FlavorPHP

versión 1.0

Manual

because PHP should have a better taste

http://www.flavorphp.com/

Índice general

Índice de cuadros

Índice de códigos fuente

Capítulo 1

FlavorPHP

1.1. Introducción

Este manual esta basado en la versión 1.0 de **FlavorPHP**.

FlavorPHP es un framework PHP5 (con licencia MIT) basado en el paradigma MVC jerárquico (HMVC, por sus siglas en inglés *Hierarchical-Model-View-Controller*), **FlavorPHP** fue creado por Pedro Santana [*http://www.pecesama.net/*] y actualmente cuenta con el siguiente equipo de desarrollo:

- **Pedro Santana**. Líder del proyecto. [http://www.pecesama.net/]
- **Victor Bracco**. Desarrollador. [http://victorbracco.com/]
- **Victor de la Rocha**. Desarrollador [http://www.mis-algoritmos.com/]
- **Jorge Condomi**. Desarrollador [http://www.jorgecondomi.com/]
- **Aaron Munguia**. Desarrollador
- **Pablo Cendejas**. Administrador de la documentación.

1.1.1. Sobre FlavorPHP

Como ya se mencionó anteriormente **FlavorPHP** es un framework MVC jerárquico desarrollado en PHP5, el cual surge en base a la experiencia previa de sus autores en el desarrollo de proyectos Web 2.0 (algunos ejemplos de estos proyectos http://gelatocms.com/ y http://sabros.us/), esta basado en las mejores prácticas de desarrollo web, proveyendo eficiencia en la creación y en el mantenimiento de aplicaciones Web.

FlavorPHP incluye las siguientes características:

- Separación de la aplicación en diferentes capas.
- Sistema de plantillas (themes) y vistas fácil de usar.
- Uso del patrón Active Record para implementar ORM.
- Desarrollo de aplicaciones multi-idiomas.
- Sistema de paginación de datos rápido de implementar.
- Soporte para AJAX y efectos JavaScript usando jQuery.

1.1.2. Prerrequisitos de FlavorPHP

FlavorPHP tiene un número muy pequeño de prerrequisitos pero que deben cumplirse para que funcione correctamente, estos se listan a continuación:

- PHP5 (recomendado 5.2).
- Servidor Web Apache con mod_rewrite instalado.
- Servidor de Base de Datos MySQL.

1.1.3. Instalación de FlavorPHP

FlavorPHP es distribuido en un paquete descargable desde su sitio web http://www.flavorphp.com/.

Este paquete debe descomprimirse completo en el directorio deseado de su servidor web. El directorio creado puede ser renombrado al nombre que desee, por ejemplo el nombre del proyecto, o bien puede descomprimir **FlavorPHP** directamente al directorio raíz de su servidor web.

1.2. Iniciando con FlavorPHP

Para iniciar a crear nuestros proyectos con **FlavorPHP**, debemos primero conocer su estructura y como configurarlo.

1.2.1. Patrón de diseño MVC (Modelo - Vista - Controlador)

En 1979, Trygve Reenskaug desarrolló una arquitectura para crear aplicaciones interactivas. En este diseño existian tres partes: modelos, vistas y controladores.

El modelo MVC permite hacer la separación de las capas de interfaz, modelo y lógica de control de esta. **FlavorPHP** esta desarrollado utilizando este patrón de diseño, por lo cual cuenta con una estructura de directorios especial, la cual analizaremos a continuación.

1.2.2. Estructura de FlavorPHP

Una vez instalado **FlavorPHP** veremos una estructura de directorios especial. El contenido que tiene cada directorio se analiza en la tabla 1.1

Al respetar y seguir esta estructura de archivos y derectorios podremos crear nuestra aplicación usando convención sobre configuración, de esta forma al tener todo en su lugar es más sencillo encontrar problemas y mantener la aplicación de forma rápida y sencilla.

1.2.3. Configuración de FlavorPHP

FlavorPHP esta pensado para que se tenga que configurar lo menor posible, por lo que todas las configuraciones las encontraremos en un mismo archivo *./config.php* (ver listado 1.1). En este archivo configuraremos nuestra conexión al servidor de Bases de Datos, así como la ruta base de nuestro proyecto.

