Manual de CodeIgniter

Autores del manual

Este manual ha sido realizado por los siguientes colaboradores de DesarrolloWeb.com:

Miguel Angel Alvarez
Director de DesarrolloWeb.com
http://www.desarrolloweb.com
(19 capítulos)

CodeIgniter

Probablemente ya sepamos que un framework es un programa para desarrollar otros programas, CodeIgniter, por tanto, es un programa o aplicación web desarrollada en PHP para la creación de cualquier tipo de aplicación web bajo PHP. Es un producto de código libre, libre de uso para cualquier aplicación.

Como cualquier otro framework, Codeigniter contiene una serie de librerías que sirven para el desarrollo de aplicaciones web y además propone una manera de desarrollarlas que debemos seguir para obtener provecho de la aplicación. Esto es, marca una manera específica de codificar las páginas web y clasificar sus diferentes scripts, que sirve para que el código esté organizado y sea más fácil de crear y mantener. CodeIgniter implementa el proceso de desarrollo llamado Model View Controller (MVC), que es un estándar de programación de aplicaciones, utilizado tanto para hacer sitios web como programas tradicionales. Este sistema tiene sus características, que veremos en artículos siguientes.

CodeIgniter no es magia, pero contiene muchas ayudas para la creación de aplicaciones PHP avanzadas, que hacen que el proceso de desarrollo más rápido. A la vez, define una arquitectura de desarrollo que hará que programemos de una manera más ordenada y contiene diversas herramientas que ayudan a hacer aplicaciones más versátiles y seguras.

CodeIgniter y otros frameworks PHP pueden ayudarte a dar el salto definitivo como desarrollador PHP, creando aplicaciones web más profesionales y con código más reutilizable, con la diferencia que Code Igniter está creado para que sea fácil de instalar en cualquier servidor y de empezar a usar que cualquier otro framework. Además muchas de sus utilidades y modos de funcionamiento son opcionales, lo que hace que goces de mayor libertad a la hora de desarrollar sitios web.

Características generales de CodeIgniter

Algunos de los puntos más interesantes sobre este framework, sobre todo en comparación con otros productos similares, son los siguientes:

Versatilidad: Quizás la característica principal de CodeIgniter, en comparación con otros frameworks PHP. CodeIgniter es capaz de trabajar la mayoría de los entornos o servidores, incluso en sistemas de alojamiento compartido, donde sólo tenemos un acceso por FTP para enviar los archivos al servidor y donde no tenemos acceso a su configuración.

Compatibilidad: CodeIgniter, al menos en el momento de escribir este artículo de desarrolloweb.com, es compatible con la versión PHP 4, lo que hace que se pueda utilizar en cualquier servidor, incluso en algunos antiguos. Por supuesto, funciona correctamente también en PHP 5.

Facilidad de instalación: No es necesario más que una cuenta de FTP para subir CodeIgniter al servidor y su configuración se realiza con apenas la edición de un archivo, donde debemos escribir cosas como el acceso a la base de datos. Durante la configuración no necesitaremos acceso a herramientas como la línea de comandos, que no suelen estar disponibles en todos los alojamientos.

Flexibilidad: CodeIgniter es bastante menos rígido que otros frameworks. Define una manera de trabajar específica, pero en muchos de los casos podemos seguirla o no y sus reglas de codificación muchas veces nos las podemos saltar para trabajar como más a gusto

encontremos. Algunos módulos como el uso de plantillas son totalmente opcionales. Esto ayuda muchas veces también a que la curva de aprendizaje sea más sencilla al principio.

Ligereza: El núcleo de CodeIgniter es bastante ligero, lo que permite que el servidor no se sobrecargue interpretando o ejecutando grandes porciones de código. La mayoría de los módulos o clases que ofrece se pueden cargar de manera opcional, sólo cuando se van a utilizar realmente.

Documentación tutorializada: La documentación de CodeIgniter es fácil de seguir y de asimilar, porque está escrita en modo de tutorial. Esto no facilita mucho la referencia rápida, cuando ya sabemos acerca del framework y queremos consultar sobre una función o un método en concreto, pero para iniciarnos sin duda se agradece mucho.

Sin duda, lo más destacable de CodeIgniter es su accesibilidad, ya que podemos utilizarlo en la mayor gama de entornos. Esta es la razón por la que en DesarrolloWeb.com hemos elegido este framework PHP para comenzar un manual que explicará cómo utilizarlo para desarrollar nuestras propias aplicaciones web. En siguientes artículos iremos contando diferentes aspectos de este framework y lo utilizaremos para crear una primera aplicación web. Para continuar puedes leer el artículo Instalación y configuración de CodeIgniter. También puedes ir al Manual de Codeigniter que estamos publicando.

*Artículo por **Miguel Angel Alvarez***

Instalación y configuración de CodeIgniter

Como apuntábamos en el artículo anterior, en el que presentamos las características principales de CodeIgniter, una de las ventajas de este framework PHP es que su instalación es muy sencilla. Veremos aquí algunos detalles sobre el proceso:

Requisitos de servidor

Necesitamos disponer de un servidor PHP 4 ó PHP 5. Ambos funcionan perfectamente y en el caso de PHP 4, la versión mínima que debemos tener es la PHP 4.3.2.

Por lo que respecta a las bases de datos, CodeIgniter es compatible con unas cuantas, las más habituales en e desarrollo de webs: MySQL (4.1 o posterior), MySQLi, MS SQL, Postgres, Oracle, SQLite, y acceso a cualquier base de datos en entornos Windows por ODBC.

Descarga de CodeIgniter

Podemos descargar la versión más actualizada de CodeIgniter directamente desde su página web, en la sección de descargas: http://codeigniter.com/downloads/

En el momento de escribir este artículo la versión más actual es la 1.7.2, pero probablemente cuando leas este texto ya la hayan actualizado, así que entra al sitio de descargas para estar seguro sobre la versión más nueva que exista.

Instalación de CodeIgniter en 4 sencillos pasos

1.- Descomprimir el paquete
Descomprime el archivo de descarga que has descargado

2.- Sube codeIgniter a tu servidor
Ahora tienes que subir todos los archivos descomprimidos a tu servidor web.

Opción A: Si estás programando en tu ordenador local, para pruebas y desarrollo, deberás tener un servidor instalado en tu ordenador que soporte PHP, para lo que te recomendamos los paquetes Wamp o Xampp. En este caso tendrás que copiar simplemente los archivos de codeIgniter en el directorio de publicación de tu servidor. Puedes colocar los archivos en la raíz del directorio de publicación o bien en un subdirectorio cualquiera.

Opción B: Si estás subiendo CodeIgniter a un servidor web de Internet, en un espacio que tengas contratado de hosting, deberás subir por FTP todos los archivos. Lo general es que copies CodeIgniter en la raíz de tu dominio, para que todas las páginas del dominio se sirvan a través del framework PHP, pero nada te impide copiar CodeIgniter en un subdirectorio particular, para que tu dominio web sólo funcione bajo CodeIgniter en la carpeta donde lo has subido.

3.- Configura la URL base de tu aplicación web
Necesitas decirle a CodeIgniter la URL base de tu aplicación, es decir, la URL para acceder a la raíz de CodeIgniter, según en el servidor y directorio donde lo has colocado, es decir, donde has subido el código del framework. Para ello tienes que abrir el archivo de configuración, que se encuentra en system/application/config/config.php, con cualquier editor de texto y cambiar la variable de configuración llamada que se guarda en $config['base_url'].
Si hemos instalado en local CodeIgniter pondremos http://localhost/ y si además lo colocamos en un directorio específico podría ser algo como http://localhost/directorio_codeigniter
Si hemos instalado el framework en un dominio de Internet podremos indicar algo como http://eldominiodeinstalacion.com/ y si hicimos una carpeta para subir CodeIgniter en ella pondremos el nombre del dominio y luego el nombre de la carpeta o carpetas, separadas por barras y acabando siempre en una barra: http://eldominioquesea.com/carpeta/otracarpeta/
Según nos indican en el manual de instalación, aparte de este dato podemos opcionalmente escribir una llave de encriptación en la variable $config['encryption_key'], que servirá si deseamos usar la clase de encriptado que proporciona CodeIgniter o queremos que nuestras variables de sesión estén encriptadas, algo que hace el framework de manera transparente para nosotros.

4. Configurar la base de datos
En este último paso tendrás que indicar los datos de acceso a la base de datos que piensas utilizar con CodeIgniter, ya que prácticamente todas las aplicaciones web que podrás crear con el framework van a tener que utilizar la base de datos para algo. Para ello tenemos que editar el archivo system/application/config/database.php e indicar los parámetros de conexión al servidor de base de datos, como el nombre del servidor y nombre de la base de datos, el usuario y la contraseña.

Con esto ya tenemos todo listo para comenzar a crear nuestras aplicaciones web PHP. Podemos testar CodeIgniter accediendo a la URL donde lo hemos instalado. Debemos ver el mensaje de bienvenida de CodeIgniter que nos confirma que está funcionando.

Para continuar la lectura puedes acceder al Manual de CodeIgniter que estamos publicando en DesarrolloWeb.com.

*Artículo por **Miguel Angel Alvarez***

Entender el flujo de aplicación de CodeIgniter

En DesarrolloWeb.com hemos empezado a publicar una línea de artículos sobre el framework PHP CodeIgniter, que es uno de los más sencillos, versátiles y apropiados para la mayoría de desarrolladores y proyectos. En este artículo vamos a comentar algunas de las cosas, más bien teóricas sobre el proceso de desarrollo y producción de aplicaciones con CodeIgniter.

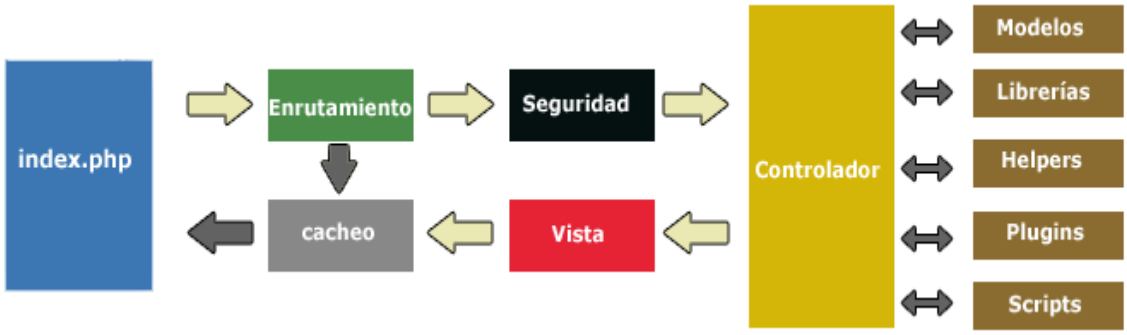
Recordemos que ya se presentaron unas características generales de CodeIgniter y que estuvimos ofreciendo una guía para la instalación del framework PHP, que resulta bastante sencilla, como podréis ver dado que CodeIgniter es apropiado para la mayoría de los entornos donde necesitemos ejecutar aplicaciones con programación en PHP.

Como cualquier herramienta o librería para el desarrollo, CodeIgniter define una serie de pautas de trabajo que necesitaríamos conocer antes de ponernos a escribir líneas de código. Si queremos utilizar este framework hay muchas cosas no podremos hacer tal como, quizás, estemos acostumbrados en el pasado con PHP. Así pues es muy importante saber unas cuantas cosas antes de empezar a ver ejemplos simples o explicar el proceso de desarrollo de una aplicación web más compleja. De hecho, la mayor complejidad de empezar a usar CodeIgniter es justamente la necesidad de cambiar nuestros esquemas y costumbres de desarrollo, pero esto también posibilitará que empecemos a pensar "a lo grande" y que los objetivos de aplicaciones avanzadas sean más fácilmente alcanzables.

Flujo de aplicación de CodeIgniter

En CodeIgniter existe un procedimiento para atender una solicitud de página del cliente. Este proceso se realiza internamente por el propio CodeIgniter y de manera transparente para nosotros. Durante el proceso participan varios módulos como el enrutamiento de la solicitud, la caché interna, etc. Está bien conocerlo de antemano, aunque algunas de las cosas aun no las entendamos con lo que hemos visto hasta el momento en este manual.

Conviene prestar atención sobre la siguiente imagen, tomada de la documentación de CodeIgniter, donde hemos traducido simplemente algunos nombres de los módulos que participan.



En resumen, para que se pueda entender el flujo de aplicación que implementa CodeIgniter, puedes seguir los siguientes puntos:

1. Toda solicitud de una página a partir de CodeIgniter comienza en un index.php que hay en la raíz del framework.
2. Luego se realiza un filtrado de la URL para saber cuál es elemento que tiene que

procesar esta página.
3. Si la página se había generado antes y está en la caché de CodeIgniter, se devuelve el archivo de la caché ya generado, con lo que se ahorra procesamientos repetidos. La caché se puede configurar y si lo deseamos, incluso deshabilitar.
4. Antes de continuar con el proceso se realiza un tratamiento de seguridad sobre la entrada que tengamos, tanto de la información que haya en la URL como de la información que haya en un posible POST, si lo hemos configurado así.
5. El controlador adecuado realiza el procesamiento de la solicitud. CodeIgniter decide el controlador que debe procesar la solicitud en función de la URL solicitada.
6. El controlador comunica con una serie de módulos, los que necesite, para producir la página.
7. A través de las vistas adecuadas, el controlador genera la página, tal cual se tiene que enviar al navegador.
8. Si la página no estaba en la caché, se introduce, para que las futuras solicitudes de esta página sean más rápidas.

