy presta, aprendiendo e incorporando herramientas similares que poseen otros frameworks.

Sección 1:

Aclaraciones

¿Qué es un framework?

Es un marco de aplicación o conjunto de bibliotecas orientadas a la reutilización a muy gran escala de componentes de software para el desarrollo rápido de aplicaciones.
Siendo muy simple, es un esquema (un esqueleto, un patrón) para el desarrollo y/o la implementación de una aplicación. Aunque el concepto también abarca al detalle de definir los nombres de ficheros, su estructura, las convenciones de programación, etc.

Las principales ventajas de la utilización de un framework son:

1. El desarrollo rápido de aplicaciones. Los componentes incluidos en un framework constituyen una capa que libera al programador de la escritura de código de bajo nivel.

2. La reutilización de componentes software al por mayor. Los frameworks son los paradigmas de la reutilización.

3. El uso y la programación de componentes que siguen una política de diseño uniforme. Un framework orientado a objetos logra que los componentes sean clases que pertenezcan a una gran jerarquía de clases, lo que resulta en bibliotecas más fáciles de aprender a usar.

¿Qué es un patrón arquitectónico?

Es un nivel en la cual la arquitectura de software:

Define la estructura básica de un sistema, pudiendo estar *relacionado con otros patrones*.

Representa una plantilla de construcción que provee un conjunto de subsistemas aportando las normas para su organización.

Patrón arquitectónico MVC

El patrón MVC (Model – View – Controller. En español: Modelo – Vista – Controlador) es un **patrón de arquitectura de software encargado de separar la lógica de negocio de la interfaz del usuario** y es más utilizado en aplicaciones Web, ya que facilita la funcionalidad, mantenibilidad y escabilidad del sistema, de forma simple y sencilla.

MVC divide las aplicaciones en tres niveles de abstracción:

Modelo: representa la lógica de negocios. Es el encargado de accesar de forma directa a los datos actuando como “intermediario” con la base de datos.

Vista: es la encargada de mostrar la información al usuario de forma gráfica y “humanamente legible”

Controlador: es el intermediario entre la vista y el model. Es quien controla las interacciones del usuario solicitando los datos al modelo y entregándolos a la vista para que ésta, lo presente al usuario, de forma “humanamente legible”.

Pero los frameworks mas conocidos que utilizan este patrón aún siguen necesitando que el archivo que posee la estructura del sitio, contenga códigos de distintos lenguajes.

Sección 2:

Librerias

El framework actualmente incluye librerias que forman parte del “core” y otras que son añadidas como plugins las cuales se encuentran en la carpeta lib de la raiz de los proyectos.

Librerias Core

Son las librerias que para 2MVC Framework son las necesarias para la creación de todo proyecto web.

connecting.phpclass: gestiona la conexión con la basede datos para permitir las consultas que obtendrán los datos.

work.phpclass: proporiciona funciones que facilitan la obtención, guardado y borrado de datos en la base de datos utilizando la función correcta y proporcionando los parametros necesarios.

index.phpclass: librería que posee la clase del cual se construirá el objeto que gestionará los controladores que a la vez llamarán a las vistas de la página con el que el usuario interactuará.

dictionary.phpclass: principal librería que implementa el algoritmo para separar las secciones de la elaboración de un proyecto web.

Librerias Plugins

Son aquellas que no son comunes entre todos los proyectos webs pero se pueden ir agregando a medida de que su utilización sea necesaria.

login.phpclass: librería que se encarga de gestionar los niveles de sesiones.

simpleImage.phpclass: provee funciones que permiten el trabajo con imágenes de una manera sencilla.

Sección 3:

Estructura de documentos y nombre de archivos

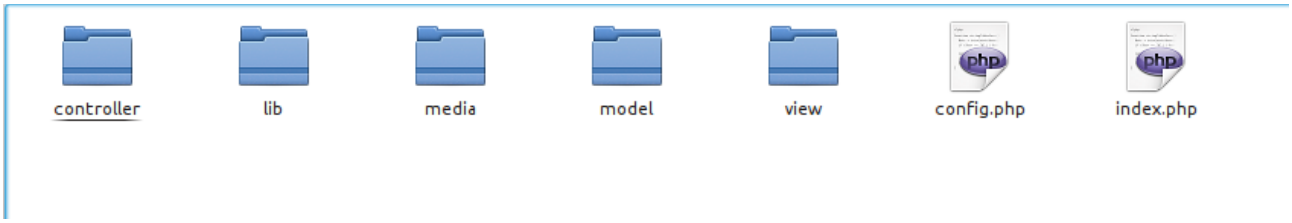

Figura 3.1 | Estructura raiz

Como observamos en la figura 3.1 la estructura principal se basa en 4 carpetas y 2 archivos.