Código 1.1: config.php

```
//requiere de BD
define('requiresBD', true);
//conector de BD.
define('DB_Engine', 'mysqli');
```

Cuadro 1.1: Estructura de directorios

Directorio	**Contenido**
controllers	Este directorio contendrá todos nuestros controladores.
flavor	Este directorio contiene el nucleo de **FlavorPHP**.
languages	Este directorio contendrá las traducciones de nuestra aplicación.
libs	Este directorio contendrá todas las librerías externas a **FlavorPHP**
models	Este directorio contendrá todos nuestros modelos.
views	Este directorio contendrá todas nuestras vistas.
.htaccess	Este archivo se encarga del hacer el ruteo de nuestra aplicación.
config.php	Este archivo contendrá la configuración de nuestra aplicación.
index.php	Este archivo es el punto de entrada a nuestra aplicación (no debe modificarse).

```
//servidor de BD.
define('DB_Server', 'localhost');
//nombre de la BD.
define('DB_name', 'nombreBD');
//usuario de la BD.
define('DB_User', 'usuarioBD');
//contraseña del usuario de la BD.
define('DB_Password', 'contraseñaBD');
define('DB_Port', false); //puerto
//ruta base del proyecto.
define('Path', "http://localhost/flavor");
```

En la tabla 1.2 se describen los valores del archivo de configuración.

Cuadro 1.2: Archivo de configuración

Valor	**Descripción**
requiresBD	Define si requiere de una base de datos (true, false).
DB_Engine	Tipo de motor de base de datos. Dependiendo de su valor es la conexión que se generará [1].
DB_Server	Nombre del equipo o IP donde se encuentra el servidor de base de datos. (generalmente es *localhost* ó *127.0.0.1*).
DB_name	Nombre de la base de datos que contrendrá el proyecto.
DB_User	Usuario para conectarse a la base de datos.
DB_Password	Contraseña de usuario para conectarse a la base de datos.
DB_Port	Número de puerto para la conexión (si es necesario).
Path	Ruta base del proyecto.

Una vez que hemos modificado nuestro *./config.php* con la información de configuración de nuestro proyecto, estamos creca de poder comenzar a desarrollar con **FlavorPHP**, pero antes de eso veamos como hacer uso de librerías de clases externas a nuestro framework.

1.2.4. Librerías de terceros en FlavorPHP

Si deseamos incluir nuestras propias (o de terceros) clases en nuestro proyectos y que FlavorPHP las detecte de forma automática [2], podemos hacerlo dentro de la carpeta *./libs*. Para esto basta con seguir la siguiente convención:

- Crear (o descargar) nuestra clase.
- Llamar (o renombrar) el archivo de nuestra clase como ***nombre_clase*.class.php**.
- Mover el archivo de nuestra clase a la carpeta *./libs*.

Por ejemplo, tenemos una clase que al crearse regresa una cadena con el valor de Ḧola(ver listado 1.2)

Código 1.2: Saludo de Hola

```
class hola {
   public function __construct() {
      return "Hola";
   }
}
```

Para poder usar esta clase en **FlavorPHP** debemos llamarla *hola.class.php* y guardarla dentro de la carpeta *./libs*, es decir quedaría *./libs/hola.class.php*. Si seguimos esta convención ya no será necesario utilizar *include* o *require* dentro de nuestro proyecto.

Ahora sí, estamos listos para comenzar a programar utilizando **FlavorPHP**.

[2] solo detecta clases, no archivos de funciones.

Capítulo 2

MVC
(Modelo-Vista-Controlador)

2.1. Implementando con MVC

Antes de continuar, entendamos un poco más el patrón de diseño MVC (Modelo - Vista - Controlador). Como se mencionó en el capítulo anterior, el modelo MVC permite hacer la separación de las capas de interfaz, modelo y lógica de control de esta. La programación por capas es un estilo de programación en la que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño, un ejemplo básico de esto es separar la capa de datos de la capa de presentación al usuario. La ventaja principal de este estilo, es que el desarrollo se puede llevar a cabo en varios niveles y en caso de algún cambio sólo se ataca al nivel requerido sin tener que revisar entre código mezclado. La división en capas reduce la complejidad, facilita la reutilización y acelera el proceso de ensamblar o desensamblar alguna capa, o sustituirla por otra distinta.