Algunos de estos módulos, como la caché o el enrutamiento, funcionan de manera transparente para nosotros. Algunos otros, como los controladores, modelos y vistas, los tenemos que programar por nuestra cuenta y localizan cada una de las partes de nuestro programa que, al estar sepadadas nos ayudan a organizar también nuestro código. También tenemos a nuestra disposición diversas librerías, ayudantes (helpers) y plugins ya escritos en CodeIgniter con numerosas clases y funciones muy útiles para el desarrollo de aplicaciones web.

Uno de los puntos más básicos, pero no por ello menos útiles, de CodeIgniter que explicaremos dentro de poco, es el múdulo de entutamiento (routing) que permite que cualquier URL que se solicite al servidor se ejecute en el controlador adecuado. La URL se analiza y los datos se procesan y aseguran antes de enviarlos al controlador adecuado, en el que simplemente tendremos que codificar sus diversos métodos.

En el siguiente artículo estudiaremos el Modelo - Vista - Controlador, que implementa CodeIgniter como arquitectura para el desarrollo.

*Artículo por **Miguel Angel Alvarez***

Modelo - Vista - Controlador en CodeIgniter

Seguimos ofreciendo capítulos introductorios en el Manual de CodeIgniter de DesarrolloWeb.com, en esta ocasión vamos a explicar estilo de programación que tenemos que seguir cuando desarrollemos nuestras aplicaciones web en CodeIgniter.

El **Modelo, Vista, Controlador** es típicamente utilizado para la creación de aplicaciones web y no sólo CodeIgniter lo implementa, sino también otra serie de frameworks de desarrollo web, en PHP u otros lenguajes. Es interesante porque separa en varios grupos las complejidades de las distintas partes que componen una página web, como la vista y la lógica, así como el acceso a la base de datos.

Quizás lo que más nos fuerce a cambiar nuestros hábitos de programación en PHP es el hecho de tener que basar nuestros scripts en este modelo de programación, que seguramente resultará novedoso para la mayoría de los lectores de este manual, porque fija un nuevo estilo de desarrollo de aplicaciones, que nos obliga a separar código fuente según su ámbito. Sin embargo, como decíamos anteriormente, estas nuevas costumbres de codificación también nos

ayudarán a que nuestros programas sean mejores y disfruten de varias ventajas como ser más organizados, extendibles y entendibles por otros desarrolladores, reutilizables, con más fácil mantenimiento, etc.

Para los que no lo conocen, el Modelo - Vista - Controlador (en inglés Model - View - Controller) es un patrón de desarrollo o un estilo de arquitectura de software que separa el código fuente de las aplicaciones en tres grupos:

Modelo:
Todo el código que tiene que ver con el acceso a base de datos. En el modelo mantendremos encapsulada la complejidad de nuestra base de datos y simplemente crearemos funciones para recibir, insertar, actualizar o borrar información de nuestras tablas. Al mantenerse todas las llamadas a la base de datos en un mismo código, desde otras partes del programa podremos invocar las funciones que necesitemos del modelo y éste se encargará de procesarlas. En el modelo nos podrán preocupar cosas como el tipo de base de datos con la que trabajamos, o las tablas y sus relaciones, pero desde las otras partes del programa simplemente llamaremos a las funciones del modelo sin importarnos qué tiene que hace éste para conseguir realizar las acciones invocadas.

Vista:
La vista codifica y mantiene la presentación final de nuestra aplicación de cara al usuario. Es decir, en la vista colocaremos todo el código HTML, CSS, Javascript, etc. que se tiene que generar para producir la página tal cual queremos que la vea el usuario. En la práctica la vista no sólo sirve para producir páginas web, sino también cualquier otra salida que queramos enviar al usuario, en formatos o lenguajes distintos, como pueden ser feeds RSS, archivos JSON, XML, etc.

Controlador:
El controlador podríamos decir que es la parte más importante, porque hace de enlace entre el modelo, la vista y cualquier otro recurso que se tenga que procesar en el servidor para generar la página web. En resumen, en el controlador guardamos la lógica de nuestras páginas y realizamos todas las acciones que sean necesarias para generarlas, ayudados del modelo o la vista.

> **Nota:** Esto quiere decir, en la práctica para el caso de CodeIgniter, que según sea el ámbito donde estemos codificando, tendremos que escribir las líneas de código de cualquier página web en tres grupos de archivos distintos. En una aplicación estándar podremos tener varios modelos (por ejemplo, para cada una de las entidades distintas de la base de datos), varias vistas (una o varias para cada página o sección) y varios controladores (para cada página o sección de la web). Luego veremos dónde tenemos que guardar los archivos de cada uno de estos tres grupos.

Durante el desarrollo con CodeIgniter será muy recomendable seguir las normas del diseño Modelo - Vista - Controlador (MVC), pero realmente el framework es bastante flexible y permite que, si lo deseamos, no sigamos el desarrollo atendiendo a dicha arquitectura. En este caso, podríamos tener simplemente controladores y dentro de ellos realizar todas las acciones de acceso a la base de datos directamente, sin hacer llamadas al modelo, o escribir texto en la salida sin utilizar las vistas. Obviamente, esta opción no aprovechará las ventajas de separación de código entre presentación, lógica y modelo de base de datos, pero si vemos que nos resulta muy complejo el MVC, podemos dejarlo de lado.

En el caso que no utilizemos los modelos, no ocurrirá ningún efecto negativo en el desempeño del framework, pero en el caso de las vistas, si no las utilizamos y escribimos salida directamente desde el controlador, como por ejemplo con sentencias echo de PHP en en código de los controladores, perderemos algunas de las ventajas que CodeIgniter realiza por nosotros para procesar la salida antes de enviarla al usuario. Esto lo detallaremos más adelante.

Nota: Como sabemos, o podremos imaginar, separar la vista o presentación de la lógica de los programas es una ventaja importante, ya que ayuda a mantener un código más sencillo, reducido y con mayor facilidad de mantenimiento. En principio puede parecer un tanto complejo el hecho de manejar varios archivos con códigos distintos, pero a medio plazo nos vendrá muy bien separar el código de cada parte de nuestra aplicación. Nos evitará muchos de los problemas que a veces tenemos en desarrollos PHP, donde en un mismo archivo tenemos mezclado código en varios lenguajes como HTML, CSS, SQL, con el propio código PHP para generar la lógica de nuestro negocio.

*Artículo por **Miguel Angel Alvarez***

URLs en CodeIgniter

Uno de los puntos que debemos conocer antes de empezar a trabajar con CodeIgniter es sobre cómo son las direcciones URL que utiliza este popular framework PHP para cada una de las páginas de las aplicaciones PHP que creemos utilizándolo. La verdad es que es un punto que realmente resulta transparente para nosotros, puesto que las URL se generan automáticamente a medida que vamos programando el sitio web, pero está bien comentar algunas cosas importantes.

Éste es un artículo del Manual de CodeIgniter que estamos publicando en DesarrolloWeb.com, por lo que para entenderlo habrá que leer alguno de los artículos previos a éste.

URLs amigables a buscadores

Uno de los puntos fuertes de este framework PHP es que las URL se presentan siempre en un formato amigable a los buscadores. Esto quiere decir que cualquier motor de búsqueda puntuará positivamente, a priori, las direcciones de las páginas. Del mismo modo, las direcciones tendrán una forma fácil de entender y recordar por los seres humanos.

En DesarrolloWeb.com, en los manuales de posicionamiento y promoción en buscadores hemos hablado repetidas veces acerca de las URL amigables, pero para el que no lo sepa, estas son URL que no suelen ser puntuadas bien por los buscadores:

www.midominio.com/articulos.php?id=32
www.midominio.com/articulos.php?nombre=miarticulo

Si lo vemos, tenemos una página, articulos.php, que se le pasan distintos parámetros. Pero los buscadores muchas veces interpretan que es la misma página. Bueno, en realidad no tiene más importancia, porque con CodeIgniter las URL tienen mucha mejor arquitectura, con formas como estas:

www.midominio.com/articulos/muestra/32
www.midominio.com/controlador/funcion/parametro

Las diferencias saltan a la vista, tanto para nosotros humanos como para motores de búsqueda como Google. Y lo bueno es que nosotros no tenemos que hacer nada para conseguir este tipo de direcciones.

Nota: Hay un pequeño matiz que comentaremos más tarde en este artículo y es que en principio todas las URL en CodeIgniter tienen el nombre de una página llamada index.php, pero esto es algo que podemos hacer desaparecer si sabemos configurar el framework.

Query String desactivado

En CodeIgniter en principio está desactivada la posibilidad de envío de variables a través de la URL, lo que se conoce en inglés como Query String. Es decir, que direcciones en las que se envían variables a través de las URL, que decíamos que eran poco amigables a buscadores, no funcionarán.

Si se desea, se puede hacer que CodeIgniter reconozca las variables enviadas por la URL, pero como en principio el sistema de URLs amigables a buscadores que implementa el framework está pensado para poder evitar el problemático Query String, su uso está desactivado.

Segmentos de la URL y el modelo - vista - controlador

Cada una de las partes de la URL de las aplicaciones creadas con el framework sirve para identificar qué controlador, del ya explicado Modelo - Vista - Controlador de CodeIgniter, se va a hacer cargo del procesamiento de la página, así como de la función que se invocará y los parámetros que se le enviarán a la misma. Por ejemplo:

aplicacioncodeiginter.com/facturacion/editarempresa/5610

- **aplicacioncodeiginter.com** es el nombre del supuesto dominio donde tenemos CodeIgniter instalado.
- **facturacion** es el nombre del controlador que se encargará de procesar la solicitud.
- **editarempresa** es el nombre de la función que habrá dentro del controlador y donde estará el código que ejecute y genere la página. Aunque para ser correctos, como el controlador está en programación orientada a objetos, en vez de función, deberíamos llamarle método.
- Por último, **5610**, es el parámetro que se le pasa a la función editarempresa, que servirá en este caso para que editarempresa sepa cuál es la empresa que se desea editar. Si queremos o necesitamos enviar varios parámetros a esta función, y no sólo el identificador de la empresa a editar, podremos colocarlos a continuación, separados por barras.

Nota: Esto quizás resulte ahora un poco complicado, porque todavía no tenemos una idea definida sobre cómo son los controladores, pero dentro de poco lo veréis todo más claro.

CodeIgniter pone a nuestra disposición una clase para trabajar con URLs llamada URI Class y una librería llamada URL Helper que contienen funciones para trabajar fácilmente con URLs y datos enviados en las mismas. En estas librerías hay funciones tan interesantes como site_url() que sirve para que el propio CodeIgniter cree una URL dentro del sitio a partir de un parámetro que le pasemos. Otro ejemplo es base_url(), que simplemente devuelve la URL raíz donde está nuestra aplicación CodeIgniter.

Todo pasa por index.php

En CodeIgniter existe un index.php que está en la raíz del framework que se encarga de las funciones de enrutamiento hacia el controlador que se debe encargar de procesar la solicitud. Por ello, de manera predeterminada en CodeIgniter veremos que las URLs incluyen el nombre del archivo index.php. Este comportamiento se puede configurar.

En el próximo artículo explicaremos cómo eliminar este index.php en las URLs de CodeIgniter,

algo que simplificará las direcciones.

Añadir un sufijo a las URL

Otro de los detalles que podemos hacer con CodeIgniter, que pueden personalizar aun más nuestras direcciones URL, es añadir un sufijo, que nosotros deseemos, al final de todas las URL que formen parte del framework. Por ejemplo, podríamos desear que todas las URL acaben en .html o en .php, o como queramos. Esto se puede hacer a través de los archivos de configuración del framework.

La idea es que una URL como esta:

http://dom.com/index.php/blog/post/cualquier-articulo

Pase a ser una dirección como esta otra:

http://dom.com/index.php/blog/post/cualquier-articulo.html

Para esto editamos el archivo de configuraciones generales: system/application/config/config.php y tenemos que buscar la variable url_suffix y colocar el valor que deseemos, por ejemplo:

```
$config['url_suffix'] = ".html";
```

*Artículo por **Miguel Angel Alvarez***

Eliminar el index.php de las direcciones de CodeIgniter

Lo cierto es que las direcciones de CodeIgniter son bastante amigables a buscadores, pero todavía pueden serlo más. Esto es algo que vamos a ver en este artículo de DesarrolloWeb.com.

