



CakePHP

CakePHP Cookbook Documentation

Versión 2.x

Cake Software Foundation

06 de diciembre de 2024

Índice general

| | |
|---|-----------|
| 1. Primeros Pasos | 1 |
| 2. Parte 1: Tutorial para desarrollar el Blog | 3 |
| Descargar CakePHP | 4 |
| Creando la base de datos para nuestro blog | 4 |
| Configurando la Base de Datos | 5 |
| Configuración Opcional | 5 |
| Sobre mod_rewrite | 6 |
| 3. Parte 2: Tutorial para desarrollar el Blog | 9 |
| Creando un modelo para los artículos (<i>Post Model</i>) | 9 |
| Crear un Controlador para nuestros Artículos (<i>Posts</i>) | 10 |
| Creando una vista para los artículos (<i>View</i>) | 10 |
| Añadiendo artículos (<i>posts</i>) | 13 |
| Validando los datos | 14 |
| Editando Posts | 15 |
| Borrando Artículos | 17 |
| Rutas (<i>Routes</i>) | 18 |
| Conclusión | 19 |
| Lectura sugerida para continuar desde aquí | 19 |
| Lectura Adicional | 19 |
| 4. Instalación | 27 |
| Descargar CakePHP | 27 |
| Permisos | 28 |
| Configuración | 28 |
| Desarrollo | 28 |
| Producción | 29 |
| Instalación avanzada y configuración flexible | 30 |
| ¡ A por todas ! | 36 |
| 5. Introducción a CakePHP | 37 |
| ¿Qué es CakePHP? y ¿Por qué usarlo? | 37 |
| Entendiendo el Modelo - Vista - Controlador | 38 |

| | |
|---|-----------|
| Dónde encontrar ayuda | 40 |
| 6. Controllers | 43 |
| Controller Methods | 43 |
| 7. Views | 49 |
| Plantillas de vistas | 49 |
| Usar bloques de vistas | 51 |
| Layouts | 53 |
| Elementos | 55 |
| Crear tus propias clases de vistas | 58 |
| API de Vistas | 59 |
| Más acerca de vistas | 61 |
| 8. Models | 63 |
| Associations: Linking Models Together | 63 |
| Retrieving Your Data | 64 |
| Saving Your Data | 64 |
| Deleting Data | 64 |
| Data Validation | 64 |
| Callback Methods | 65 |
| Behaviors | 65 |
| DataSources | 65 |
| Model Attributes | 66 |
| Additional Methods and Properties | 66 |
| Virtual fields | 66 |
| Transactions | 66 |
| 9. Core Libraries | 67 |
| General Purpose | 67 |
| Behaviors | 69 |
| Components | 70 |
| Helpers | 73 |
| Utilities | 78 |
| 10. Plugins | 83 |
| How To Install Plugins | 83 |
| How To Use Plugins | 84 |
| How To Create Plugins | 84 |
| 11. Shells, Tasks & Console Tools | 85 |
| Shell Helpers | 85 |
| Running Shells as cronjobs | 86 |
| Completion Shell | 86 |
| Code Generation with Bake | 86 |
| Schema management and migrations | 86 |
| I18N shell | 87 |
| ACL Shell | 87 |
| Test shell | 87 |
| Upgrade shell | 87 |
| 12. Development | 89 |
| Configuration | 89 |
| Routing | 90 |
| Sessions | 90 |

| | |
|---|------------|
| Exceptions | 90 |
| Error Handling | 90 |
| Debugging | 91 |
| Testing | 91 |
| REST | 91 |
| Dispatcher Filters | 91 |
| 13. ES - Deployment | 93 |
| 14. Tutoriales y Ejemplos | 95 |
| Parte 1: Tutorial para desarrollar el Blog | 95 |
| Parte 2: Tutorial para desarrollar el Blog | 99 |
| Aplicación con autenticación y autorizacion | 109 |
| Simple Acl controlled Application | 116 |
| Simple Acl controlled Application - part 2 | 117 |
| 15. Contributing | 119 |
| Documentation | 119 |
| Tickets | 120 |
| Code | 120 |
| Coding Standards | 120 |
| Backwards Compatibility Guide | 120 |
| CakePHP Development Process | 121 |
| 16. Appendices | 123 |
| 2.10 Migration Guide | 123 |
| 2.9 Migration Guide | 124 |
| 2.8 Migration Guide | 124 |
| 2.7 Migration Guide | 124 |
| 2.6 Migration Guide | 125 |
| 2.5 Migration Guide | 125 |
| 2.4 Migration Guide | 125 |
| 2.3 Migration Guide | 126 |
| 2.2 Migration Guide | 126 |
| 2.1 Migration Guide | 126 |
| 2.0 Migration Guide | 127 |
| Migration from 1.2 to 1.3 | 128 |
| General Information | 128 |
| 17. Indices and tables | 131 |
| Índice | 133 |

Primeros Pasos

CakePHP te proporciona una base robusta para construir tus aplicaciones. Se encarga de todo, desde la petición inicial del usuario hasta la construcción del código HTML final. Como CakePHP sigue los fundamentos del patrón MVC, te permite personalizar y extender fácilmente cada una de sus partes.

Este framework también proporciona una estructura organizacional: desde los nombres de ficheros hasta los nombres de las tablas en la base de datos. Esto mantiene tu aplicación consistente y ordenada. Siendo un concepto sencillo, seguir las convenciones definidas te facilitará encontrar rápidamente cada cosa y aprenderás en menos tiempo los fundamentos del framework.

La mejor manera de empezar es ponerte manos a la obra y desarrollar algo. Para empezar construiremos un Blog sencillo.

Parte 1: Tutorial para desarrollar el Blog

Bienvenido a CakePHP. Probablemente estás consultando este tutorial porque quieres aprender cómo funciona CakePHP. Nuestro objetivo es potenciar tu productividad y hacer más divertido el desarrollo de aplicaciones. Esperamos que puedas comprobarlo a medida que vas profundizando en el código.