Archivo *Config.php*

```php
<?php

    require_once 'lib/connecting.phpclass';
    require_once 'lib/work.phpclass';
    require_once 'lib/index.phpclass';
    require_once 'lib/dictionary.phpclass';

    //require_once 'lib/login.phpclass';
    //require_once 'lib/simpleImage.phpclass';

?>
```

Simplemente incluye las librerias core y permite habilitar las librerias plugins con solo borrar las barras de comentario

Figura 3.2 | Contenido del archivo config.php

Archivo *index.php*

```php
<?php

    require_once 'config.php';

//************************************************************************
//*********************** VARIABLES GENERALES ****************************
//************************************************************************

    $headTemplate = file_get_contents( 'media/html/includes/head.html' );

    $headerTemplate = file_get_contents( 'media/html/includes/header.html' );

    $footerTemplate = file_get_contents( 'media/html/includes/footer.html' );


    $index = new Index( $headTemplate, $headerTemplate, $footerTemplate );

    $dictionary = new Dictionary( $index );//pasa el objeto index por referencia

    $urlController = $index->getMVC( 'action', 'index', 'error' );//obtiene el co


    require_once $urlController;

?>
```

Figura 3.3 | Contenido del archivo index.php

Es el archivo en donde todas la páginas serán dibujadas gracias a la instrucción de la *linea 24* que concadenará llamadas a los archivos que poseen la información del archivo solicitado.

El framework propone este contenido, ya que esta estrictamente relacionado con las librerias core, y posee las variables que se utilizarán en toda la elaboración del sitio y además hará que el programador solamente empiece a hacer su trabajo con el patrón que 2MVC Framework propone sin preocuparse por este archivo. Pero algunas librerias plugins como login.phpclass pueden requerir añadir algunas líneas de códigos más.

En la carpeta controller se guardan los controladores, en model los modelos (de las cuales sus funciones fueron descriptas anteriormente) en la carpeta lib se almacenan las librerias y en view, se almacenan los archivos de la vista pero la parte lógica, en cuanto a términos adoptados en el framework, en ella se cargan los *words* que compondran al *dictionary* de la página que le pertenece, esto es fundamental en el algoritmo de 2MVC Framework.

Los anteriores directorios son destinados para el trabajo del desarrollador mientras que la carpeta *media* es el directorio de trabajo del diseñador. Veamos que hay dentro en la figura 3.4

Figura 3.4 | Directorio media

Los nombres de las carpetas son muy descriptivos y muy utilizados a nivel internacional. Con una carpeta llamada *css* que almacena las hojas de estilos que el diseñador proponga como el que el framework proporciona.

La carpeta *font* guardará las fuentes de letras que se utilizan en el sitio, por lo que no será necesario que el usuario lo tenga instalado en su ordenador para poder visualizar las páginas con la tipografía asignada.

En *images* se guardan las imágenes y en *js* los documentos javascript.

Se pueden incluir mas directorios en caso de necesitarlas como podrian ser una que guarde videos u uno para audios, entre otros.

En tanto la carpeta *html* vuelve a tener directorios que cumplen un rol en la estructura de desarrollo.

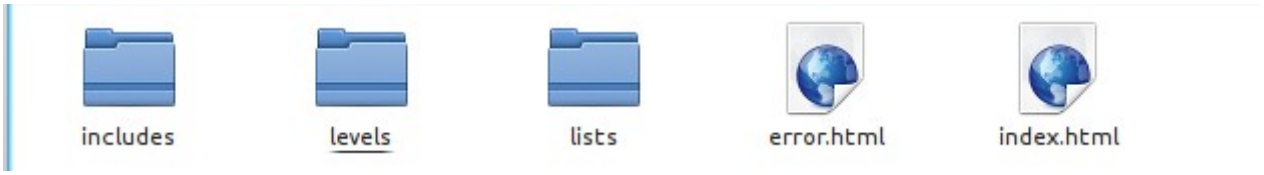

Figura 3.5 | Estructura del directorio html

El directorio *html* (figura 3.5) propone 3 carpetas y 2 archivos desde el inicio del proyecto a realizar. El archivo *index.html* es el archivo de inicio (la home del sitio web) mientras que *error.html* es la que se cargará en caso de ocurrir errores de documento.