En el artículo anterior, sobre las características de las URL de CodeIgniter, ya explicamos varias cosas que conviene saber acerca de componer direcciones para las páginas de nuestras aplicaciones PHP. Como se decía en ese artículo, todas las solicitudes a páginas de una aplicación web en CodeIgniter pasan por un archivo index.php que está en la raíz del framework. Este index.php se encarga de redirigir, o enrutar, la solicitud a través del controlador que se esté invocando (que se indica en el primer segmento después del nombre del dominio).

Así pues, por defecto las URL de CodeIgniter tienen un formato como el que sigue:

http://pruebas.com/index.php/empresas/editar/1

Pues bien, si deseamos eliminar el index.php de esta URL, para simplificar la dirección, hacerla todavía más amigable a buscadores y también más entendible por los seres humanos, podemos utilizar un archivo .htaccess.

> **Nota:** En DesarrolloWeb.com hemos hablado en repetidas ocasiones acerca de los archivos .htaccess, explicando lo que son y cómo podemos utilizarlos para crear automáticamente URLs amigables a buscadores, sin que tengan que existir físicamente los archivos en el servidor. Recomendamos la lectura del manual Editar htaccess para crear direcciones URL amigables

http://www.desarrolloweb.com/manuales/htaccess-para-urls-amigables.html para obtener una información de referencia.

Así pues, podríamos conseguir que nuestras direcciones no tuvieran siempre el mencionado index.php y quedasen con una forma similar a esta:

http://pruebas.com/empresas/editar/1

Para ello existirían diversos métodos y según nuestro dominio de los archivos .htaccess y de las expresiones regulares, así como de la configuración de Apache, podremos implementar una u otra. Nosotros en este artículo vamos a explicar un modo que está disponible en la propia guía de uso de CodeIgniter, que hemos probado y resulta sencillo y efectivo.

Se trata de utilizar un método de trabajo con .htaccess que llaman "negativo", donde todas las URLs son redirigidas a través del archivo index.php, a no ser que tengan una forma determinada que no queramos que se redireccione.

Por ejemplo, una URL como esta:

www.loquesea.com/empresas

Se redirigiría a una URL como esta otra:

www.loquesea.com/index.php/empresas

Pero esa redirección se haría de manera transparente al usuario y al navegador que nos visita. Es decir, el procesamiento de la página se hace a través del index.php, pero nosotros no llegamos a percibir que en el servidor se ha llevado a cabo esa redirección, quedando la URL en la barra de direcciones siempre sin el index.php.

Ahora bien, hay determinadas direcciones que no vamos a desear que se redirijan, como puede ser una hoja de estilos CSS. Es decir, podemos tener en nuestro servidor un archivo como este:

www.loquesea.com/css/estilos.css

Y en ningún caso queremos que se procese a través del index.php, sino que directamente se devuelva el archivo CSS que hay en esa ruta. Este es el caso en el que se utiliza el método "negativo" del htaccess, a través del comando "RewriteCond" (condición de redirección). Este caso obvio también lo tendríamos, por ejemplo, en archivos como el robots.txt o aquellos donde podamos tener librerías de código Javascript.

Código htaccess para eliminar el index.php de CodeIgniter

Así pues, podríamos tener un archivo .htaccess en la raíz de la instalación del framework, que suele ser también la raíz del dominio donde estamos trabajando, con un código como el que sigue:

```
RewriteEngine on
RewriteCond $1 !^(index.php|css|js|images|robots.txt)
RewriteRule ^(.*)$ /index.php/$1 [L]
```

Esto diría que se redirijan todos los archivos que se soliciten a este dominio, menos los que contengan el propio index.php, css, js, images o robots.txt.

La redirección se hará a través de la misma URL, pero agregando "index.php/" después del nombre del dominio y antes de lo que haya justo después de ese nombre de dominio.

Nota: si tuviéramos CodeIgniter instalado en un subdirectorio de nuestro dominio, por ejemplo, en el subdirectorio "carpeta", la regla de redirección "RewriteRule" cambiaría un poco:

```
RewriteRule ^(.*)$ /carpeta/index.php/$1 [L]
```

Ahora, sólo nos faltaría decirle a CodeIgniter que, al componer URLs para los enlaces entre las distintas páginas de nuestra aplicación web PHP, no coloque el index.php, pues no lo necesitamos. Esto lo podemos hacer desde el archivo de configuración principal, que están en: system/application/config/config.php

Simplemente tendríamos que editar la variable de configuración "index_page", poniéndola a la cadena vacía.

```
$config['index_page'] = "";
```

Este no sería necesariamente el único método. Por ejemplo, en la Wiki oficial de Codeigniter explican un modo distinto de conseguirlo, aunque bastante más complejo: http://codeigniter.com/wiki/mod_rewrite/

*Artículo por **Miguel Angel Alvarez***

Controllers en CodeIgniter

Los controladores son uno de los componentes que forman parte del modelo de programación MVC (Modelo - Vista - Controlador o Model - View - Controller en inglés) que sirven como engranaje principal a la hora de crear aplicaciones web. Dado que CodeIgniter utiliza la arquitectura MVC, tendremos que aprender a manejar y dominar los controladores como un primer paso para comenzar a trabajar con este framework PHP.

Para los que no sepan qué es el MVC cabe decir que ya fue explicado en este manual de CodeIgniter, concretamente en el capítulo Modelo - Vista - Controlador en CodeIgniter. Así que podemos pasar directamente a explicar las particularidades de los controladores en este entorno de programación.

Para explicar los controladores voy a basarme en la propia documentación del framework PHP, que está tutorializada, como ya dijimos en la introducción a CodeIgniter. Por lo que resulta una manera excelente para aprender a dar los primeros pasos en el desarrollo de aplicaciones.

Qué es un controlador

Un controlador en CodeIgniter es un archivo que contiene el código de una clase, de programación orientada a objetos, que colocamos en un directorio específico del esquema de carpetas de nuestro sitio. Tiene un nombre cualquiera, que se asociará con una URL de nuestra aplicación web.

Por ejemplo, esta podría ser una URL de nuestra aplicación:

midominio.com/index.php/articulos

En la URL anterior podemos ver que la palabra "artículos" determina la página que queremos ver dentro de nuestra aplicación. Pues bien, para poder atender esta solicitud nosotros vamos a tener que crear un archivo llamado articulos.php (el controlador) dentro del directorio que aloja los controladores de nuestra aplicación.

Por tanto, cuando CodeIgniter detecta una URL como esta, intentará acceder al archivo articulos.php, para cargarlo, procesarlo y de ese modo mostrar la página con los contenidos de esta sección.

Cuando ese controlador no se encuentre entre los archivos de controladores de CodeIgniter, simplemente se mostrará un error 404 de página no encontrada. Si se encontró el controlador, como se decía, se carga y se procesa para mostrar la página.

Los controladores en CodeIgniter se guardan en la carpeta "system/application/controllers/", que se encuentra dentro de los archivos de CodeIgniter.

Nota: Todos los archivos que vamos a tener que crear o editar para desarrollar nuestra aplicación PHP con CodeIgniter están dentro de la carpeta "system/application" y existe una carpeta dentro para cada tipo de cosa que deseemos crear o modificar, como archivos de configuración, modelos, vistas, etc.

Creando un primer controlador en CodeIgniter

Para empezar a programar en CodeIgniter vamos a crear un primer controlador, llamado artículos, que simplemente mostrará un mensaje de bienvenida para saber que todo está funcionando correctamente.

El código de este primer controlador será el siguiente:

```
<?php
class Articulos extends Controller {

  function index()
  {
    echo 'Bienvenido a mi primer controlador en CodeIgniter';
  }
}
?>
```

Este archivo lo tenemos que guardar como "articulos.php" en la carpeta "system/application/controllers/". Caben señalar unos detalles importantes:

- En nombre del archivo de controlador, en este caso articulos.php, va en minúsculas.
- El nombre de la clase que implementa el controlador se tiene que llamar igual que el nombre del archivo, pero fijaros que tiene obligatoriamente la primera letra en mayúscula. Por eso aparece como class **Articulo** extends Controller.
- Todos los controladores tienen que extender la clase "Controller" (que también tiene la primera letra "C" en mayúscula), que está creada dentro de CodeIgniter y en principio no necesitamos modificarla para nada.

Una vez creado el archivo, podemos acceder con el navegador al controlador, con una URL como esta:

http://localhost/index.php/articulos

Nota: Esta URL supone que has instalado CodeIgniter en la raíz de tu servidor web, pero podrías haber instalado CodeIgniter en otro directorio, en tal caso la URL tendría una forma como http://localhost/directorio_de_codeigniter/index.php/articulos

En caso que esté funcionando todo correctamente, tendrás que visualizar una página con el mensaje que habías colocado en el echo que hay dentro de la función index() del controlador.

Si no funciona, comprueba tu URL, que tiene acceder al directorio correcto donde hayas instalado CodeIgniter y que tiene que acabar con el nombre del controlador en minúsculas. Comprueba también que la clase que has creado en el código del controlador tiene la primera letra en mayúscula. No debe haber mucho problema ni dificultad.

En el siguiente artículo veremos cómo podemos crear funciones en los controladores, para dar lugar páginas distintas que dependan del mismo controlador.

Controlador por defecto

Pero, antes de acabar el presente artículo, quiero hablar sobre el controlador por defecto, que es el que se invoca en CodeIgniter cuando no se especifica ningún nombre de directorio y por tanto ningún controlador, en la siguiente URL:

midomino.com/index.php/

O bien en esta otra:

midominio.com/

Esto, que sería la home de la aplicación CodeIgniter, y eventualmente la home del dominio, si es que hemos instalado el framework en la raíz del directorio de publicación, invoca también un controlador, que es el controlador por defecto.

El nombre del controlador predeterminado (Default Controller) puede ser variable, siendo el desarrollador el encargado de configurarlo en uno de los archivos de configuración de CodeIgniter, que se encuentra en el fichero "application/config/routes.php".

En ese archivo simplemente tenemos que buscar el valor $route['default_controller'] y asignarle el nombre de la clase donde está el controlador que queramos que se invoque. Como es el nombre de una clase de un controlador tendremos que escribirlo con la primera letra en mayúscula.

```
$route['default_controller'] = 'Articulos';
```

Si observamos el valor de esta variable de configuración, tal como está en el momento inicial, después de la instalación de CodeIgniter, veremos que el controlador por defecto está en la clase "Welcome", archivo application/controllers/welcome.php

*Artículo por **Miguel Angel Alvarez***

Funciones en los controladores

En el capítulo anterior del Manual de CodeIgniter, que venimos publicando en DesarrolloWeb.com, estuvimos explicando lo que son los controladores, la parte fundamental de las aplicaciones basadas en la arquitectura de desarrollo denominada MVC. Conocer los controladores es básico y nos permitirá hacer nuestras primeras pruebas con este framework Javascript. A continuación explicaremos para qué sirven las distintas funciones o métodos que podemos incluir en los controladores.

CodeIgniter dispone de un mecanismo sencillo para dar pie a URLs distintas que dependen del mismo controlador. Es decir, podemos asociar a un controlador muchas páginas, con URL distintas, que se ejecutan llamando siempre a un mismo controlador.

Por ejemplo, pensemos que tenemos una tienda online de venta de productos informáticos. Entonces podríamos tener una URL donde se muestran los productos, pero a su vez tendremos muchas clasificaciones de productos y cada cliente podría ver una ficha completa con cada uno de ellos. Todas las fichas de productos se podrían meter dentro de un mismo controlador, dando lugar a distintas URL, como pueden ser las siguientes:

dominio.com/index.php/productos
dominio.com/index.php/productos/ordenadores
dominio.com/index.php/productos/monitores
dominio.com/index.php/productos/perifericos/raton_genius
dominio.com/index.php/productos/ordenadores/mac_mini/2gigas
dominio.com/index.php/productos/ordenadores/mac_mini/4gigas
...

Todas estas URL dependen del mismo controlador, que sería el controlador de productos (tal como se dijo en el anterior artículo sería la clase "Productos" que extiende la clase "Controller" y debemos colocar en el archivo "productos.php" dentro del directorio donde se colocan los controladores (application/controllers).

Como se puede ver, el primer nombre del directorio después del dominio y el index.php es el nombre del controlador, en este caso siempre "productos". Por eso todas las URL que teníamos antes dependen del mismo controlador. Sin embargo, las URL que he colocado aquí son un puro ejemplo, para que se pueda ver cómo se pueden generar URLs que dependen de un mismo controlador. Pero dependiendo de cómo deseemos organizar nuestro sitio web podríamos decidir hacer uso de otras URL para mostrar más o menos lo mismo. Lo bueno, como también se puede comprobar viendo esas URL, es que en CodeIgniter siempre se generan URLs sin paso de parámetros, es decir, siempre totalmente amistosas a motores de búsqueda.

El segundo directorio corresponde a los nombres de las funciones

Ahora que ya sabemos que el primer directorio después del index.php corresponde con el nombre del controlador, podemos observar el segundo directorio de las URL generadas con CodeIgniter. Éste se refiere al nombre de la función o método que atenderá esa solicitud dentro del controlador.