En este tutorial vamos a crear un blog sencillo desde cero. Empezaremos descargando e instalando CakePHP, luego crearemos una base de datos y el código necesario para listar, añadir, editar o borrar artículos del blog.

Esto es lo que necesitas:

1. Servidor web funcionando. Asumiremos que estás usando Apache, aunque las instrucciones para otros servidores son similares. Igual tendremos que ajustar un poco la configuración inicial, pero todos los pasos son sencillos. La mayor parte de nosotros podrá tener CakePHP funcionando sin tocar nada en su configuración.
2. Base de datos funcionando. Usaremos MySQL en este tutorial. Necesitarás saber cómo crear una base de datos nueva. CakePHP se encargará del resto.
3. Nivel básico de PHP. Si estás familiarizado con la programación orientada a objetos, mucho mejor. Aún así puedes seguir desarrollando con tu estilo procedimental si lo prefieres.
4. Conocimiento sobre patrón MVC. Puedes encontrar una definición rápida aquí: [Entendiendo el Modelo - Vista - Controlador](#). No tengas miedo, sólo es media página.

¡ Vamos allá !

Descargar CakePHP

Vamos a descargar la última versión de CakePHP.

Para ello, visita la web del proyecto en GitHub: <https://github.com/cakephp/cakephp/tags> y descargar / descomprimir la última versión de la rama 2.0

También puedes clonar el repositorio usando `git`¹. `git clone git://github.com/cakephp/cakephp.git`

Usa el método que prefieras y coloca la carpeta que has descargado bajo la ruta de tu servidor web (dentro de tu DocumentRoot). Una vez terminado, tu directorio debería tener esta estructura:

```
/path_to_document_root
  /app
  /lib
  /plugins
  /vendors
  .htaccess
  index.php
  README
```

Es buen momento para aprender algo sobre cómo funciona esta estructura de directorios: echa un vistazo a «Directorios en CakePHP», Sección: *Estructura de directorios de CakePHP*.

Creando la base de datos para nuestro blog

Vamos a crear una nueva base de datos para el blog. Puedes crear una base de datos en blanco con el nombre que quieras. De momento vamos a definir sólo una tabla para nuestros artículos («posts»). Además crearemos algunos artículos de test para usarlos luego. Una vez creada la tabla, ejecuta el siguiente código SQL en ella:

```
/* tabla para nuestros articulos */
CREATE TABLE posts (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(50),
  body TEXT,
  created DATETIME DEFAULT NULL,
  modified DATETIME DEFAULT NULL
);

/* algunos valores de test */
INSERT INTO posts (title,body,created)
  VALUES ('The title', 'This is the post body.', NOW());
INSERT INTO posts (title,body,created)
  VALUES ('A title once again', 'And the post body follows.', NOW());
INSERT INTO posts (title,body,created)
  VALUES ('Title strikes back', 'This is really exciting! Not.', NOW());
```

La elección de los nombres para el nombre de la tabla y de algunas columnas no se ha hecho al azar. Si sigues las convenciones para nombres en la Base de Datos, y las demás convenciones en tus clases (ver más sobre convenciones aquí: *Convenciones en CakePHP*), aprovecharás la potencia del framework y ahorrarás mucho trabajo de configuración.

¹ <https://git-scm.com/>

CakePHP es flexible, si no quieres usar las convenciones puedes configurar luego cada elemento para que funcione con tu Base de Datos legada. Te recomendamos que utilices estas convenciones ya que te ahorrarán tiempo.

Al llamar “posts” a nuestra tabla de artículos, estamos diciendo a CakePHP que vincule esta tabla por defecto al Modelo “Post”, e incluir los campos “modified” y “created” con ese nombre, serán automáticamente administrados por CakePHP.

Configurando la Base de Datos

Rápido y sencillo, vamos a decirle a CakePHP dónde está la Base de Datos y cómo conectarnos a ella. Probablemente ésta será la primera y última vez que lo hagas en cada proyecto.

Hay un fichero de configuración preparado para que sólo tengas que copiarlo y modificarlo con tu propia configuración.

Cambia el nombre del fichero `/app/Config/database.php.default` por `/app/Config/database.php` (hemos eliminado el “.default” del final).

Edita ahora este fichero y verás un array definido en la variable `$default` que contiene varios campos. Modifica esos campos para que se correspondan con tu configuración actual de acceso a la Base de Datos. Debería quedarte algo similar a esto:

```
public $default = array(
    'datasource' => 'Database/Mysql',
    'persistent' => false,
    'host' => 'localhost',
    'port' => '',
    'login' => 'cakeBlog',
    'password' => 'c4k3-rU13Z',
    'database' => 'cake_blog_tutorial',
    'schema' => '',
    'prefix' => '',
    'encoding' => ''
);
```

Ten en cuenta que los campos “login”, “password”, “database” tendrás que cambiarlos por tu usuario de MySQL, tu contraseña de MySQL y el nombre que le diste a la Base de Datos.

Guarda este fichero.

Ahora ya podrás acceder a la página inicial de bienvenida de CakePHP en tu máquina. Esta página podrás accederla normalmente en <http://localhost/cakeblog> si has llamado a la carpeta raíz del proyecto “cakeblog”. Verás una página de bienvenida que muestra varias informaciones de configuración y te indica si tienes correctamente instalado CakePHP.

Configuración Opcional

Hay otras tres cosas que puedes querer configurar, aunque no son requeridas para este tutorial no está mal echarles un vistazo. Para ello abre el fichero `/app/Config/core.php` que contiene todos estos parámetros.

1. Configurar un string de seguridad “salt” para usarlo al realizar los “hash”.
2. Configurar un número semilla para el encriptado “seed”.
3. Definir permisos de escritura en la carpeta `Tmp`. El servidor web (normalmente “apache”) debe poder escribir dentro de esta carpeta y subcarpetas.