Todas las plantillas html se deben guardar en este directorio, por ejemplo creamos una página a la cual llamamos *cursos*, para debemos guardar la plantilla correspondiente como *cursos.html*.

En *includes*, se guardan las plantillas que son comunes en todas las páginas del sitio, como el *head.hml*, el *header.html* y el *footer.html*. Además se pueden agregar otros de acuerdo a las necesidades del proyecto.

En *levels* se guardan los templates que variaran la visualización que el usuario tiene en una página de acuerdo a su nivel dentro del sistema por ejemplo podemos tener un sitio donde tenemos:

Nivel 0: Usuario no registrado o identificado
Nivel 1: Usuario registrado e identificado
Nivel 2: Moderador
Nivel 3: Administrador

Cada uno tiene sus privilegios o sus restricciones las cuales se manejan en la carpeta *levels*

En la carpeta *lists* se guardan la estructura en que se mostrarán los datos obtenidos dinámicamente por ejemplo de una base de datos.

En cuanto al nombre de los archivos es importante mencionar que si el nombre del controlador es *cursosController.php*, el nombre de la vista lógica debe ser *cursosView.php* y el template correspondiente debe llamarse *cursos.html*. Esto es obligatorio para el funcionamiento

Los finales **Controller.php* y **View.php* son obligatorios ya que **2MVC Framework** también busca que se sea lo más específico posible con lo que los archivos hacen dentro del proyecto.

Sección 4:

Ejemplo de uso

Antes de adentrarnos a la manera correcta de utilizar el framework hablemos un poco sobre como se harían las cosas si usarlo.

```php
<?php

	require_once 'connecting.php';

	$mysqli = Connecting::startConnection();

	$query = 'SELECT * FROM Productos';

	$results = $mysqli->query( $query );

	$query = 'SELECT titulo FROM Titulos WHERE idTitulo='.rand( 1, 4 );

	$titulo = $mysqli->query( $query );

	$titulo = $titulo->fetch_assoc();

?>

<!doctype html>
<html lang="es">
	<head>

		<?php

			require_once 'includes/head.html';

		?>
		<meta charset="UTF-8">
		<link rel="stylesheet" href="style.css" />
		<title><?php echo $titulo[ 'titulo' ]; ?></title>
	</head>
	<body>

		<div id="content">

			<?php
				require_once 'includes/header.html';
			?>

			<table>
				<tr>
					<td>Nombre</td>
					<td>Precio</td>
				</tr>

				<?php

					for( $i = 0; $reg = $results->fetch_assoc(); $i++ ) {

				?>

				<tr id="tr<?php echo $i ?>">
					<td><?php echo $reg[ 'nombreProducto' ] ?></td>
					<td><?php echo $reg[ 'precio' ] ?></td>
				</tr>

				<?php

					}//end for

				?>
			</table>

			<script src="https://gist.github.com/GVSO/7962284ad14763dd24a6.js"></script>

		</div>

		<?php
			require_once 'includes/footer.html';
		?>

	</body>
</html>
```

Figura 4.1 | Ejemplo que muestra como se mostrarían datos sin usar el framework

Como se observa en la Figura 4.1 el uso **embebido** de scripts PHP en el documento HTML pueden resultar confusos para el diseñador y molestoso para el programador a la hora de mantener y ampliar el sistema.

Además tiene el inconveniente de que si se crea otra plantilla HTML en una carpeta distinta los require_once a las estructuras constantes (head, header y footer) deberían cambiar, ya que el directorio cambiaría y para obtener los archivos multimedias, los estilos y los scripts del lado del

cliente que estos puedan requerir se deben adaptar nuevamente generando un gran problema.

Si bien con 2MVC Framework se deben crear 4 archivos para hacer esto (uno de ellos no es mostrado ya que es solo el controller que en este ejemplo lo único que hace es requerir indexView y crear una instancia de la clase), no generará ninguno de los inconvenientes del anterior ejemplo ya que el algoritmo soluciona este problema.