Si no se indica ningún nombre de función CodeIgniter siempre invoca a la función llamada index(), así pues, la URL:

dominio.com/index.php/productos

Será procesada por la función index() que coloquemos en el controlador. Este es un caso especial, porque no se ha indicado nombre de función, pero dadas URLs como estas:

dominio.com/index.php/productos/ordenadores
dominio.com/index.php/productos/monitores

Tendremos que generar dichas funciones en el controlador, como se puede ver en el siguiente código:

```
<?php
class Productos extends Controller {

  function index(){
    echo 'Esto es la portada de productos';
  }
```

```
  function ordenadores(){
    echo 'Aquí se muestran los productos de ordenadores';
  }

  function monitores(){
    echo 'Aquí se muestran los productos de monitores';
  }
}
?>
```

Como se puede ver, hemos creado dos nuevas funciones dentro de un controlador para productos, para poder mostrar dos tipos de productos distintos.

> **Aclaración:** Estamos refiriéndonos siempre como funciones de los controladores, pero realmente deberíamos llamarles métodos, porque los controladores son clases y las clases tienen métodos y no funciones.

Tercer directorio y los siguientes son los parámetros que se envían a las funciones

Ahora vamos a observar otras URL que dependen del controlador de productos, pero que tienen más de un directorio adicional al nombre del controlador. Vimos que el primer directorio es el controlador, el segundo el método que se invocará dentro del controlador y ahora veamos que pasa con el tercer parámetro y los siguientes, en caso que haya.

dominio.com/index.php/productos/perifericos/raton_genius
dominio.com/index.php/productos/ordenadores/mac_mini/2gigas
dominio.com/index.php/productos/ordenadores/mac_mini/4gigas

Estos directorios adicionales son simples parámetros en los métodos del controlador. Así, por ejemplo, la primera de las URL que acabamos de ver se corresponde con el método "perifericos", pasándole el nombre del periférico que se desea ver "raton_genius". Este dato adicional se puede recibir en el método con un código como este:

```
function perifericos($modelo){
  echo 'Estás viendo el periférico ' . $modelo;
}
```

Así tenemos el método "perifericos" que recibe el parámetro modelo. Este parámetro tomará el valor del tercer directorio. Entonces en la URL:

dominio.com/index.php/productos/perifericos/raton_genius

Estaremos invocando el método perifericos() y pasando como valor en el parámetro $modelo la cadena "raton_genius".

Esto se puede complicar un poco más en URL como estas:

dominio.com/index.php/productos/ordenadores/mac_mini/2gigas

Como se puede comprobar, aquí estamos pasando dos parámetros al método ordenadores() del controlador "Productos", que podríamos procesar de una manera similar a esta:

```
function ordenadores($marca=null, $modelo=null){
  if (is_null($marca) && is_null($modelo)){
    echo 'Aquí se muestran los productos de ordenadores';
  }elseif(is_null($modelo)){
    echo 'Mostrando ordenadores de marca ' . $marca;
```

```
  }else{
    echo 'Mostrando ordenadores de marca ' . $marca . ' y modelo ' . $modelo;
  }
}
```

En la anterior función se comprueba qué parámetros se recibe en el método antes de dar por sentado que estamos recibiendo alguno de ellos. Procesará correctamente URLs como estas:

http://localhost/index.php/productos/ordenadores
http://localhost/index.php/productos/ordenadores/toshiba
http://localhost/index.php/productos/ordenadores/hp/pavillon590

Con todo lo que hemos visto hasta ahora de controladores, tenemos este código en el archivo "productos.php" del directorio "controllers".

```
<?php
class Productos extends Controller {

  function index(){
    echo 'Esto es la portada de productos';
  }

  function ordenadores($marca=null, $modelo=null){
    if (is_null($marca) && is_null($modelo)){
      echo 'Aquí se muestran los productos de ordenadores';
    }elseif(is_null($modelo)){
      echo 'Mostrando ordenadores de marca ' . $marca;
    }else{
      echo 'Mostrando ordenadores de marca ' . $marca . ' y modelo ' . $modelo;
    }
  }

  function monitores(){
    echo 'Aquí se muestran los productos de monitores';
  }

  function perifericos($modelo){
    echo 'Estás viendo el periférico ' . $modelo;
  }
}
?>
```

Esperamos que con lo que se sabe hasta el momento se puedan hacer las primeras pruebas con controladores en CodeIgniter. Sin embargo, hay que decir que aun no lo hemos visto todo con relación a los controladores, aunque lo dejaremos para el próximo artículo.

*Artículo por **Miguel Angel Alvarez***

Vistas en CodeIgniter

Vamos a pasar a un nuevo tema en el Manual de CodeIgniter en el que comenzaremos las explicaciones de las vistas, parte fundamental en este framework PHP y en el patrón de desarrollo de aplicaciones que utiliza, llamado modelo - vista - controlador.

En artículos anteriores de DesarrolloWeb.com ofrecimos una buena introducción a los controladores, y aunque realmente no hemos visto todo lo que los controladores nos ofrecen, estamos en condiciones de aprender otras cosas jugosas en CodeIgniter, que seguro nos motivarán durante nuestro aprendizaje, al ver que todo es bastante sencillo de asimilar.

Qué son las vistas

Una vista es una página web o un fragmento que se guarda en un archivo aparte. En una vista podríamos guardar, por tanto, toda la estructura de una página, o si preferimos una organización más minuciosa y por módulos, podremos guardar simplemente una sección, como puede ser la cabecera, pie, barra de navegación, etc. El grado de complejidad en el trabajo con vistas podremos marcarlo nosotros, según nuestras preferencias, costumbres de desarrollo o necesidades específicas, dado que podremos anidar unas vistas dentro de otras en cualquier nivel de jerarquía.

Las vistas no se acceden ni se invocan directamente con la solicitud de una URL en nuestra aplicación web, como ya podremos suponer. En realidad son módulos que se invocan desde los controladores, ya que en el modelo - vista - controlador que implementa CodeIgniter, la lógica de nuestra aplicación se almacena en el controlador y es éste el que debe llamar a las vistas que necesite para mostrar los resultados al visitante.

Así pues, los controladores decidirán qué hacer cuando se reciba una solicitud y las vistas decidirán cómo mostrar los resultados. Por decirlo de otra forma, la lógica de nuestra aplicación residirá en el controlador y la vista mantendrá el aspecto de nuestra página, el diseño de la página que se mostrará al usuario.

Crear una vista

Crear una primera vista en nuestra aplicación web es muy sencillo. Simplemente creemos un archivo con un poco de código HTML.

```
<html lang="es">
<head>
<title>Mi página web</title>
</head>
<body>
<h1>Bienvenido a mi web</h1>
</body>
</html>
```

Ahora podemos guardar esa vista con el nombre que deseemos y extensión ,php, por ejemplo mivista.php, en el directorio de las vistas que es "system/application/views".

Cargar una vista

Como decíamos, podemos invocar una vista desde un controlador. De hecho, cada vez que deseemos mostrar en la página cualquier texto, debemos invocar a la vista que necesitemos, o al menos sería la manera de proceder, en vez de hacer sentencias "echo" o similares directamente en el controlador.

Para cargar una vista hacemos lo siguiente:

```
$this->load->view('nombreDeLaVista');
```

En 'nombreDeLaVista' tendremos que indicar el nombre del archivo donde hemos guardado la vista, pero sin el ".php".

Veamos entonces cómo quedaría un controlador que llama a la vista que hemos hecho antes en este artículo.

```
<?php
```

```
class MiControlador extends Controller {

  function index(){
    $this->load->view('mivista');
  }

}
?>
```

Este controlador lo guardamos en la carpeta de controllers con el nombre "micontrolador.php". Y ahora podremos acceder a él por medio de una URL como esta:

http://localhost/index.php/micontrolador

O si tenemos CodeIgniter instalado en nuestro dominio, con una URL como esta otra:

http://www.midominio.com/index.php/micontrolador

Deberíamos ver simplemente el contenido de esa vista creada anteriormente. Es así de simple.

Almacenar vistas en subdirectorios

Debemos colocar todas las vistas en el directorio "views" (system/application/views), pero si lo deseamos podemos organizarlas por subdirectorios, lo que puede servir de utilidad si vamos a manejar gran número de vistas distintas, que dependen de secciones variadas de la página.

Para ello simplemente creamos un subdirectorio con el nombre que queramos, dentro de "views" y guardamos allí las vistas. Por ejemplo pensemos que tenemos varias vistas que pertenecen todas a la zona de registro de usuarios y queremos organizarlas en la carpeta "registro" (ruta completa "system/application/views/registro"). Entonces, las vistas colocadas allí se invocarían indicando el subdirectorio donde se encuentran, de la siguiente manera:

```
$this->load->view('registro/formulario_registro');
```

Llamar a varias vistas desde un controlador

Nada impide que organicemos nuestro contenido por partes, en vistas distintas, de hecho es una buena práctica que nos ayudará a mantener nuestro código de una manera más sencilla. Llamar a las distintas vistas irá concatenando todo el código HTML creado por cada vista y luego se enviarán al navegador todas juntas como el resultado de procesar una página.

Podremos invocar varias vistas en una función de un controlador, de una manera similar a esta:

```
function cargarVistas(){
  //cargo una vista cabecera.php que está en el directorio plantillas
  $this->load->view('plantillas/cabecera');

  //cargo una vista llamada formulario_busqueda.php
  $this->load->view('formulario_busqueda');

  //creo un array de datos para enviarlo a una vista
  $datos['page_title'] = 'Mi Título';
  //cargo una vista llamada cuerpo.php a la que le mando un array de datos para configurarla
  $this->load->view('cuerpo', $datos);

  //cargo una vista que está en el directorio plantillas
  $this->load->view('plantillas/pie');
}
```

En este ejemplo hemos cargado varias vistas y en una de ellas le hemos enviado datos para configurar cómo se mostrará al visitante, en concreto la vista cuerpo.php. Esta utilidad es fundamental para que las vistas muestren cualquier información que se envíe desde los controladores y no solamente un texto siempre igual. En el próximo artículo sobre vistas aprenderemos a realizar esta acción.

*Artículo por **Miguel Angel Alvarez***

Más sobre las vistas

Como hemos dicho anteriormente, las vistas serán parte fundamental de nuestras aplicaciones web en CodeIgniter. En el artículo anterior del Manual de CodeIgniter ya comenzamos a explicar qué eran las vistas y los detalles generales sobre su uso. No obstante, aun tenemos algunas cosas en el tintero que debes saber sobre las vistas.

En el presente artículo vamos a conocer varias cosas importantes sobre las vistas en CodeIgniter. Comenzaremos explicando cómo podemos pasar valores a las vistas para personalizarlas, de manera que muestren una información dada y no siempre el mismo texto fijo. Luego veremos que en las vistas podemos colocar cualquier código PHP, además del código HTML necesario para mostrar la información y que existen diversos esquemas de codificación alternativos en CodeIgniter, que podemos utilizar para ahorrar código fuente. Acabaremos mostrando cómo una vista puede devolver el texto resultado de procesarse a una variable, en vez volcarlo en la salida por defecto, es decir, en vez de enviarlo al cliente como código de la página.

Añadiendo datos dinámicos a las vistas

Una de las cosas más básicas que querremos hacer con las vistas es personalizar las informaciones que muestren a través de unas variables de configuración. Así, las vistas podrán mostrar diferentes datos con un mismo código.

Por ejemplo, pensemos que tenemos una vista con el código de una página completa, con el diseño propio de nuestro sitio. Esa vista podríamos estar usándola para todas las páginas del sitio web, pero en ese caso enviaremos distintas variables de configuración con los datos propios de cada página, como pueden ser el título de la página, el cuerpo, etc.

Todos esos datos se envían a través de un array asociativo que se pasa a la vista, y dentro de ella podremos acceder a cada uno de los elementos de ese array a partir del índice de ese elemento. Lo podremos ver fácilmente con un ejemplo.

Comenzamos por construir un array asociativo con los datos de la vista. En el código siguiente podemos ver el controlador que hicimos en el artículo anterior para mostrar el trabajo con vistas, en el que hemos añadido el código para generar el array asociativo y enviarlo a la vista.

```
<?php
class MiControlador extends Controller {

  function index(){

    $datos = array(
      'titulo' => 'Página de prueba',
      'descripcion' => 'Esta es la descripción de esta página, un poco más larga.',
      'cuerpo' => 'El cuerpo de la página probablemente será un texto muy largo...<p>Con varios párrafos</p>'
```

```
    );

    $this->load->view('mivista', $datos);
  }

}
?>
```

Ahora, en la vista, podemos acceder a todos los datos de ese array asociativo, como si fueran variables normales, a través de los nombre de los índices del array. Veamos el código de mivista.php que consume los datos que se le envían para configurar su contenido.

```
<html>
<head>
<title><?php echo $titulo?></title>
</head>
<body>
<h1><?php echo $titulo?></h1>
  <blockquote>
    <p><b><?php echo $descripcion?></b></p>
  </blockquote>
  <p><?php echo $cuerpo?></p>
</body>
</html>
```

Como estamos viendo, podemos mostrar los contenidos de las variables de configuración enviadas a la vista a través de simples instrucciones echo de PHP. Los datos que se reciben en la vista están disponibles por medio de variables normales, con nombres iguales a los índices del array asociativo que se envió al cargar la vista.