El string de seguridad se utiliza en la generación de “hashes”. Cambia el valor inicial y escribe cualquier cosa diferente. Cualquier cosa vale. Para cambiarlo vete a la línea 203 del fichero `/app/Config/core.php` y verás algo así:

```
/**
 * A random string used in security hashing methods.
 */
Configure::write('Security.salt', 'p1345e-P45s_7h3*S@l7!');
```

El número semilla se utiliza para encriptar y desencriptar cadenas. Cambia el valor por defecto en el fichero `/app/Config/core.php` línea 208. No importa qué número pongas, que sea difícil de adivinar.

```
/**
 * A random numeric string (digits only) used to encrypt/decrypt strings.
 */
Configure::write('Security.cipherSeed', '7485712659625147843639846751');
```

Para dar permisos al directorio `app/Tmp`, la mejor forma es ver qué usuario está ejecutando el servidor web (`<?php echo `whoami` ; ?>`) y cambiar el directorio para que el nuevo propietario sea el usuario que ejecuta el servidor web.

En un sistema *nix esto se hace así:

```
$ chown -R www-data app/tmp
```

Suponiendo que `www-data` sea el usuario que ejecuta tu servidor web (en otras versiones de *unix como fedora, el usuario suele llamarse “apache”).

Si CakePHP no puede escribir en este directorio, te informará de ello en la página de bienvenida, siempre que tengas activado el modo depuración, por defecto está activo.

Sobre mod_rewrite

Si eres nuevo usuario de Apache, puedes encontrar alguna dificultad con `mod_rewrite`, así que lo trataremos aquí.

Si al cargar la página de bienvenida de CakePHP ves cosas raras (no se cargan las imágenes ni los estilos y se ve todo en blanco y negro), esto significa que probablemente la configuración necesita ser revisada en el servidor Apache. Prueba lo siguiente:

1. Asegúrate de que existe la configuración para procesar los ficheros `.htaccess`. En el fichero de configuración de Apache: “`httpd.conf`” debería existir una sección para cada “Directory” de tu servidor. Asegúrate de que `AllowOverride` está fijado a `All` para el directorio que contiene tu aplicación web. Para tu seguridad, es mejor que no asignes `All` a tu directorio raíz `<Directory />` sino que busques el bloque `<Directory>` que se refiera al directorio en el que tienes instalada tu aplicación web.
2. Asegúrate que estás editando el fichero `httpd.conf` correcto, ya que en algunos sistemas hay ficheros de este tipo por usuario o por aplicación web. Consulta la documentación de Apache para tu sistema.
3. Comprueba que existen los ficheros `.htaccess` en el directorio en el que está instalada tu aplicación web. A veces al descomprimir el archivo o al copiarlo desde otra ubicación, estos ficheros no se copian correctamente. Si no están ahí, obtén otra copia de CakePHP desde el servidor oficial de descargas.
4. Asegúrate de tener activado el módulo `mod_rewrite` en la configuración de Apache. Deberías tener algo así:

```
LoadModule rewrite_module      libexec/httpd/mod_rewrite.so

(para Apache 1.3)::
```

(continué en la próxima página)

(proviene de la página anterior)

```
AddModule      mod_rewrite.c  
  
en tu fichero httpd.conf
```

Si no puedes (o no quieres) configurar `mod_rewrite` o algún otro módulo compatible, necesitarás activar las url amigables en CakePHP. En el fichero `/app/Config/core.php`, quita el comentario a la línea:

```
Configure::write('App.baseUrl', env('SCRIPT_NAME'));
```

Borra también los ficheros `.htaccess` que ya no serán necesarios:

```
/.htaccess  
/app/.htaccess  
/app/webroot/.htaccess
```

Esto hará que tus url sean así: `www.example.com/index.php/nombredelcontrolador/nombredelaaccion/parametro` en vez de `www.example.com/nombredelcontrolador/nombredelaaccion/parametro`.

Si estás instalando CakePHP en otro servidor diferente a Apache, encontrarás instrucciones para que funcione la reescritura de URLs en la sección *Instalación Avanzada*

Parte 2: Tutorial para desarrollar el Blog

Creando un modelo para los artículos (*Post Model*)

Los modelos son una parte fundamental en CakePHP. Cuando creamos un modelo, podemos interactuar con la base de datos para crear, editar, ver y borrar con facilidad cada ítem de ese modelo.

Los ficheros en los que se definen los modelos se ubican en la carpeta `/app/Model`, y el fichero que vamos a crear debe guardarse en la ruta `/app/Model/Post.php`. El contenido de este fichero será:

```
class Post extends AppModel {
    public $name = 'Post';
}
```

Los convenios usados para los nombres son importantes. Cuando llamamos a nuestro modelo *Post*, CakePHP deducirá automáticamente que este modelo se utilizará en el controlador `PostsController`, y que se vinculará a una tabla en nuestra base de datos llamada `posts`.

Nota: CakePHP creará dinámicamente un objeto para el modelo si no encuentra el fichero correspondiente en `/app/Model`. Esto significa que si te equivocas al nombrar el fichero (por ejemplo lo llamas `post.php` con la primera p minúscula o `posts.php` en plural) CakePHP no va a reconocer la configuración que escribas en ese fichero y utilizará valores por defecto.

Para más información sobre modelos, como prefijos para las tablas, validación, etc. puedes visitar [Models](#) en el Manual.