2.2. Conceptos del MVC

El nucleo de **FlavorPHP** es el MVC, un patrón de diseño muy popular que funciona en tres capas:

- **Modelos**: Representan los datos que utiliza nuestra aplicación, es decir su lógica de negocios.
- **Vistas**: Visualizan el modelo e interactuando con los usuarios de estas.
- **Controladores**: Obtiene las acciones de los usuarios e invocan cambios en las vistas o en los modelos según sea necesario.

Cuando un usuario realiza una petición a nuestra aplicación Web, esta petición es hecha utilizando el protocolo HTTP, **FlavorPHP** redirecciona esta petición enviandola directamente a su respectivo controlador. Los controladores pueden interactuar con los modelos y llamar a las vistas (interfaces de usuario), estas últimas se muestran al usuario respondiendo a la petición inicial.

2.3. URLs en FlavorPHP (routing)

FlavorPHP utiliza una convención especial en sus URLs para que funcionen nuestras aplicaciones y permitir el acceso a los controladores y sus acciones.

Por ejemplo:

```
http://localhost/saludo/hola/pedro/
```

De la URL anterior obtenemos:

- Nuestra aplicación esta instalada en: **http://localhost/**.
- **FlavorPHP** invocará al `controlador` llamado **saludo**.

- **FlavorPHP** ejecutará el `método` **hola** del `controlador` **saludo**.
- **FlavorPHP** pasará al `método` **hola** del `controlador` **saludo** el parámetro **pedro**.

Como podemos notar, los URLs en **FlavorPHP** se componen de 4 elementos principales:

1. URL base de la aplicación.
2. Nombre del modelo.
3. Nombre del método a ejecutar en el modelo.
4. Parámetro (argumento) que se la pasa al método a ejecutar en el modelo.

En base a estos 4 componentes principales, podemos realizar algunas combinaciones para hacer más fácil el uso de **FlavorPHP**.

2.3.1. Llamar al controlador index

Vemos que obtenemos al utilizar la siguiente URL:

```
http://localhost/
```

Cuando se usa solo la URL base de la aplicación, **FlavorPHP** ejecuta el método **index** del controlador **index** sin ningún parámetro.

Controlador index con parámetro

Si deseamos pasar un parámetro numérico al método **index** del controlador **index**, podemos utilizar la siguiente URL:

```
http://localhost/10/
```

De esta forma el método **index** trendrá como parámetro el número **10**. Esta función trabaja **sí y solo sí** es númerico el primer (y único) elemento después de la URL base de la aplicación.

2.3.2. Invocar a los métodos index

Todos los controladores por defecto deben tener su método index, por lo que si deseamos llamar estos métodos solo hay que poner el nombre del controlador y **FlavorPHP** se encargará de ejecutar el método index.
Por ejemplo:

```
http://localhost/usuario/
```

Esta URL ejecutará el método **index** del controlador **usuario**.

2.4. Hola mundo con FlavorPHP

Para crear la aplicación que nos salude desde **FlavorPHP** vamos a seguir las convenciones aprendidas en la sección anterior.

La URL final será:

`http://localhost/saludo/hola/pedro/`

Primero debemos crear nuestro controlador `saludo` y almacenarlo en la carpeta *controllers* con la siguiente regla `nombreControlador_controller.php`.

Por lo que nuestro controlador se llamará **saludo_controller.php**, no nos olvidemos de esta convención al nombrar nuestros controladores para que el framework pueda manejarlos correctamente.

En el código 2.1 podemos ver la estructura base que deben tener todos nuestros controladores.

Código 2.1: saludo_controller.php

```
class saludo_controller extends appcontroller {
      public function __construct() {
         parent::__construct();

      }
     public function index($id=NULL) { }
}
```

Si en estos momentos llamamos a la URL `http://localhost/saludo/` veremos en blanco nuestro navegador, debido a que aun no generamos ninguna vista.