Código PHP dentro de las vistas

Las vistas en CodeIgniter son bastante flexibles y permiten ejecutar cualquier tipo de código PHP que necesitemos. En principio las acciones que necesitaremos codificar dentro de una vista serán bastante simples, como mostrar el contenido de variables, o realizar algún bucle por alguna estructura de datos.

Sin embargo, nada impide escribir código PHP, lo complejo que queramos, con accesos a recursos del servidor, bases de datos, etc. Sin embargo, insisto que si utilizamos correctamente las vistas, según como se debería a en el modelo - vista - controlador, no deberíamos hacer operaciones muy complicadas.

CodeIgniter dispone además de algunas estructuras de código, adicionales a las que existen en el propio PHP, que podemos utilizar dentro de las vistas y que nos pueden ayudar a reducir nuestro código o a esquematizarlo un poco más.

El primero de los códigos alternativos es muy útil, pues se trata de un resumen de la sentencia echo. Por ejemplo, el código siguiente:

```
<?php echo $variable?>
```

Se puede resumir por este otro:

```
<?=$variable?>
```

Del mismo código CodeIgniter dispone de estructuras de control **if, for, foreach y while** que se pueden escribir de una manera resumida, en comparación con las estructuras de control habituales en PHP. Ahora no vamos a explicar la sintaxis de estos elementos, por lo que recomendamos leer la documentación del framework para encontrar más referencias.

Nota: Estas maneras nuevas de hacer bucles y condicionales, en mi opinión, pueden ser útiles para usuarios que no conocen PHP, pero no merece mucho la pena aprenderlas si ya dominamos PHP, como probablemente ocurra si estamos leyendo este manual. Sin embargo, si la responsabilidad de crear las vistas cae sobre un diseñador que no conoce PHP, puede que le sea más sencillo utilizar, por ejemplo, el IF de CodeIgniter en vez del IF típico de PHP.

Todo depende de con qué se sienta más cómodo. La ventaja puede ser la sencillez y que en las estructuras de CodeIgniter no hace falta utilizar las llaves en los bucles o estructuras IF. Sin embargo, existe una desventaja y es que CodeIgniter necesitará preprocesar la vista para convertir las estructuras de código de CodeIgniter en las estándares de PHP y esto llevará un tiempo. Además, cuando tengamos un error de sintaxis en el código de una vista, si estamos utilizando las estructuras de control de CodeIgniter, es muy probable que la línea de código donde PHP informe del error no corresponda con la línea de código en la vista donde hemos hecho algo mal, lo que puede dar algún que otro dolor de cabeza en la etapa de depuración.

Revolver vistas como un dato de tipo string

Cuando cargamos una vista en CodeIgniter la salida producida por ella se envía al usuario visitante una vez hayan sido procesada la función del controlador que invoca las vistas. Este comportamiento es el estándar dentro del framework, pero también podríamos conseguir que el código HTML resultante de cargar una vista se devuelva al controlador para poder almacenarlo en una variable y hacer con él lo que podamos necesitar.

Esto lo conseguimos enviando un tercer parámetro al método que carga una vista, con un valor true, que indica justamente eso, que deseamos que el método devuelva el código procesado por la vista.

```
$cadena = $this->load->view('mivista', '', true);
```

La línea de código anterior provocaría que la carga de la vista se almacene en la variable $cadena. Como se puede ver, al llamar a la vista tenemos que indicar el nombre de la vista, seguido con el nombre del array que contiene los datos para configurar la vista (o comillas comillas si no estamos pasando datos a la vista), y finalmente el valor true como tercer parámetro. En un esquema como este, el contenido de la vista no se enviaría al navegador, a no ser que luego hagamos un echo de la variable $cadena o algo parecido.

*Artículo por **Miguel Angel Alvarez***

Modelos en CodeIgniter

En este artículo, que forma parte del Manual de CodeIgniter, vamos a comenzar con los modelos en CodeIgniter, el tercero y último de los integrantes del patrón de desarrollo utilizado en este framework PHP.

En artículos anteriores ya vimos de manera general qué eran los modelos, cuando explicamos el Modelo - Vista - Controlador (MVC), así que en este caso vamos a ver cómo CodeIgniter los gestiona. No obstante cabe decir que los modelos en el MVC son los módulos que tienen como responsabilidad el acceso y control de los datos que hay en bases de datos y mantienen encapsuladas todas las particularidades y complejidades de los accesos a las tablas para realizar cualquier tipo de operación.

Al centralizarse todas las acciones sobre la base de datos por medio de los modelos, en los controladores podemos olvidarnos de lo compleja o simple que sea nuestra base de datos,

incluso de qué tipo de base de datos tiene el servidor con el que estamos trabajando.

En el patrón MVC, los modelos son una de las partes fundamentales, pero en el caso de CodeIgniter, su uso es sólo opcional. El desarrollador es el responsable de decidir si le viene bien el uso de los modelos o si prefiere realizar sus aplicaciones haciendo cualquier tipo de operación sobre la base de datos en los propios controladores.

Nota: La posibilidad de usar o no modelos depende de nosotros. Esta libertad en cuanto a la elección de nuestras propias costumbres de codificación es muy propia de CodeIgniter, que se vende a si mismo como un framework flexible, que se puede adaptar a la mayoría de los entornos. Nosotros, no obstante, queremosrecomendar su utilización, por lo menos para probar las ventajas de uso y ver si en el futuro nos interesa o no basarnos en ellos para realizar los accesos a la base de datos. Sin duda, si queremos programar "a lo grande", debemos utilizarlos. De hecho ofrecen muchas ventajas para proyectos grandes, como la abstracción de base de datos.

Cómo es un modelo en CodeIgniter

Los modelos en la práctica son clases, de programación orientada a objetos, que tienen sus métodos o funciones, a los que se puede invocar desde los controladores para hacer operaciones con la base de datos.

Los modelos, como decíamos, contendrán una serie de funciones o métodos para realizar las operaciones típicas. Por ejemplo, pensemos en una aplicación que tiene que trabajar con usuarios. Entonces tendremos un modelo de usuarios que tendrá una serie de funciones, como la selección de usuarios, inserción, actualización y borrado. Nuestra aplicación generalmente tendrá varios modelos, para trabajar con cada una de las entidades de nuestra base de datos.

Aquí podemos ver un ejemplo de modelo en CodeIgniter, aunque incompleto, que puede darnos alguna idea. Puede merecer la pena echar un vistazo, a pesar que no lo entendamos del todo. No hay que preocuparse, pues en breve explicaremos cada una de sus partes con detalle.

```
class Usuario_model extends Model {

  function __construct(){
    parent::Model();
  }

  function existe_email($email){
    $this->db->select('email_usuario');
    $this->db->where('email_usuario', $email);
    $query = $this->db->get('usuario');
    if ($query->num_rows() > 0){
      return 1;
    }
    return 0;
  }

  function usuario_login($email){
    $this->db->where('email_usuario', $email);
    //$this->db->where('clave', md5($clave));
    $query = $this->db->get('usuario');
    if ($query->num_rows() > 0){
      return $query->row();
    }
    return 0;
  }

  function inserta_usuario($datos = array()){
```

```
    if(!$this->_required(array("email_usuario","clave"), $datos)){
      return FALSE;
    }
    //clave, encripto
    $datos['clave']=md5($datos['clave']);

    $this->db->insert('usuario', $datos);
    return $this->db->insert_id();
  }
}
```

Bases de la construcción de modelos en CodeIgniter

Los modelos se construyen extendiendo la clase Model y tenemos que nombrarlos con la primera letra en mayúsculas. Dentro del modelo que estamos creando tenemos que definir obligatoriamente un constructor, donde tenemos que hacer una llamada al constructor de la clase de la que hereda (clase parent, llamada Model).

Este sería el modelo más básico, que está vacío, ya que no tiene ninguna función para operar con la base de datos. Mostramos primero el código básico de un modelo en PHP 5, donde los constructores tienen el nombre __construct().

```
class Nombre_model extends Model {
  function __construct(){
    parent::Model();
  }
}
```

Ahora veamos cómo sería el modelo básico en PHP 4, donde los modelos tendrían una pequeña diferencia, dado que los constructores en la versión 4 de PHP tienen siempre el mismo nombre que la propia clase.

```
class Nombre_model extends Model {
  function Nombre_model(){
    parent::Model();
  }
}
```

Los modelos se han de guardar en la carpeta "system/application/models/". El nombre de archivo del código del modelo debe tener extensión .php y se ha de escribir con todas las letras en minúsculas. Por ejemplo nuestro modelo, cuya clase que se llama "Nombre_model", deberíamos guardarlo en la ruta:

system/application/models/nombre_model.php

Si lo deseamos, podemos organizar nuestros modelos por subdirectorios, siempre dentro de la carpeta "system/application/models/". Simplemente cuando los carguemos tendremos que indicar en qué carpeta están localizados.

En el siguiente artículo ampliaremos esta información y mostraremos cómo utilizar los modelos en una aplicación web, cargándolos desde los controladores e invocando sus métodos para el acceso a las tablas. Luego veremos cómo configurar CodeIgniter para decirle cuál es nuestro servidor de base de datos, el nombre de la base de datos a utilizar y otros datos de acceso, para que el propio Framework pueda hacer conexiones desde los modelos.

*Artículo por **Miguel Angel Alvarez***

Utilizar los modelos desde los controladores en CodeIgniter

En el artículo anterior del Manual de CodeIgniter comenzamos a explicar los modelos y cómo gestionar las conexiones a base de datos con ellos. Vimos cómo se construyen los modelos, por lo que ahora pasaremos mostrar cómo utilizarlos desde los controladores.

Cargar un modelo

La carga de un modelo se realiza desde el controlador que lo va a utilizar, por medio del método de carga de modelos, con un código como este:

```
$this->load->model('Nombre_model');
```

Esto supone que el modelo está en el directorio general de los modelos (system/application/models).

Ahora, si pretendiésemos cargar un modelo que está en un subdirectorio, tendríamos que indicar la ruta en la función de carga del modelo.

```
$this->load->model('carpeta_del_modelo/Nombre_model');
```

Si queremos cargar varios modelos de una sola vez, podemos hacerlo pasando un array con todos los modelos que deseamos cargar:

```
$this->load->model(array('Cliente_model','Factura_model'));
```

Una vez cargado un modelo, sus métodos para el acceso a los datos estarán disponibles en nuestro controlador, a través del propio objeto controlador, el nombre del modelo que queremos accionar y la función que queremos invocar dentro del modelo.

```
$this->Nombre_model->funcion_del_modelo();
```

Carga automática de modelos

Si deseamos cargar automáticamente uno o varios modelos cada vez que se inicia cualquier solicitud de página en nuestra aplicación web, es decir, sin que tengamos que cargarlos explícitamente desde los controladores, podemos especificar en el archivo de configuración "autoload.php" cuáles son los modelos que vamos a requerir siempre.

Este archivo de configuración se encuentra en la ruta "system/application/config/autoload.php". Tenemos que modificar la lista de modelos a cargar en la variable de configuración "model", con un código como este:

```
$autoload['model'] = array('Usuario_model', 'Empresa_model');
```

Esto cargaría el modelo de usuarios y el modelo de empresas en todas las páginas de nuestra aplicación web con CodeIgniter.

Ejemplo de controlador que utiliza modelos

Para que se pueda ver mejor cómo utilizar un modelo desde un controlador, veamos un ejemplo. A continuación tenemos un esquema de un controlador que utiliza los modelos para extraer datos de la base de datos y enviarlos a una vista para mostrarlos en la página.

```
class Factura extends Controller {
```

```
  function mostrar_factura($id_factura){
    $this->load->model('Factura_model');
    $factura = $this->Factura_model->dame_factura_id($id_factura);
    $this->load->view('mostrar_factura', $factura);
  }
}
```

Este controlador llamado Factura tiene una función para mostrar una factura, que recibe el identificador de la factura que se desea ver.

En dicha función se carga el modelo adecuado para el trabajo con facturas "Factura_model". Luego llamamos a la función dame_factura_id() del modelo cargado, a la que le pasamos el identificador de la factura que deseábamos ver. La función dame_factura_id() del modelo devuelve un array con los datos de la factura. Luego mostramos los datos de la factura con la vista "mostrar_factura" y con los datos de configuración, que es el array que obtuvimos al invocar la función del modelo.

Con esto hemos podido ver un controlador simplificado, que utiliza tanto un modelo como una vista para mostrar el contenido de una factura que se ha traído de la base datos. Por fin hemos visto todos los componentes del MVC trabajando por separado y coordinados desde el controlador. Espero que hasta este punto haya quedado claro al menos el modo en el que trabajaremos con CodeIgniter, aunque todavía hay muchas cosas en el aire que trataremos de explicar en el futuro.