Crear un Controlador para nuestros Artículos (*Posts*)

Vamos a crear ahora un controlador para nuestros artículos. En el controlador es donde escribiremos el código para interactuar con nuestros artículos. Es donde se utilizan los modelos para llevar a cabo el trabajo que queramos hacer con nuestros artículos. Vamos a crear un nuevo fichero llamado `PostsController.php` dentro de la ruta `/app/Controller`. El contenido de este fichero será:

```
class PostsController extends AppController {
    public $helpers = array('Html', 'Form');
}
```

Y vamos a añadir una acción a nuestro nuevo controlador. Las acciones representan una función concreta o interfaz en nuestra aplicación. Por ejemplo, cuando los usuarios recuperan la url `www.example.com/posts/index` (que CakePHP también asigna por defecto a la ruta `www.example.com/posts/` ya que la acción por defecto de cada controlador es `index` por convención) esperan ver un listado de *posts*. El código para tal acción sería este:

```
class PostsController extends AppController {
    public $helpers = array ('Html', 'Form');

    function index() {
        $this->set('posts', $this->Post->find('all'));
    }
}
```

Si examinamos el contenido de la función `index()` en detalle, podemos ver que ahora los usuarios podrán acceder a la ruta `www.example.com/posts/index`. Además si creáramos otra función llamada `foobar()`, los usuarios podrían acceder a ella en la url `www.example.com/posts/foobar`.

Advertencia: Puede que tengas la tentación de llamar tus controladores y acciones de forma determinada para que esto afecte a la ruta final, y así puedas predeterminedir estas rutas. No te preocupes por esto ya que CakePHP incorpora un potente sistema de configuración de rutas. Al escribir los ficheros, te recomendamos seguir las convenciones de nombres y ser claro. Luego podrás generar las rutas que te convengan utilizando el componente de rutas (*Route*).

La función `index` tiene sólo una instrucción `set()` que sirve para pasar información desde el controlador a la vista (*view*) asociada. Luego crearemos esta vista. Esta función `set()` asigna una nueva variable “posts” igual al valor retornado por la función `find('all')` del modelo `Post`. Nuestro modelo `Post` está disponible automáticamente en el controlador y no hay que importarlo ya que hemos usado las convenciones de nombres de CakePHP.

Para aprender más sobre los controladores, puedes visitar el capítulo [Controllers](#)

Creando una vista para los artículos (*View*)

Ya tenemos un modelo que define nuestros artículos y un controlador que ejecuta alguna lógica sobre ese modelo y envía los datos recuperados a la vista. Ahora vamos a crear una vista para la acción `index()`.

Las vistas en CakePHP están orientadas a cómo se van a presentar los datos. Las vistas encajan dentro de *layouts* o plantillas. Normalmente las vistas son una mezcla de HTML y PHP, aunque pueden ser también XML, CSV o incluso datos binarios.

Las plantillas (*layouts*) sirven para recubrir las vistas y reutilizar código. Además pueden crearse tantos layouts como se deseen y se puede elegir cuál utilizar en cada momento. Por el momento vamos a usar el la plantilla por defecto `default`.

¿ Recuerdas que el controlador envió a la vista una variable `posts` que contiene todos los posts mediante el método `set()` ? Esto nos generará una variable en la vista con esta pinta:

```
// print_r($posts) output:
Array
(
    [0] => Array
        (
            [Post] => Array
                (
                    [id] => 1
                    [title] => The title
                    [body] => This is the post body.
                    [created] => 2008-02-13 18:34:55
                    [modified] =>
                )
            )
        [1] => Array
            (
                [Post] => Array
                    (
                        [id] => 2
                        [title] => A title once again
                        [body] => And the post body follows.
                        [created] => 2008-02-13 18:34:56
                        [modified] =>
                    )
            )
        [2] => Array
            (
                [Post] => Array
                    (
                        [id] => 3
                        [title] => Title strikes back
                        [body] => This is really exciting! Not.
                        [created] => 2008-02-13 18:34:57
                        [modified] =>
                    )
            )
    )
)
```

Las vistas en CakePHP se almacenan en la ruta `/app/View` y en un directorio con el mismo nombre que el controlador al que pertenecen, en nuestro caso `Posts`, así que para mostrar estos elementos formateados mediante una tabla tendremos algo como esto:

```
<!-- File: /app/View/Posts/index.ctp -->
<h1>Blog posts</h1>
<table>
  <tr>
    <th>Id</th>
    <th>Title</th>
    <th>Created</th>
```

(continué en la próxima página)

(proviene de la página anterior)

```

</tr>

<!-- Here is where we loop through our $posts array, printing out post info -->

<?php foreach ($posts as $post): ?>
<tr>
    <td><?php echo $post['Post']['id']; ?></td>
    <td>
        <?php echo $this->Html->link($post['Post']['title'],
array('controller' => 'posts', 'action' => 'view', $post['Post']['id'])); ?>
    </td>
    <td><?php echo $post['Post']['created']; ?></td>
</tr>
<?php endforeach; ?>
</table>

```

Esto debería ser sencillo de comprender.

Como habrás notado, hay una llamada a un objeto `$this->Html`. Este objeto es una instancia de una clase *Helper HtmlHelper*. CakePHP proporciona un conjunto de *Helpers* para ayudarte a completar acciones habituales, como por ejemplo realizar un link, crear un formulario, utilizar Javascript y Ajax de forma sencilla, etc. Puedes aprender más sobre esto en *Helpers* en otro momento. Basta con saber que la función `link()` generará un link HTML con el título como primer parámetro y la URL como segundo parámetro.

Cuando crees URLs en CakePHP te recomendamos emplear el formato de array. Se explica con detenimiento en la sección de *Routes*. Si utilizas estas rutas, podrás aprovecharte de las potentes funcionalidades de generación inversa de rutas de CakePHP en el futuro. Además puedes especificar rutas relativas a la base de tu aplicación de la forma “/controlador/accion/param1/param2”.

Llegados a este punto, deberías poder ver esta página si escribes la ruta a tu aplicación en el navegador, normalmente será algo así `http://localhost/blog/posts/index`. Deberías ver los posts correctamente formateados en una tabla.