Agreguémosle lo siguiente (código 2.2) a nuestro controlador:

Código 2.2: Controlador saludo incorrecto

```
class saludo_controller extends appcontroller {
      public function __construct() {
         parent::__construct();

      }
      public function index($id=null){
         echo '<h1>Hola Mundo</h1>';
      }
   }
```

Si refrescamos (F5) nuestro navegador veremos el texto '`Hola Mundo`'. Pero, **¡esto no es correcto!** debemos crear una vista desde la cual se muestre la información al usuario.

Para generar la vista, debemos crear una carpeta nueva dentro de la carpeta `views`, esta nueva carpeta debe llamarse exactamente igual que el controlador, en este caso quedará *./views/**saludo**/*, dentro de esta carpeta debemos crear un

archivo que se llame igual que el método llamado, para este ejemplo el archivo debe ser *./views/saludo/**index.php*** (código 2.3).

Código 2.3: vista index del controlador saludo

```
<h1>Hola mundo</h1>
```

Una vez creada la vista modificamos el controlador para mostrarla.

Código 2.4: Controlador saludo correcto

```
class saludo_controller extends appcontroller {
      public function __construct() {
         parent::__construct();

      }
      public function index($id=null){
         $this->render();
      }
   }
```

Al usar la instrucción `render()` automáticamente **FlavorPHP** sabrá cual vista buscar y la mostrará al usuario.

Ahora que ya entendemos mejor como utilizar el framework, volvamos al objetivo inicial.

A nuestro controlador saludo le agregaremos el método **hola** que recibirá un parámetro de nombre **msg** (código 2.5). Este parámetro será asignado a la vista para que lo despliegue junto a un saludo.

Código 2.5: Método hola

```
class saludo_controller extends appcontroller {
      public function __construct() {
         parent::__construct();
      }
      public function index($id=null){
         $this->render();
      }
      public function hola($msg){
         $this->view->nombre = $msg;
         $this->render();
      }
   }
```

La vista (*./views/saludo/**hola.php***) quedará como muestra el código 2.6

Código 2.6: Vista del método hola

```
<h1>Hola <?php echo $nombre; ?></h1>
```

Como podemos observar, al asignarle en nuestros controladores una variable a nuestro objeto **$this->view**, la podemos utilizar desde la vista, haciendo uso de PHP.

Capítulo 3

Mi primera aplicación con Flavor

3.1. Introducción

Para ilustrar la facilidad con que **FlavorPHP** nos permite crear aplicaciones web con pocas lineas de código, crearemos una aplicación que nos permita almacenar nuestros enlaces favoritos.

Lo primero que debemos hacer es descomprimir **FlavorPHP** en nuestro servidor y llamarle a nuestra aplicacion `demo` (el nombre de la carpeta que contiene al framework).

El siguiente paso es realizar la configuración en el archivo `config.php`.

Una vez que hayamos configurado correctamente, procedemos a crear la tabla en la base de datos.

3.2. El modelo de datos

Necesitaremos una tabla en nuestra base de datos (BD), la cual almacenará nuestros enlaces. Esta tabla la llamaremos **bookmarks**.

Los campos a utilizar se describen en la tabla 3.1.

Cuadro 3.1: Campos en la tabla bookmarks

Campo	**Utilidad**
id_link	Valor numérico que se llena automáticamente de forma incremental (debe ser llave primaria).
title	Almacena el título del sitio Web.
url	Almacena el enlace al sitio Web.
description	Descripción del sitio Web.
tags	Etiquetas para organizar los sitios Web.
created	Este Campo lo llena automáticamente **FlavorPHP** al agregar un nuevo sitio Web.
modified.php	Este Campo lo llena automáticamente **FlavorPHP** cuando se modifica un enlace.

Como podemos ver, existen tres campos por convención que necesita y usa **FlavorPHP**:

1. Campo llave primaria (id_link), en este campo no importa el nombre.

2. Campo **created**.

3. Campo **modified**.

3.2.1. Crear la BD

El script para generar la base de datos lo encontramos en el código 3.1.