Para obtener el mismo resultado debemos hacer lo siguiente.

index.html HTML

```
<!DOCTYPE html>
<html lang="es">
        <head>
                <meta charset="UTF-8">
                {HEAD}
        </head>
        <body>

                <div id="content">
                        {HEADER}
                        <table>
                                <tr>
                                        <td>Nombre</td>
                                        <td>Precio</td>
                                </tr>
                                {Listado Productos}
                        </table>

                </div>

                {FOOTER}

        </body>
</html>
```

Figura 4.2 | Estructura de la plantilla principal

Es evidente lo cómodo que resulta para el diseñador esto, pues solo hay código que el si puede comprender. Las frases encerradas en llaves son lo que en términos del framework llamamos *words* y los que van en mayúsculas (HEAD, HEADER, FOOTER) son normas que se establecen para los *words* que representan zonas genéricas en todas las páginas.

Pero también se encuentra el *word* ***Listado Productos***, que es un nombre asignado por el diseñador a la estructura en donde se mostraran el listado (en este caso de productos) obtenidos dinámicamente desde una base de datos. Pero para establecer la manera en que se debe estructurar el listado se debe crear otra plantilla y guardarlo en la carpeta *lists*. Por ejemplo

```
<tr id="tr{Id Fila}">
        <td>{Nombre Producto}</td>
        <td>{Precio Producto}</td>
</tr>
```

productosList.html hosted with ❤ by GitHub view raw

Figura 4.3 | Estructura de la lista

Es la estructura de la lista, aquí observamos que se decidió agregar 3 *words* en la plantilla.

Con estos dos archivos nos damos cuenta que el trabajo del diseñador fue adaptado sin que tenga que experimentar en mundos desconocidos.

Ahora, veamos la como agregar los *words* al *dictionary* y traducirlos, mostrando al usuario la plantilla resultante.

```php
<?php

	class IndexView {
		private $listaProductosTraducida;
		private $templateList;

		private $list;

		private function traducirListaProductos() {
			global $dictionary;

			$this->templateList = file_get_contents( 'media/html/lists/productosList.html' );

			//Obtiene los productos
			$productos = Work::getRegisters( 'Productos' );

			$i = 0;
			//Por cada producto
			foreach ( $productos as $producto ) {

				$this->lista[] = array(
							'Nombre Producto' => $producto[ 'nombreProducto' ],
							'Precio Producto' => $producto[ 'precio' ],
							'Id Fila' => $i
						);
				$i++;

			}//end foreach

			$dictionary->convertListToString( $this->lista, $this->listaProductosTraducida, $this->templateList );

		}//end traducirListaProductos


		public function traducirPagina() {
			global $dictionary;

			$this->traducirListaProductos();

			$titulo = Work::getRegister( 'Titulos', 'titulo', 'idTitulo='.rand( 1, 4 ) );

			$diccionario = array(
						'Titulo' => $titulo[ 'titulo' ],
						'Listado Productos' => $this->listaProductosTraducida
					);

			$dictionary->translate( $diccionario );

		}//end traducirPagina

	}//end ErrorView

?>
```

Básicamente se obtienen la plantilla de la lista, se obtienen los datos y se genera una nueva plantilla que se guarda en un *word* agregado en un vector, luego se pasa el vector al objeto *diccionary->translate()* y se obtiene una plantilla resultante.
Como se observa los *words* que “dibujan” las zonas genéricas no son definidas en el vector, el sistema se encarga de hacer el *translate* de manera interna.

Este es un ejemplo sencillo, claro que 2MVC Framework permite agregar los listados necesarios y sin necesidad de que sean en tablas, utilizando la misma técnica.

Conclusión

El proyecto 2MVC Framework busca solucionar el problema del uso de más de un lenguaje en un único archivo, utilizando un algoritmo que se basa en la existencia en un *dictionary* de *words* que reemplaza palabras definidas por el diseñador por los datos dinámicos obtenidos.

Pero 2MVC Framework basa su comportamiento en la existencia de librerias core y librerias plugins, lo que hace que solo tenga que utilizar lo necesario, y está construido de una manera tal para que el consumo de recursos sea ínfimo.

Además no solo se encargará de la parte de la lógica de negocio, si no también en el diseño... ya que actualmente posee una versión muy básica de una librería css que permite crear webs adaptables con solo agregar clases a los elementos HTML