Conectar con una base de datos

Pero antes de terminar este tema, por si alguien desea hacer sus primeras pruebas para construir un modelo, conviene explicar cómo conectar con la base de datos en CodeIgniter. En realidad existen diversos modos de conexión, automática y manual, pero no los vamos a ver todos ahora.

Veremos simplemente el método más sencillo para hacer la conexión con la base de datos, que es la conexión automática. Ésta la realiza CodeIgniter por nosotros en cada página que se solicite al servidor y tenemos que configurarla en el archivo de configuración autoload.php (system/application/config/autoload.php). En ese archivo veremos un array de librerías que se tienen que incluir en todos los controladores, array "libraries". En ese array tenemos que colocar la librería 'database', de una forma como esta:

```
$autoload['libraries'] = array('database', 'otras_librerias...");
```

Además, tendremos que modificar otro archivo de configuración llamado database.php (system/application/config/database.php), colocando todos los datos de la base de datos que queremos usar: el servidor, usuario, contraseña, base de datos, etc. Por ejemplo, esto sería un ejemplo de configuración de una base de datos que tenemos en local.

```
$db['default']['hostname'] = "localhost";
$db['default']['username'] = "root";
$db['default']['password'] = "";
$db['default']['database'] = "facturacion";
$db['default']['dbdriver'] = "mysql";
$db['default']['dbprefix'] = "";
$db['default']['pconnect'] = TRUE;
$db['default']['db_debug'] = TRUE;
$db['default']['cache_on'] = FALSE;
$db['default']['cachedir'] = "";
$db['default']['char_set'] = "utf8";
$db['default']['dbcollat'] = "utf8_general_ci";
```

*Artículo por **Miguel Angel Alvarez***

Repaso: tutorial para hacer una aplicación de prueba en CodeIgniter

Vamos a dar un primer repaso a lo todo lo que conocemos sobre CodeIgniter, que será suficiente para poder hacer una primera aplicación, sencilla, pero útil para ver en marcha todo lo que hemos ido aprendiendo en el Manual de CodeIgniter.

La idea es que podamos hacer nuestra primera aproximación al trabajo de desarrollo en este framework, basado en el Modelo - Vista - Controlador (MVC).

Haremos un ejemplo de aplicación que mostrará artículos de una base de datos. Lo cierto es que será mucho llamarlo aplicación, porque sólo es una pequeña parte de lo que podríamos necesitar en una aplicación completa, pero esperamos que el ejemplo sea suficientemente didáctico. No obstante, recordar que para entender todos los códigos tendrás que buscar las explicaciones de cada uno de los artículos que hemos visto en este manual de desarrollo web .com.

1.- Crear un controlador

Vamos a crear un controlador para empezar, con un código inicial básico.

```
<?php
class Articulos extends Controller {
  function index(){
    $this->load->view('home');
  }
}
?>
```

Simplemente hemos colocado una función index(), que es la que se llamará cuando se acceda a este controlador tal cual. Dentro de index estamos invocando una vista.

Este controlador lo tenemos que guardar en un archivo llamado articulos.php que meteremos en el directorio de los controladores: system/application/controllers.

2.- Creamos la vista "home"

Ahora vamos a crear la vista "home", que llamamos desde el anterior controlador, con un código como este:

```
<html>
<head>
<title>Portada de mi sitio</title>
</head>
<body>
<h1>Bienvenido a mi web</h1>
<p>Esta es la portada de página web, basada en la publicación de artículos interesantes.</p>
</body>
</html>
```

Guardamos esta vista como "home.php" en el directorio de las vistas: system/application/views.

3.- Configuramos este controlador como controlador por defecto

Ahora podríamos acceder a este controlador por medio de una URL como esta:

http://localhost/index.php/articulos

Deberíamos ver el contenido de nuestra vista, con la bienvenida al sitio. No obstante, yo quiero que este controlador sea el controlador por defecto, para que cuando accedamos a la raíz de la aplicación se muestre ese contenido de bienvenida. Para ello, voy a editar el archivo de configuración routes.php (application/config/routes.php).

Busco la variable de configuración "default_controller" para colocar el nombre de este controlador:

```
$route['default_controller'] = "Articulos";
```

Ahora podremos acceder a la URL raíz de CodeIgniter y ver el mismo mensaje de bienvenida, en una url como esta:

http://localhost/

En el siguiente artículo continuaremos con la creación de esta aplicación web de prueba en CodeIgniter. Veremos cómo crear nuestra base de datos y hacer la primera página que muestre los datos de la tabla de artículos.

*Artículo por **Miguel Angel Alvarez***

Repaso 2: Creamos la base de datos y conectamos desde una página

En el artículo anterior del Manual de CodeIgniter comenzamos un repaso de todo lo que hemos aprendido hasta el momento. Así pues, si no lo hemos hecho, conviene leer las explicaciones de la primera parte de esta aplicación de prueba en CodeIgniter. Anteriormente vimos cómo crear nuestro primero controlador y nuestra primera vista. Ahora comenzaremos a trabajar con los módulos y las bases de datos.

4.- Creo una tabla "articulos" en la base de datos

Como en mi aplicación voy a tener artículos, voy a crear una tabla donde almacenarlos. Utilizaré una base de datos MySQL que tengo en local. Crearla con el método que prefiráis, pero yo utilizo PhpMyAdmin que tengo instalado por defecto con mi paquete Wamp.

Tendré que crear una base de datos que voy a llamar, por ejemplo, "aplicacionarticulos" y luego una tabla llamada "articulo", que tendrá los siguientes campos:

- id (int, auto incremental y clave primaria)
- titulo (varchar 100)
- descripcion (varchar 200)
- cuerpo (text)

Por si sirve de algo, coloco aquí el código SQL del create table.

```
CREATE TABLE `articulo` (
`id` INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY ,
`titulo` VARCHAR( 100 ) NOT NULL ,
`descripcion` VARCHAR( 200 ) NOT NULL ,
`cuerpo` TEXT NOT NULL
)
```

Antes de salir de PhpMyAdmin, o cualquier otro gestor de base de datos, vamos a insertar manualmente unos cuantos registros en la tabla de "articulo" con datos de prueba.

5.- Configuro CodeIgniter para la conexión con esta base de datos

Ahora voy a decirle a CodeIgniter que realice una conexión automática con la base de datos que acabo de crear, que se active en todas las páginas de dentro de mi sitio web. Para ello comienzo editando el archivo de configuración database.php (system/application/config/database.php) donde deben figurarlos datos de la base de datos que deseo utilizar.

Tenemos que editar al menos los siguientes datos:

```
$db['default']['hostname'] = "localhost";
$db['default']['username'] = "root";
$db['default']['password'] = "";
$db['default']['database'] = "aplicacionarticulos";
```

Ahora debo decirle a CodeIgniter que conecte automáticamente, incluyendo siempre las librerías de conexión con base de datos. Para esto tenemos que editar el archivo de configuración autoload.php (system/application/config/autoload.php), en el lugar donde se indica qué librerías queremos cargar siempre.

```
$autoload['libraries'] = array('database');
```

6.- Creo un primer modelo de artículos

Ahora voy a crear una primera aproximación al modelo de artículos. Por el momento sólo voy a colocar una función para traer los últimos 5 artículos.

> **Nota**: El código que veremos a continuación es PHP 5. PHP 4 tendría alguna diferencia por los nombres de los constructores, como ya se explicó.

```
<?php
class Articulo_model extends Model {

  function __construct(){
    parent::Model();
  }

  function dame_ultimos_articulos(){
    $ssql = "select * from articulo order by id desc limit 5";
    return mysql_query($ssql);
  }
}
?>
```

> **Nota**: En este modelo estamos utilizando las propias funciones de acceso a base de datos de MySQL.

Simplemente comentar que en CodeIgniter existe un mecanismo de abstracción de base de datos que se podría utilizar para escribir código con funciones que sirvan para cualquier tipo de base de datos que podamos utilizar, y no sólo MySQL.

Este modelo lo guardamos en el archivo "articulo_model.php" dentro de la carpeta de los modelos (system/application/models).

7.- Actualizo el controlador

Ahora vamos a cambiar un poco el código de nuestro controlador inicial, para colocar algunas otras cosas que nos servirán para dos cosas:

- Decirle que voy a trabajar con el modelo de artículos
- Enviarle a la vista de la portada ("home") los datos de los últimos artículos publicados
- Además, de paso voy a cargar un helper de URL, que tiene una función que me servirá de utilidad dentro de la vista para construir las URL para ver los artículos.

```
<?php
class Articulos extends Controller {
  function index(){
    //cargo el helper de url, con funciones para trabajo con URL del sitio
    $this->load->helper('url');

    //cargo el modelo de artículos
    $this->load->model('Articulo_model');

    //pido los ultimos artículos al modelo
    $ultimosArticulos = $this->Articulo_model->dame_ultimos_articulos();

    //creo el array con datos de configuración para la vista
    $datos_vista = array('rs_articulos' => $ultimosArticulos);

    //cargo la vista pasando los datos de configuacion
    $this->load->view('home', $datos_vista);
  }
}
?>
```

Nota: El código está comentado y si se encuentran dudas recomendamos releer este Manual de CodeIgniter. No obstante, en el código anterior hay una cosa que son los "helpers" que no hemos explicado todavía. Son simplemente librerías de código con funciones que resultarán útiles a la hora de hacer aplicaciones web con CodeIgniter, que están clasificadas por temática. Nosotros estamos cargando el helper "url", que contiene unas serie de funciones para trabajar con URLs. En concreto utilizaremos la función site_url() del helper "url", un poco más adelante.

8.- Actualizo la vista "home"

Ahora vamos a cambiar el código de nuestra vista de portada, puesto que queremos que nos muestre también los últimos artículos publicados en el sitio, cada uno con un enlace a la URL donde se muestre cada artículo.

```
<html>
<head>
<title>Portada de mi sitio</title>
</head>
<body>
<h1>Bienvenido a mi web</h1>
```

```
<p>Estos son los últimos artículos publicados.</p>
<?php
while ($fila = mysql_fetch_array($rs_articulos)){
  echo '<p>';
  echo '<a href="' . site_url('/articulos/muestra/' . $fila['id']) . '">' . $fila['titulo'] . '</a>';
  echo '</p>';
}
?>

</body>
</html>
```

Podemos ver cómo hacemos un recorrido a los registros de últimos artículos traídos desde la base de datos con ayuda del modelo y enviados a la vista por medio del controlador. Además, vemos que se está utilizando la función site_url() que pertenece al helper "url" que habíamos cargado en el controlador.

> **Nota**: Aquí también estamos utilizando directamente funciones de MySQL. Sin embargo, CodeIgniter tiene algunas librerías que nos ayudarían a hacer esto de otra manera. No obstante, por ahora queremos limitarlos a lo que conocemos del framework.

Una vez actualizada la vista, podemos acceder a la página raíz de nuestra aplicación, que debería mostrar los últimos artículos que habían cargados en la base de datos, en la tabla articulo, con un enlace a varias páginas, donde se muestra cada uno de los artículos (no creadas todavía). Por supuesto, tenéis que haber insertado algún artículo en la tabla para que lo muestre.

Tenemos la primera parte de nuestra aplicación con acceso a base de datos, creada en pocos minutos con CodeIgniter. Pero todavía tenemos que complicarlo un poco más. En el próximo artículo haremos un poco más grande esta aplicación web.

*Artículo por **Miguel Angel Alvarez***

Repaso 3: Creamos páginas para mostrar artículos

Cabe señalar antes que nada que este artículo es la continuación de una serie de entregas del Manual de CodeIgniter, en el que estamos haciendo un repaso de lo que llevamos aprendido para hacer una aplicación web. Si te interesa y no has leído los repasos anteriores, debes comenzar por el artículo

En las próximas modificaciones o mejoras en el código de nuestra aplicación de artículos pretendo hacer la parte en la que se muestran los artículos completos, es decir, las páginas que mostrarán cada uno de los artículos que tenemos en nuestra base de datos, con su texto completo.

En principio, los enlaces para acceder a estas páginas están en la portada de nuestra aplicación y los hemos hecho en la vista "home", creada anteriormente. Podemos observar que los enlaces de la portada, que nos mostraban los últimos 5 artículos publicados, van a URLs como estas:

http://localhost/index.php/articulos/muestra/1
http://localhost/index.php/articulos/muestra/4

Como se puede ver, tenemos una URL que pasa por el directorio artículos (el controlador de artículos), el directorio muestra (que es la función que tenemos que realizar en el controlador

para atender esta URL) y el número del identificador del artículo que deseamos ver (que es el parámetro que se enviará a la función muestra() del controlador).

Así pues, tenemos que hacer varias cosas para que esas URL funcionen y muestren el contenido de los artículos.