Código 3.1: Tabla bookmarks

```
CREATE TABLE `bookmarks` (
  `id_link` int(3) NOT NULL auto_increment,
  `title` varchar(100) NOT NULL default '',
  `url` varchar(100) NOT NULL default '',
  `description` text ,
  `tags` varchar(75) NOT NULL default '',
  `created` datetime NOT NULL,
  `modified` datetime NOT NULL,
  PRIMARY KEY (`id_link`)
);
```

Nota: *Para poder apreciar el ejemplo inserte en esta parte un par de registros a su tabla de forma manual.*

3.2.2. Crear el modelo

Los modelos forman parte del MVC, representan la lógica de datos y encapsulan las acciones que se podrán hacer sobre las tablas en las bases de datos, tales como insertar, modificar, borrar, consultar, etc.

Los modelos en **FlavorPHP**, implementan el patrón ORM (Mapeo Objeto Relacional), el cual nos permite trabajar con nuestras tablas como si fueran clases y nuestros registros como si fueran objetos. Estos modelos mapean directamente nuestas tablas en las bases de datos, debido a esto, debemos tomar en cuenta **otra convención**, el modelo **se debe de llamar** como la tabla que vamos a mapear **pero en singular**. Es decir, si nuestra tabla se llama `bookmarks` nuestro modelo se llamará `bookmark`.

Este modelo (ver código 3.2) lo vamos a crear en el directorio `models` de la siguiente forma: */models/**bookmark.php***.

Código 3.2: Modelo bookmarks

```
class bookmark extends models {

}
```

Con este modelo será suficiente para nuestra aplicación. **FlavorPHP** automáticamente ya ha creado un enlace entre nuestro modelo y la tabla en la BD por lo que ya podemos trabajar con ella como si de un objeto se tratara.

3.3. Crear los controladores y las vistas

Vamos a crear los controladores y vistas necesarias para nuestra aplicación.

3.3.1. Usando un modelo desde un controlador

Lo primero que debemos hacer es crear el controlador `index` (ver código 3.3), es decir */controllers/**index_controller.php***.

Código 3.3: Bookmarks - index_controller.php

```
class index_controller extends appcontroller {
      public function __construct() {
         parent::__construct();

      }
     public function index($id=NULL) {

     }
}
```

Ya que tenemos nuestro controlador, le agregamos el código siguiente:

Código 3.4: Seleccionar todos

```
class index_controller extends appcontroller {
      public function __construct() {
         parent::__construct();

      }
      public function index($id=null){
         $link = new bookmark();
         $this->view->links = $link->findAll();
         $this->render();
      }
}
```

En la línea 3 se llama al modelo que creamos previamente, después le asignamos a la vista una variable que contendrá todos los enlaces en nuestra tabla y por último mostramos la vista.

3.3.2. Crear la vista

Creamos nuestra vista (ver código 3.5) en */views/index/**index.php***

Código 3.5: La vista del index

```
<?php foreach ($links as $link) { ?>
   <a href="<?php echo $link["url"]; ?>"><?php echo $link
     ["title"]; ?></a>
```

```
    <p><? echo $link["description"]; ?></p>
    <p><? echo $link["tags"]; ?> el
    <? echo date("m.d.y", strtotime($link["created"]));
      ?></p>
<?php } ?>
```

Ahora que conocemos como conectarnos a la BD, obtener todos los registros y desplegarlos en nuestro navegador con muy pocas lineas de código, aprendamos a realizar las operaciones de insertar, modificar y eliminar enlaces en nuestra aplicación.

A nuestra vista le agregamos las siguientes líneas (ver código 3.6), para generar las URLs de editar, eliminar y agregar enlaces.