Verás que si pinchas sobre alguno de los enlaces que aparecen en esta página (que van a una URL “/posts/view/some_id”, verás una página de error que te indica que la acción `view()` no ha sido definida todavía, y que debes definirla en el fichero `PostsController`. Si no ves ese error, algo ha ido mal, ya que esa acción no está definida y debería mostrar la página de error correspondiente. Cosa muy rara. Creemos esta acción para evitar el error:

```

class PostsController extends AppController {
    public $helpers = array('Html', 'Form');
    public $name = 'Posts';

    public function index() {
        $this->set('posts', $this->Post->find('all'));
    }

    public function view($id = null) {
        $this->set('post', $this->Post->findById($id));
    }
}

```

Si observas la función `view()`, ahora el método `set()` debería serte familiar. Verás que estamos usando `findById()` en vez de `find('all')` ya que sólo queremos un post concreto.

Verás que nuestra función `view` toma un parámetro (`$id`), que es el ID del artículo que queremos ver. Este parámetro

se gestiona automáticamente al llamar a la URL `/posts/view/3`, el valor “3” se pasa a la función `view` como primer parámetro `$id`.

Vamos a definir la vista para esta nueva función, como hicimos antes para `index()` salvo que el nombre ahora será `/app/View/Posts/view.ctp`.

```
<!-- File: /app/View/Posts/view.ctp -->

<h1><?php echo $post['Post']['title']?></h1>

<p><small>Created: <?php echo $post['Post']['created']?></small></p>

<p><?php echo $post['Post']['body']?></p>
```

Verifica que ahora funciona el enlace que antes daba un error desde `/posts/index` o puedes ir manualmente si escribes `/posts/view/1`.

Añadiendo artículos (*posts*)

Ya podemos leer de la base de datos nuestros artículos y mostrarlos en pantalla, ahora vamos a ser capaces de crear nuevos artículos y guardarlos.

Lo primero, añadir una nueva acción `add()` en nuestro controlador `PostsController`:

```
class PostsController extends AppController {
    public $helpers = array('Html', 'Form', 'Flash');
    public $components = array('Flash');

    public function index() {
        $this->set('posts', $this->Post->find('all'));
    }

    public function view($id) {
        $this->set('post', $this->Post->findById($id));
    }

    public function add() {
        if ($this->request->is('post')) {
            if ($this->Post->save($this->request->data)) {
                $this->Flash->success('Your post has been saved.');
                $this->redirect(array('action' => 'index'));
            }
        }
    }
}
```

Nota: Necesitas incluir el `FlashComponent` y `FlashHelper` en el controlador para poder utilizarlo. Si lo prefieres, puedes añadirlo en `AppController` y será compartido para todos los controladores que hereden de él.

Lo que la función `add()` hace es: si el formulario enviado no está vacío, intenta guardar un nuevo artículo utilizando el modelo `Post`. Si no se guarda bien, muestra la vista correspondiente, así podremos mostrar los errores de validación si el artículo no se ha guardado correctamente.

Cuando un usuario utiliza un formulario y efectúa un POST a la aplicación, esta información puedes accederla en `$this->request->data`. Puedes usar la función `pr()` o `debug()` para mostrar el contenido de esa variable y ver la pinta que tiene.

Utilizamos el `FlashComponent`, concretamente el método `FlashComponent::success()` para guardar el mensaje en la sesión y poder recuperarlo posteriormente en la vista y mostrarlo al usuario, incluso después de haber redirigido a otra página mediante el método `redirect()`. Esto se realiza a través de la función `FlashHelper::render()` que está en el layout, que muestra el mensaje y lo borra de la sesión para que sólo se vea una vez. El método `Controller::redirect` del controlador nos permite redirigir a otra página de nuestra aplicación, traduciendo el parámetro `array('action' => 'index')` a la URL `/posts`, y la acción `index`. Puedes consultar la documentación de este método aquí `Router::url()`. Verás los diferentes modos de indicar la ruta que quieres construir.

Al llamar al método `save()`, comprobará si hay errores de validación primero y si encuentra alguno, no continuará con el proceso de guardado. Veremos a continuación cómo trabajar con estos errores de validación.

Validando los datos

CakePHP te ayuda a evitar la monotonía al construir tus formularios y su validación. Todos odiamos teclear largos formularios y gastar más tiempo en reglas de validación de cada campo. CakePHP está aquí para echarnos una mano.

Para aprovechar estas funciones es conveniente que utilices el `FormHelper` en tus vistas. La clase `FormHelper` está disponible en tus vistas por defecto mediante llamadas del estilo `$this->Form`.

Nuestra vista sería así

```
<!-- File: /app/View/Posts/add.ctp -->

<h1>Add Post</h1>
<?php
echo $this->Form->create('Post');
echo $this->Form->input('title');
echo $this->Form->input('body', array('rows' => '3'));
echo $this->Form->end('Save Post');
?>
```

Hemos usado `FormHelper` para generar la etiqueta “form”. Esta llamada al `FormHelper` : `$this->Form->create()` generaría el siguiente código

```
<form id="PostAddForm" method="post" action="/posts/add">
```

Si `create()` no tiene parámetros al ser llamado, asume que estás creando un formulario que realiza el `submit` al método del controlador `add()` o al método `edit()` si hay un `id` en los datos del formulario. Por defecto el formulario se enviará por POST.

Las llamadas `$this->Form->input()` se usan para crear los elementos del formulario con el nombre que se pasa por parámetro. El primer parámetro indica precisamente el nombre del campo del modelo para el que se quiere crear el elemento de entrada. El segundo parámetro te permite definir muchas otras variables sobre la forma en la que se generará este `input field`. Por ejemplo, al enviar `array('rows' => '3')` estamos indicando el número de filas para el campo `textarea` que vamos a generar. El método `input()` está dotado de introspección y un poco de magia, ya que tiene en cuenta el tipo de datos del modelo al generar cada campo.