9.- Ampliación del modelo

Ahora vamos a hacer el modelo para que tenga una función que devuelva los datos de un artículo cuyo identificador pasaremos por parámetro. La nueva función devolverá un array con los datos del artículo encontrado, o false en caso que no encuentre el artículo.

```
<?php
class Articulo_model extends Model {

  function __construct(){
    parent::Model();
  }

  function dame_ultimos_articulos(){
    $ssql = "select * from articulo order by id desc limit 5";
    return mysql_query($ssql);
  }

  function dame_articulo($id){
    $ssql = "select * from articulo where id=" . $id;
    $rs = mysql_query($ssql);
    if (mysql_numrows($rs)==0){
      return false;
    }else{
      return mysql_fetch_array($rs);
    }
  }
}
?>
```

Como se puede ver, es exactamente el modelo anterior, en el que hemos creado la función dame_articulo().

10.- Ampliación del controlador

En este momento queremos que el controlador de artículos también muestre las páginas con los artículos, que tendrán URL como estas:

http://localhost/index.php/articulos/muestra/3

Así pues, tenemos que crear la función muestra() en el controlador, que recibe un parámetro que es el id del artículo que se desea ver. Esta función tendrá que solicitar al controlador los datos del artículo en cuestión y enviarlos a una vista para que lo muestre. Además, en caso que se intente acceder a artículos que no existen en la base de datos, tenemos que mostrar un error 404 de página no encontrada.

Pongo el código del controlador, tal como quedaría después de incorporar esta nueva función.

```
<?php
class Articulos extends Controller {
  function index(){
    //cargo el helper de url, con funciones para trabajo con URL del sitio
    $this->load->helper('url');
```

```
    //cargo el modelo de artículos
    $this->load->model('Articulo_model');

    //pido los ultimos artículos al modelo
    $ultimosArticulos = $this->Articulo_model->dame_ultimos_articulos();

    //creo el array con datos de configuración para la vista
    $datos_vista = array('rs_articulos' => $ultimosArticulos);

    //cargo la vista pasando los datos de configuacion
    $this->load->view('home', $datos_vista);
  }

  function muestra($id){
    //cargo el helper de url, con funciones para trabajo con URL del sitio
    $this->load->helper('url');

    //cargo el modelo de artículos
    $this->load->model('Articulo_model');

    //pido al modelo el artículo que se desea ver
    $arrayArticulo = $this->Articulo_model->dame_articulo($id);

    //compruebo si he recibido un artículo
    if (!$arrayArticulo){
      //no he recibido ningún artículo
      //voy a lanzar un error 404 de página no encontrada
      show_404();
    }else{
      //he encontrado el artículo
      //muestro la vista de la página de mostrar un artículo pasando los datos del array del artículo
      $this->load->view('muestra_articulo', $arrayArticulo);
    }
  }
}
?>
```

Nos tenemos que fijar en la nueva función creada: muestra(). En esta función cargamos el helper de URL (aun no hemos hablado de los helpers, pero lo veremos pronto). Luego cargamos el modelo de artículos y solicitamos el artículo al modelo. Luego comprobamos si existe el artículo. Si no existe, lanzamos un error 404, con la función show_404() (tampoco hemos visto aun la gestión de errores). Si existe, llamamos a la vista que debe mostrar el contenido completo del artículo, enviando el array con los datos del artículo encontrado.

Nos fijamos que la vista que solicitamos "muestra_articulo" no ha sido creada todavía.

11.- Creo la vista que muestra el artículo

Para acabar esta etapa de la construcción de nuestra web, vamos a crear la vista "muestra_articulo", que tiene que mostrar el contenido del artículo al usuario. Tiene un código muy sencillo.

```
<html>
<head>
<title><?=$titulo?></title>
</head>
<body>
<h1><?=$titulo?></h1>
<blockquote><b><?=$titulo?></b></blockquote>
<?=nl2br($cuerpo)?>
<p><a href="<?=site_url()?>">Volver</a></p>
```

```
</body>
</html>
```

Como se puede ver, se muestran todos los datos del artículo, con su título, descripción y cuerpo completo. Luego tenemos un enlace de volver que nos llevará a la portada del sitio, cuya URL obtenemos con la función site_url() que tenemos disponible al haber cargado el helper "url" en el controlador.

Ahora podemos comprobar, entrando a la portada del sitio, que los enlaces a los últimos artículos ya están funcionando. Si los pulsamos accederemos a las páginas que acabamos de hacer en esta etapa, en la que se nos muestra el contenido de los artículos. Si intentamos acceder a una URL de un artículo que no existe, como:

http://localhost/codeigniter/index.php/articulos/muestra/923

Podremos visualizar un error 404 de página no encontrada.

Hasta aquí tenemos nuestra aplicación que muestra artículos a sus visitantes, cargados en una base de datos. Es una aplicación incompleta, puesto que a poco que nos imaginemos, se nos ocurrirán multitud de cosas para mejorarla. Sin embargo, nuestro objetivo era más bien que los lectores sean capaces de entender el patrón de desarrollo de CodeIgniter, el MVC.

Quizás penséis que, siguiendo vuestras costumbres habituales de desarrollo en PHP, podríais haber hecho esta pequeña aplicación en menos tiempo y quizás con menos código fuente. No obstante, a medio plazo observaremos los beneficios de codificar por medio de los controladores, vistas y módulos, con los que obtendremos código más claro, organizado, y con mantenimiento más sencillo, entre diversas ventajas.

*Artículo por **Miguel Angel Alvarez***

Helpers en CodeIgniter

Los helpers son una de las dos bibliotecas de código disponibles en CodeIgniter y la más sencilla de manejar en un principio, puesto que son funciones que están a nuestra disposición sin depender de ningún objeto. Lo veremos con detenimiento en este artículo del Manual de CodeIgniter.

En cualquier framework PHP, como nos podremos imaginar, se pone a nuestra disposición una biblioteca de funciones que resuelven las cosas típicas que podemos necesitar al realizar una aplicación web. De hecho, una de las ventajas de usar frameworks es justamento no tener que programar por nosotros mismos todas las funciones que se utilizan reiteradas veces en la mayoría de los sitios web profesionales. Así pues, los helpers son una de las bibliotecas de funciones que supongo esperábamos que CodeIgniter nos proporcionara.

Podemos decir que los helpers son juegos de funciones, ordenados por temática en diferentes paquetes, que nos ayudan en la realización de tareas habituales en las aplicaciones web. Existen helpers para muchas cosas distintas, como para trabajo con arrays, fechas, cookies, emails, URLs, formularios, etc.

Nota: En CodeIgniter disponemos de dos tipos de bibliotecas de códigos, Helper y Class. Los Helpers son funciones propiamente dichas y las Class son clases de programación orientada a objetos (POO). Su carga y utilización son ligeramente distinta, así que las veremos por separado. De momento quedémonos con que los helpers no dependen de programación orientada a objetos, sino que son funciones (de programación funcional o prodecimental, más tradicional que la POO) que podremos utilizar para solucionar temas

concretos y cuyo uso es independiente.

Los helpers están almacenados en el directorio system/helpers o en el directorio system/application/helpers. Al intentar cargar un helper (que se tiene que realizar de manera explícita, como veremos más adelante) CodeIgniter mirará si se encuentra primero en system/application/helpers y si no es así lo buscará en la carpeta system/helpers.

Carga de un helper en CodeIgniter

En CodeIgniter no se carga de manera predeterminada ninguna librería o juego de funciones (salvo algunas librerías muy básicas), para aumentar el rendimiento del sistema, no haciendo procesar a PHP nada que realmente no necesite. Por ello tenemos que cargar los helpers, así como cualquier otra librería explícitamente, ya sea en todas las páginas a través de los archivos de configuración, o en una página en concreto a través de las funciones de carga de helpers.

Una vez un helper ha sido cargado, estará a nuestra disposición en los controladores y vistas, de manera global. Para cargar un helper podemos invocar este código.

```
$this->load->helper('mihelper');
```

En el código anterior la cadena "mihelper" corresponde con el nombre del helper que se desea cargar. Este nombre es una palabra como "url", "file", etc. y no el propio archivo del código del helper. Por ejemplo con el código:

```
$this->load->helper('url');
```

Se está cargando el helper con nombre "url" que realmente está en el archivo "url_helper.php". Pero para cargarlo simplemente tenemos que indicar el nombre del helper y no el nombre del archivo donde está el helper.

Puedes cargar un helper en cualquier parte del código de tus controladores o incluso de las vistas (aunque no sea una buena práctica cargar helpers en las vistas), siempre y cuando lo cargues antes de utilizarlo. En un controlador podrías cargar el helper en el constructor, con lo que estaría disponible en todas las funciones del mismo, o bien en una de las funciones donde lo necesites.

Puedes cargar múltiples helpers a través de una única llamada a load().

```
$this->load->helper( array('cookie', 'file', 'form') );
```

Carga automática de helpers

En CodeIgniter tenemos a disposición los archivos de configuración para definir cosas como la carga automática de helpers, que estarán disponibles de manera predeterminada en toda la aplicación sin que los tengamos que cargar cada vez que los pretendamos utilizar. La carga automática nos ahorrará repetir el código de carga de los helpers que deseemos en toda la aplicación y como podremos imaginar, es una buena idea cuando pensamos usar un helper en varios sitios distintos.

El archivo de configuración que tenemos que editar para definir los helpers que deseamos cargar de manera predeterminada es el siguiente:

```
system/application/config/autoload.php
```

Para indicar qué helpers queremos cargar tenemos que editar el array $autoload, en su índice

"helpers", con un código parecido al siguiente:

```
$autoload['helper'] = array('url', 'file', 'cookie');
```

Extender helpers en CodeIgniter

Los helpers se pueden extender con nuevas funcionalidades personalizadas por nosotros mismos. Para ello tienes que crear un archivo con el helper modificado con tus propias funciones y colocarlo en el directorio system/application/helpers y nombrarlo como el helper original pero con el prefijo "MY_".

En ese archivo modificado del helper podrás añadir las funciones que te hagan falta o incluso modificar el código de las funciones ya existentes en los propios helpers.

La lista completa de helpers de CodeIgniter

Podemos saber los helpers que se encuentran a nuestra disposición en la guía del usuario de CodeIgniter, que está en la propia página del framework. Aunque esa guía de usuario está en Inglés, no resulta difícil de seguir y podrás encontrar cada helper y sus funciones relatadas con bastante detalle. En el futuro mostraremos algunos ejemplos de uso de al menos los helpers más fundamentales.

*Artículo por **Miguel Angel Alvarez***

Ejemplo de Helper en CodeIgniter

El Helper de URL es una de las librerías de funciones más fundamentales de CodeIgniter, que quizás utilicen todas las aplicaciones web que puedas construir con el framework, por muy sencillas que sean.

El URL Helper contiene una serie de funciones que nos servirán a trabajar con URLs en las aplicaciones web, para realizar acciones tan básicas como generar URLs completas de páginas de la aplicación web a partir de una ruta sencilla, la obtención de la URL base se nuestra aplicación, la URL actual, etc.

Este artículo pretende servir de guía de uso de Helpers en general, aunque habría que comentar que, en el Manual de CodeIgniter, ya hemos publicado alguna información previa sobre los helpers, que deberías conocer.

> **Nota:** Para que quede claro hasta que punto es básico este Helper de URL cabe decir que, hasta donde hemos explicado en este Manual de CodeIgniter, ya lo hemos utilizado en el primer ejemplo de aplicación sencilla. Ese ejemplo está dividido en tres artículos y en el segundo, titulado Creamos la base de datos y conectamos desde una página se puede encontrar el uso del helper URL.

Controlador para probar el helper

Vamos a crear en este artículo un controlador que hace uso del helper URL, con un par de usos del mismo para construir enlaces y ver la URL principal de la aplicación web. Haremos un controlador bastante simple, para centrarnos más bien en el uso del helper.

Como cualquier controlador, debe extender la clase Controller.

```
class Probando_helper_url extends Controller {
  //código del controlador
}
```

Cargar el helper desde el constructor del controlador

Como hemos dicho cuando se hablaba de los helpers en el artículo anterior, tenemos que cargar explícitamente el Helper de URL para poder utilizarlo. Esto lo podemos hacer desde el archivo de configuración autoload.php, con lo que tendremos el helper cargado en cualquier lugar de la aplicación, o bien desde el controlador donde lo necesitemos. En este caso vamos a ver un controlador que carga el helper.

En el caso de cargar el helper desde el controlador, lo podemos hacer desde el constructor, para que esté accesible desde cualquier función, o desde una de las funciones del controlador. Veremos cómo cargarlo desde el constructor.

```
function __construct(){
  parent::Controller();

  //cargo el helper de url
  $this->load->helper('url');
}
```

> **Nota:** Como se puede ver, el método de realización del constructor es de PHP 5. Recordar que en PHP 4 el constructor es una función con el mismo nombre que la clase y en PHP 5 el constructor es una función con el nombre __construct().

Dentro de los constructores de controladores se debe llamar primeramente al constructor de la clase padre (de la que heredamos), con parent::Controller().

Luego hacemos la carga del helper URL con la llamada al método helper() de la clase load que forma parte del controlador, indicando entre paréntesis el helper que deseamos cargar: 'url'.