Código 3.6: La vista con operaciones de CRUD

```
<p>
<?php echo $this->html->linkTo("Agregar enlace", "index/
  agregar/", " title=\"Agregar un nuevo enlace\""); ?>
</p>
<hr />
<?php foreach ($links as $link) { ?>
    <a href="<?php echo $link["url"]; ?>"><?php echo $link
      ["title"]; ?></a>

    <p><?php echo $link["description"]; ?></p>
    <p><?php echo $link["tags"]; ?> el
    <?php echo date("m.d.y", strtotime($link["created"]));
      ?></p>
     <p>Operaciones con el enlace: <?php echo $this->html
       ->linkTo("Editar", "index/editar/".$link["id_link"
       ]."/", " title=\"Editar enlace\""); ?>  |
     <?php echo $this->html->linkToConfirm("Eliminar", "
       index/eliminar/".$link["id_link"]."/"); ?>
     </p>
<?php } ?>
```

Como podemos observar, en nuestra vista estamos usando un objeto `html`, este es el helper HTML que incluye FlavorPHP y nos permite crear elementos HTML de forma más sencilla y respetando siempre la estructura del framework, los dos métodos del helper que en estos momentos hemos utilizado son los siguientes:

1. `linkTo(text, url, html_attributes)` .- Crea un enlace. Los parametros son:

   text Texto que llevará el enlace.

   url URL a donde se dirijirá el usuario.

html_attributes Atributos extras del enlace.

2. `linkToConfirm(text, url)` .- Crea un enlace, el cual para activarse, el usuario primero debe aceptarlo. Los parametros son:

   text Texto que llevará el enlace.

   url URL a donde se dirijirá el usuario.

3.3.3. Agregar enlace

En el código 3.8 veremos como queda el método agregar.

Código 3.7: Agregar enlace

```
public function agregar($id=null){
        $link = new bookmark();
        if($this->data){
            $link->prepareFromArray($this->data);
            $link->save();
            $this->redirect("index");
        } else {
            $this->title_for_layout("Agregar Enlace");
            $this->render();
        }
}
```

Y la vista para agregar */views/index/**agregar.php*** (ver código **??**)

Código 3.8: Agregar enlace

```
<?php echo $this->html->form("index/agregar/"); ?>
    URL: <?php echo $this->html->textField("url"); ?>
    <br />
    Titulo: <?php echo $this->html->textField("title"); ?>
    <br />
    Descripcion: <?php echo $this->html->textArea("
      description", "", " rows=\"3\" cols=\"84\" "); ?>
    <br />
    Etiquetas: <?php echo $this->html->textField("tags");
      ?>
    <br />
    <input type="submit" value="Agregar" /><br />
</form>
```

3.3.4. Editar enlace

En el código 3.9 veremos como queda el método editar.

Código 3.9: Editar enlace

```
public function editar($id){
        $link = new bookmark();
        $this->view->link = $link->find($id);
        $this->view->id = $id;
        if($this->data){
            $link->prepareFromArray($this->data);
            $link->save();
            $this->redirect("index");
        } else {
            $this->title_for_layout("Editar Enlace");
            $this->render();
        }
}
```

Y la vista para editar */views/index/**editar.php*** (ver código 3.10)

Código 3.10: Vista editar enlace

```
<?php echo $this->html->form("index/editar/".$id."/"); ?>
    URL: <?php echo $this->html->textField("url", " value
      =\"".$link["url"]."\" "); ?>
    <br />
    Titulo: <?php echo $this->html->textField("title", "
      value=\"".$link["title"]."\" "); ?>
    <br />
    Descripcion: <?php echo $this->html->textArea("
      description", $link["description"], " rows=\"3\"
      cols=\"84\" "); ?>
    <br />
    Etiquetas: <?php echo $this->html->textField("tags", "
       value=\"".$link["tags"]."\" "); ?>
    <br />
    <input type="submit" value="Modificar" /><br />
</form>
```

3.3.5. Eliminar enlace

En el código 3.11 veremos como queda el método eliminar.

Código 3.11: Eliminar enlace

```
public function eliminar($id){
        $link = new bookmark();
        $link->find($id);
        $link->delete();
        $this->redirect("index");
```

```
}
```

Recordemos que todos estos métodos anteriores deben de ir en el index_controller que creamos al inicio */controllers/**index_controller.php***

Este ejemplo nos permite entrar de lleno al uso de **FlavorPHP**, por lo que el siguiente paso es conocer el API completa del framework para obtener el máximo provecho de su funcionalidad.

FlavorPHP
Versión 1.0

Manual

http://www.flavorphp.com/