Una vez creados los campos de entrada para nuestro modelo, la llamada `$this->Form->end()` genera un botón de `submit` en el formulario y cierra el tag `<form>`. Puedes ver todos los detalles aquí [Helpers](#).

Volvamos atrás un minuto para añadir un enlace en `/app/View/Post/index.ctp` que nos permita agregar nuevos artículos. Justo antes del tag `<table>` añade la siguiente línea:

```
echo $this->Html->link('Add Post', array('controller' => 'posts', 'action' => 'add'));
```

Te estarás preguntando: ¿ Cómo le digo a CakePHP la forma en la que debe validar estos datos ? Muy sencillo, las reglas de validación se escriben en el modelo. Abre el modelo Post y vamos a escribir allí algunas reglas sencillas

```
class Post extends AppModel {
    public $name = 'Post';

    public $validate = array(
        'title' => array(
            'rule' => 'notEmpty'
        ),
        'body' => array(
            'rule' => 'notEmpty'
        )
    );
}
```

El array `$validate` contiene las reglas definidas para validar cada campo, cada vez que se llama al método `save()`. En este caso vemos que la regla para ambos campos es que no pueden ser vacíos `notEmpty`. El conjunto de reglas de validación de CakePHP es muy potente y variado. Podrás validar direcciones de email, codificación de tarjetas de crédito, incluso añadir tus propias reglas de validación personalizadas. Para más información sobre esto [Data Validation](#).

Ahora que ya tienes las reglas de validación definidas, usa tu aplicación para crear un nuevo artículo con un título vacío y verás cómo funcionan. Como hemos usado el método `FormHelper::input()`, los mensajes de error se construyen automáticamente en la vista sin código adicional.

Editando Posts

Seguro que ya le vas cogiendo el truco a esto. El método es siempre el mismo: primero la acción en el controlador, luego la vista. Aquí está el método `edit()`:

```
public function edit($id = null) {
    if (!$id) {
        throw new NotFoundException(__('Invalid post'));
    }

    $post = $this->Post->findById($id);
    if (!$post) {
        throw new NotFoundException(__('Invalid post'));
    }

    if ($this->request->is(array('post', 'put'))) {
        $this->Post->id = $id;
        if ($this->Post->save($this->request->data)) {
            $this->Flash->success(__('Your post has been updated.'));
            return $this->redirect(array('action' => 'index'));
        }
        $this->Flash->error(__('Unable to update your post.'));
    }

    if (!$this->request->data) {
```

(continué en la próxima página)

(proviene de la página anterior)

```

        $this->request->data = $post;
    }
}

```

Esta acción primero comprueba que se trata de un GET request. Si lo es, buscamos un *Post* con el id proporcionado como parámetro y lo ponemos a disposición para usarlo en la vista. Si la llamada no es GET, usaremos los datos que se envíen por POST para intentar actualizar nuestro artículo. Si encontramos algún error en estos datos, lo enviaremos a la vista sin guardar nada para que el usuario pueda corregirlos.

La vista quedará así:

```

<!-- File: /app/View/Posts/edit.ctp -->

<h1>Edit Post</h1>
<?php
    echo $this->Form->create('Post', array('action' => 'edit'));
    echo $this->Form->input('title');
    echo $this->Form->input('body', array('rows' => '3'));
    echo $this->Form->input('id', array('type' => 'hidden'));
    echo $this->Form->end('Save Post');

```

Mostramos el formulario de edición (con los valores actuales de ese artículo), junto a los errores de validación que hubiese.

Una cosa importante, CakePHP asume que estás editando un modelo si su id está presente en su array de datos. Si no hay un “id” presente, CakePHP asumirá que es un nuevo elemento al llamar a la función `save()`. Puedes actualizar un poco tu vista “index” para añadir los enlaces de edición de un artículo específico:

```

<!-- File: /app/View/Posts/index.ctp (edit links added) -->

<h1>Blog posts</h1>
<p><?php echo $this->Html->link("Add Post", array('action' => 'add')); ?></p>
<table>
    <tr>
        <th>Id</th>
        <th>Title</th>
        <th>Action</th>
        <th>Created</th>
    </tr>

<!-- Here's where we loop through our $posts array, printing out post info -->

<?php foreach ($posts as $post): ?>
    <tr>
        <td><?php echo $post['Post']['id']; ?></td>
        <td>
            <?php echo $this->Html->link($post['Post']['title'], array('action' => 'view
            =>', $post['Post']['id']));?>
        </td>
        <td>
            <?php echo $this->Form->postLink(
                'Delete',
                array('action' => 'delete', $post['Post']['id']),

```

(continué en la próxima página)

(proviene de la página anterior)

```

        array('confirm' => 'Are you sure?')
    )?>
    <?php echo $this->Html->link('Edit', array('action' => 'edit', $post['Post']
    => 'id'));?>
    </td>
    <td><?php echo $post['Post']['created']; ?></td>
    </tr>
<?php endforeach; ?>
</table>

```

Borrando Artículos

Vamos a permitir a los usuarios que borren artículos. Primero, el método en nuestro controlador:

```

function delete($id) {
    if (!$this->request->is('post')) {
        throw new MethodNotAllowedException();
    }
    if ($this->Post->delete($id)) {
        $this->Flash->success('The post with id: ' . $id . ' has been deleted.');
```

```

        $this->redirect(array('action' => 'index'));
    }
}

```

Este método borra un artículo cuyo “id” enviamos como parámetro y usa `$this->Flash->success()` para mostrar un mensaje si ha sido borrado. Luego redirige a “/posts/index”. Si el usuario intenta borrar un artículo mediante una llamada GET, generaremos una excepción. Las excepciones que no se traten, serán procesadas por CakePHP de forma genérica, mostrando una bonita página de error. Hay muchas excepciones a tu disposición *Exceptions* que puedes usar para informar de diversos problemas típicos.