Uso el helper en las funciones de los controladores

Ahora que ya he cargado el helper en el constructor, en todo el código del controlador tendré disponibles, como globales, todas las funciones del helper URL. (También en las vistas que se carguen desde el controlador, aunque nosotros no vamos a hacer en este ejemplo ninguna vista)

Veamos entonces un ejemplo de función del controlador.

```
function index(){
  //escribo desde el controlador, aunque debería hacerlo desde la vista
  echo "<h1>Probando helper URL</h1>";

  //genero el enlace de este controlador, para la función creada muestra_base_url()
  $enlace = site_url("probando_helper_url/muestra_base_url");
```

```
  //escribo un enlace con esa función del controlador
  echo '<a href="' . $enlace . '">Muestra la URL base</a>';
}
```

Este es el método index() que es el que se invoca por defecto en el controlador si no se especifica ningún otro en la solicitud de URL. Podemos ver el uso de las funciones del helper de URL en la línea:

```
$enlace = site_url("probando_helper_url/muestra_base_url");
```

Con esta línea invocamos la función site_url() del helper URL, que sirve para obtener una URL del sitio a partir de una ruta parcial. Esta función es útil porque sirve para saber cómo es la URL completa con la que hay que enlazar distintas páginas del sitio, que podría variar dependiendo del lugar donde se haya instalado CodeIgniter, sólo a partir de un segmento que no puede cambiar. Dicho de otra manera, al enlazar con otras páginas del sitio conviene componer las URL siempre con la función site_url(), para que esos enlaces sigan funcionando aunque la aplicación la cambiemos a otro directorio, otro dominio, etc.

> **Nota:** Un detalle que queremos señalar también sobre la función anterior, index(), es que hacemos uso de la sentencia echo de PHP para escribir cosas en la página directamente con el controlador. Esto no es recomendable, aunque se pueda hacer en CodeIgniter, pues la salida hacia la página desde nuestros scripts debería estar siempre en las vistas.

Ahora podemos ver un segundo método del controlador con otro uso de las funciones del helper URL.

```
function muestra_base_url(){
  //escribo desde el controlador, aunque debería hacerlo desde la vista
  echo base_url();

  //un enlace para volver
  echo '<p><a href="' . site_url('probando_helper_url') . '">Volver</a></p>';
}
```

Esa función escribe por pantalla lo que devuelve la función base_url(), que pertenece al helper de URL y sirve para informar de la ruta raíz de la aplicación CodeIgniter, que también dependerá del dominio donde estemos trabajando y el directorio donde se haya instalado el framework.

Luego también escribe un enlace para volver a la página anterior y para crear la ruta del enlace también utilizamos el helper de URL con la función site_url() relatada anteriormente, indicando en este caso tan solo el nombre del controlador al que queremos dirigir el enlace.

Código completo del controlador

Ahora veamos el código del controlador completo que hace uso del helper URL:

```
class Probando_helper_url extends Controller {

  ////////////////////////////////////////////////////////////////////////
  //Constructor
  function __construct(){
    parent::Controller();

    //cargo el helper de url
    $this->load->helper('url');
  }

////////////////////////////////////////////////////////////////////////
//método index, función por defecto del controlador
function index(){
  //escribo desde el controlador, aunque debería hacerlo desde la vista
  echo "<h1>Probando helper URL</h1>";

  //genero el enlace de este controlador, para la función creada muestra_base_url()
  $enlace = site_url("probando_helper_url/muestra_base_url");

  //escribo un enlace con esa función del controlador
  echo '<a href="' . $enlace . '">Muestra la URL base</a>';
}

  ////////////////////////////////////////////////////////////////////////
  //funcion muestra_base_url, para mostrar la URL principal de esta aplicación web
  function muestra_base_url(){
    //escribo desde el controlador, aunque debería hacerlo desde la vista
    echo base_url();

    //un enlace para volver
    echo '<p><a href="' . site_url('probando_helper_url') . '">Volver</a></p>';
  }

  ////////////////////////////////////////////////////////////////////////
  //funcion muestra_url_actual, para mostrar la URL actual de esta página
  function muestra_url_actual(){
    //escribo desde el controlador, aunque debería hacerlo desde la vista
    echo current_url();

    //un enlace para volver
    echo '<p><a href="' . site_url('probando_helper_url') . '">Volver</a></p>';
  }
}
```

En este código hemos agregado al controlador un método llamado muestra_url_actual(), para probar la función del helper URL current_url(), que sirve para obtener la URL actual de la página que se está ejecutando.

Recordar que todo lo visto en este artículo sirve para entender el uso de los helpers en CodeIgniter, aunque nos hayamos saltado algunas recomendaciones del trabajo con el framework, como escribir contenido en la página directamente con el controlador y sin utilizar las vistas. Así mismo, es importante señalar que el helper URL tiene otras funciones también bastante útiles que seguramente nos vendrá bien conocer y que podemos revisar en la documentación del framework.

*Artículo por **Miguel Angel Alvarez***

Librerías de CodeIgniter

CodeIgniter, el más accesible de los frameworks PHP populares, tiene bibliotecas de código para realizar diversas cosas interesantes y útiles. Por un lado tenemos los Helpers, que implementan juegos de funciones y que ya relatamos en artículos anteriores del Manual de CodeIgniter. Por otro lado tenemos las librerías, que vamos a tratar en este artículo.

Las librerías en CodeIgniter son clases de programación orientada a objetos (POO) preparadas para realizar tareas típicas en el desarrollo de páginas web. Implementan clases, de POO, para resolver problemas muchas veces similares a los que resuelven los Helpers, pero son un poco más especializadas en temas particulares. Algunos ejemplos de librerías que se dispone en CodeIgniter son para trabajar con bases de datos, FTP, upload de archivos, sesiones, calendario, etc.

Así pues, podemos decir que las liberías (libraries en inglés) son uno de los componentes que más tiempo nos ahorrarán a la hora de desarrollar una web, pues contienen código que resultará siempre útil. El trabajo con las librerías es bastante simple, como veremos a continuación.

Carga de librerías

Para cargar una library en CodeIgniter tenemos que indicarlo explícitamente, pues en este framework no se carga nada por defecto, salvo contadas excepciones, para optimizar el tiempo de respuesta de PHP al procesar las páginas. Para ello tenemos que invocar el siguiente código:

```
$this->load->library('nombre de la clase');
```

Donde 'nombre de la clase' es el nombre de la librería, o clase de POO, que queremos cargar.

> **Nota:** Realmente todas las librerías no se necesitan cargar, puesto que hay algunas que ya se encuentra por defecto disposición de nuestros scripts, sin tener que cargarlas explícitamente. Esto es porque algunas librerías son fundamentales para el funcionamiento del núcleo de CodeIgniter. Por ejemplo, siempre tendremos disponibles las librerías que implementan las clases de Output, Input o URI.

Por ejemplo, si quisiéramos cargar la librería de Email, que facilita el envío de correos por medio de PHP, tendríamos que utilizar el siguiente código:

```
$this->load->library('email');
```

Ya que son clases de POO, una vez cargada tenemos que utilizar sus métodos para poner en marcha sus funcionalidades. Dentro de los controladores, para acceder a la clase y sus métodos utilizamos el nombre de la clase como una propiedad de $this.

Por ejemplo, en el caso de la librería email, para definir quién envía un correo electrónico utilizamos un código como este:

```
$this->email->from('tu@dominio.com', 'Nombre del remitente');
```

En la guía de usuario de CodeIgniter existe una referencia completa sobre cada una de las clases y su utilización. Nosotros iremos viendo ejemplos de uso en futuros artículos.

Carga automática de librarías en CodeIgniter

Como ya sabemos, antes de utilizar una clase de una de las librerías de CodeIgniter, tenemos que cargarla en el controlador. No obstante, también tenemos la opción de indicar en nuestros archivos de configuración la carga predeterminada de algunas librerías, para que estén disponibles en todas las páginas sin necesidad de cargarlas en los controladores.

Para indicar las librerías que queremos cargar automáticamente tenemos que editar el archivo autoload.php, que está en la ruta:

```
system/application/config/autoload.php
```

En ese archivo encontraremos un array llamado libraries, donde podemos indicar todos los nombres de las librerías a cargar, por ejemplo:

```
$autoload['libraries'] = array('database', 'session', 'form_validation');
```

Para ilustrar cómo se deben utilizar las librerías de CodeIgniter hemos publicado un ejemplo de uso de la librería Calendar.

*Artículo por **Miguel Angel Alvarez***

Ejemplo de librería en CodeIgniter: Calendar Library

En el artículo anterior del Manual de CodeIgniter ya explicamos qué eran las librerías y como cargarlas desde los controladores o desde los archivos de configuración. En este caso vamos a mostrar un ejemplo de uso de librerías, que nos vendrá bien para aprender a manejar de manera general las bibliotecas de clases, de programación orientada a objetos, disponibles en este framework PHP.

Como ejemplo de uso vamos a utilizar la clase Calendar, disponible en la library con el mismo nombre, que sirve para implementar un calendario. Esta clase, como casi todas las que nos ofrecen las librerías, tenemos que cargarla explícitamente en el controlador que queramos utilizarla.

Cargo la librería desde el constructor del controlador

La librería la puedo cargar en cualquier lugar, siempre que se haya cargado antes de intentar utilizarla. Ahora bien, podría cargarla en el archivo de configuración autoload.php, con lo que estaría disponible en todos los controladores, o en un controlador en concreto donde pensemos utilizarla.

En este caso la voy a cargar en un controlador dado y todavía aquí tendría dos posibilidades. Cargarla en una función concreta del controlador o en el constructor. Si la cargo en el constructor tendré la ventaja que estará disponible en todas las funciones de nuestro controlador, lo que puede venirnos bien para ahorrar código, si es que pensamos utilizar esa librería en todo el controlador.

```
class Probando_library_calendar extends Controller {
  function __construct(){
    parent::Controller();

    //cargo la librería Calendar
    $this->load->library('calendar');
```

```
  }
}
```

Como se ha visto, tengo el controlador y dentro tengo el constructor que carga la librería Calendar. Ahora la clase "calendar" estará disponible en todos los métodos de nuestro controlador.

Genero un calendario del mes y año actuales

Esta clase permite escribir calendarios de un mes y un año dados en la página. Para ello tiene un método llamado generate() que recibe los datos del calendario a generar y devuelve el código HTML de ese calendario. Si generate() no recibe parámetros, simplemente muestra el calendario en el mes y año actuales.

Veamos un método index() del controlador, que mostraría el calendario del mes y año actuales.

```
function index(){
  //muestro el calendario del mes actual
  echo $this->calendar->generate();
}
```

> **Nota:** Aunque ya hemos dicho esto en muchas partes, merece la pena repetirlo. Todos los echos o funciones que escriben salida en la página deberían estar en las vistas. Aquí tenemos un echo que escribe en la página directamente desde el controlador, y aunque CodeIgniter lo permite, no es lo correcto.

Ahora podríamos acceder a ese método con una URL como esta:

http://localhost/index.php/probando_library_calendar/

Al indicar simplemente el nombre del controlador en la URL, se llama siempre al método index() que lo que hace es escribir el calendario del mes actual.

Genero un calendario de un mes y un año cualquiera

Veamos un segundo método en este controlador un poco más elaborado, que nos permitiría mostrar calendarios de cualquier mes y año. Para ello vamos a hacer que el método reciba dos parámetros, con el mes y el año solicitados.

```
function calendario($ano, $mes){
  echo $this->calendar->generate($ano, $mes);
}
```

Esta función en el controlador se accede por medio de URLs como esta:

http://localhost/index.php/probando_library_calendar/calendario/2010/1

Con esa URL mostraría el calendario de enero de 2010.

Genero enlaces dentro del calendario

Para acabar vamos a mostrar cómo generar unos links en días concretos del calendario, como si fueran citas que se pueden pulsar para obtener más información.

```
function calendario_enlaces($ano, $mes){
  $data = array(
    1 => 'http://www.desarrolloweb.com/manuales',
    10 => 'http://www.desarrolloweb.com/php',
```

```
    12 => 'http://www.desarrolloweb.com/javascript',
    21 => 'http://www.desarrolloweb.com/css',
  );
  echo $this->calendar->generate($ano, $mes, $data);
}
```

Este método pone unos enlaces en días fijos marcados en el array $data. Podríamos acceder a una página que se procesa con ese método por medio de una URL como esta:

http://localhost/index.php/probando_libraryzx_calendar/calendario_enlaces/2010/2

+

Conclusión sobre la library Calendar

Esperamos que haya sido interesante e ilustrador este ejemplo de uso de librerías y que se haya visto que con poco esfuerzo podemos utilizar componentes bastante complejos para páginas web.

La librería de CodeIgniter para generación de calendarios todavía tiene otras opciones de configuración, que nos servirán de utilidad para implementar funcionalidades sobre los calendarios, como movernos entre meses, modificar su aspecto, idioma, etc. Para mayores referencias podéis consultar la guía de uso de CodeIgniter.

*Artículo por **Miguel Angel Alvarez***