Como estamos ejecutando algunos métodos y luego redirigiendo a otra acción de nuestro controlador, no es necesaria ninguna vista (nunca se usa). Lo que si querrás es actualizar la vista `index.ctp` para incluir el ya habitual enlace:

```

<!-- File: /app/View/Posts/index.ctp -->

<h1>Blog posts</h1>
<p><?php echo $this->Html->link('Add Post', array('action' => 'add')); ?></p>
<table>
    <tr>
        <th>Id</th>
        <th>Title</th>
        <th>Actions</th>
        <th>Created</th>
    </tr>

    <!-- Here's where we loop through our $posts array, printing out post info -->

    <?php foreach ($posts as $post): ?>
    <tr>
        <td><?php echo $post['Post']['id']; ?></td>

```

(continué en la próxima página)

(proviene de la página anterior)

```

        <td>
        <?php echo $this->Html->link($post['Post']['title'], array('action' => 'view',
        =>$post['Post']['id']));?>
        </td>
        <td>
        <?php echo $this->Form->postLink(
        'Delete',
        array('action' => 'delete', $post['Post']['id']),
        array('confirm' => 'Are you sure?'));
        ?>
        </td>
        <td><?php echo $post['Post']['created']; ?></td>
    </tr>
    <?php endforeach; ?>
</table>

```

Nota: Esta vista utiliza el FormHelper para pedir confirmación al usuario antes de borrar un artículo. Además el enlace para borrar el artículo se construye con Javascript para que se realice una llamada POST.

Rutas (*Routes*)

En muchas ocasiones, las rutas por defecto de CakePHP funcionan bien tal y como están. Los desarrolladores que quieren rutas diferentes para mejorar la usabilidad apreciarán la forma en la que CakePHP relaciona las URLs con las acciones de los controladores. Vamos a hacer cambios ligeros para este tutorial.

Para más información sobre las rutas, visita esta referencia [Routes Configuration](#).

Por defecto CakePHP responde a las llamadas a la raíz de tu sitio (por ejemplo `www.example.com/`) usando el controlador `PagesController`, y la acción `display"/home`". Esto muestra la página de bienvenida con información de CakePHP que ya has visto. Vamos a cambiar esto mediante una nueva regla.

Las reglas de enrutamiento están en `/app/Config/routes.php`. Comentaremos primero la regla de la que hemos hablado:

```
Router::connect('/', array('controller' => 'pages', 'action' => 'display', 'home'));
```

Como habíamos dicho, esta regla conecta la URL `"/` con el controlador `"pages"` la acción `"display"` y le pasa como parámetro `"home"`, así que reemplazaremos esta regla por esta otra:

```
Router::connect('/', array('controller' => 'posts', 'action' => 'index'));
```

Ahora la URL `"/` nos llevará al controlador `"posts"` y la acción `"index"`.

Nota: CakePHP también calcula las rutas a la inversa. Si en tu código pasas el array `array('controller' => 'posts', 'action' => 'index')` a una función que espera una url, el resultado será `"/`". Es buena idea usar siempre arrays para configurar las URL, lo que asegura que los links irán siempre al mismo lugar.

Conclusión

Creando aplicaciones de este modo te traerá paz, amor, dinero a carretas e incluso te conseguirá lo demás que puedas querer. Así de simple.

Ten en cuenta que este tutorial es muy básico, CakePHP tiene *muchas* otras cosas que harán tu vida más fácil, y es flexible aunque no hemos cubierto aquí estos puntos para que te sea más simple al principio. Usa el resto de este manual como una guía para construir mejores aplicaciones (recuerda todo los los beneficios que hemos mencionado un poco más arriba)

Ahora ya estás preparado para la acción. Empieza tu propio proyecto, lee el resto del manual y el API Manual API².

Lectura sugerida para continuar desde aquí

1. *Layouts*: Personaliza la plantilla *layout* de tu aplicación
2. *Elementos* Incluir vistas y reutilizar trozos de código
3. *Scaffolding*: Prototipos antes de trabajar en el código final
4. *Code Generation with Bake* Generación básica de CRUDs
5. *Authentication*: Gestión de usuarios y permisos

Lectura Adicional

Disecionando un Request típico en CakePHP

Ya hemos cubierto los ingredientes básicos de CakePHP, así que ahora vamos a ver cómo interactúan sus componentes para completar una petición de usuario o Request. Continuando con nuestro ejemplo anterior, imaginemos que nuestro amigo Ricardo acaba de pinchar en el link «Compra un Pastel Personalizado» en una aplicación CakePHP.

Diagrama: 2. Request típico CakePHP.

Negro = elemento requerido, Gris = elemento opcional, Azul = retorno (callback)

1. Ricardo pincha en el enlace que apunta a <http://www.example.com/cakes/buy>, y su navegador realiza una petición (request) al servidor web.
2. El Router interpreta la URL para extraer los parámetros para esta petición: el controlador, la acción y cualquier otro argumento que afecte a la lógica de negocio durante el request.
3. Usando las rutas, se construye una URL objetivo relacionada con una acción de un controlador (un método específico en una clase controlador). En este caso se trata del método `buy()` del controlador `CakesController`. El callback `beforeFilter()` de este controlador es invocado antes de ejecutar ningún otro método.
4. El controlador puede utilizar uno o varios modelos para acceder a los datos. En este ejemplo, el controlador utiliza un modelo para recuperar las últimas compras que ha hecho Ricardo de la Base de Datos. Cualquier callback del modelo, comportamiento (behavior), o DataSource que sea aplicable puede ser ejecutado en este momento. Aunque utilizar un modelo no es obligatorio, todos los controladores de CakePHP requieren inicialmente un modelo.
5. Una vez el modelo ha recuperado los datos, es devuelto al controlador. Se aplican aquí los callbacks del modelo.

² <https://api.cakephp.org/2.x/>

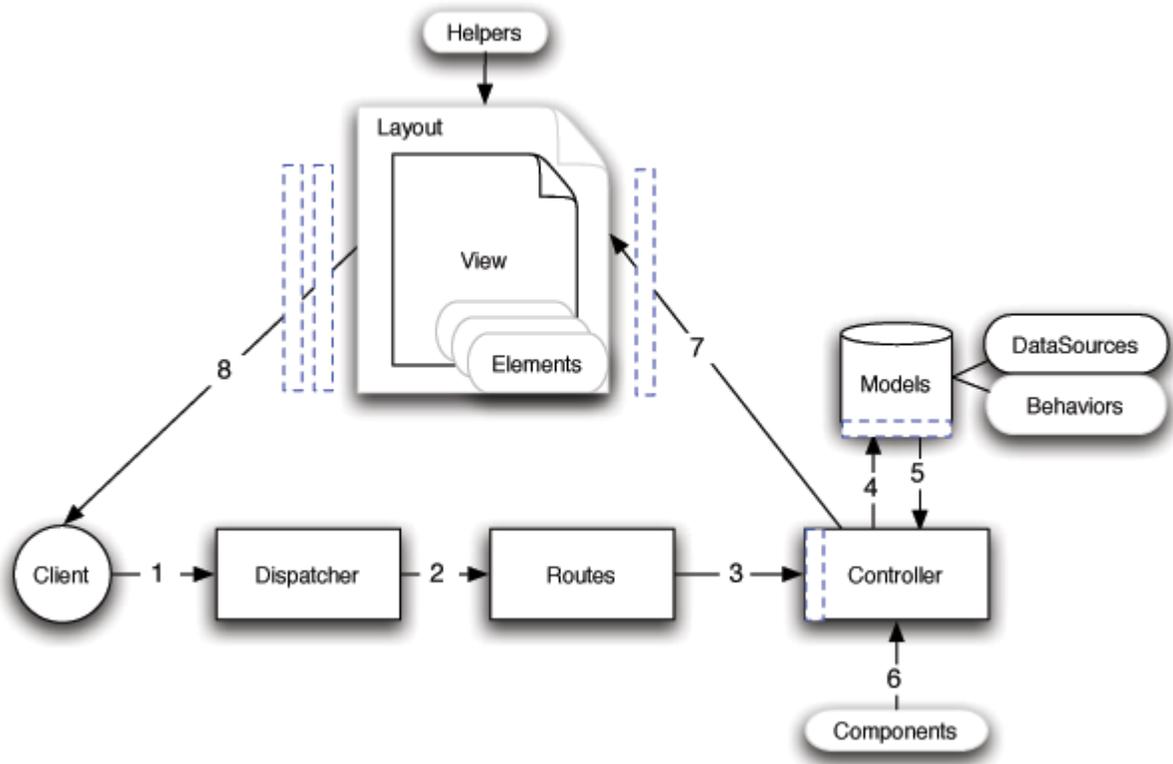


Figura 1: Diagrama de flujo que muestra un Request típico en CakePHP

6. El controlador puede utilizar componentes para refinar los datos o realizar otras operaciones (manipular la sesión, autenticación o enviar emails, por ejemplo).
7. Una vez el controlador ha empleado los modelos y componentes para preparar los datos, se envían a la vista utilizando el método set(). Los callback del controlador pueden ser ejecutados antes de que los datos sean enviados. La lógica de la vista se realiza en este punto. Esto puede incluir el uso de elementos (elements) y/o helpers. Por defecto, las vistas son generadas dentro de una plantilla (layout).
8. Callback adicionales pueden ejecutarse ahora (como afterFilter) en el controlador. La vista, ya generada por completo, se envía al navegador de Ricardo, que puede realizar su crítica compra de Pastel Personalizado.

Convenciones en CakePHP

(Nota del Traductor: Posiblemente la traducción de «conventions» sea muy literal. Queremos expresar el uso por defecto de determinados acuerdos que nos permiten establecer un marco común de trabajo).

Preferimos el uso de convenciones sobre la configuración. Aunque ocuparás algo de tu tiempo aprendiendo las convenciones usadas en CakePHP, ahorrarás mucho más en el camino. Cuando usas las convenciones, aprovechas funcionalidad gratuita y te liberas de la pesadilla de mantener los ficheros de configuración. Trabajar con convenciones también estandariza el proceso de despliegue de tu aplicación, permitiendo a otros desarrolladores conocer tu estructura más fácilmente.

Hemos empleado varios años y toda nuestra experiencia y buenas prácticas en la creación de estas convenciones. Ya sabes que, aunque te recomendamos que las sigas, puedes evitarlas con facilidad. Esto te resultará especialmente útil cuando trates con sistemas legados.

Convenciones en los Controladores

Nombre del Controlador en plural, CamelCased, y colocando Controller al final. PeopleController y LatestArticlesController son ejemplos que siguen esta convención.

El primer método a escribir en el controlador es `index()`. Cuando una petición vaya dirigida a este controlador, pero no se especifique acción, CakePHP ejecutará por defecto el método `index()`. Por ejemplo, la petición `http://example.com/apples/` será dirigida al método `index()` del controlador `ApplesController`, así como la llamada a `http://example.com/apples/view/` se mapeará al método `view()` de este mismo controlador.

Puedes cambiar la visibilidad de los métodos de CakePHP usando el carácter subrayado «`_`» al principio para ocultar su acceso directo desde la web, aunque será accesible internamente. Por ejemplo:

```
class NewsController extends AppController {

    function latest() {
        $this->_findNewArticles();
    }

    function _findNewArticles() {
        //Logic to find latest news articles
    }
}
```

El acceso a la url `http://example.com/news/latest` podrá realizarse con normalidad, mientras que al acceder a la url `http://example.com/news/_findNewArticles/` retornará un error, ya que este método está precedido por un «`_`». También puedes usar los modificadores de visibilidad de PHP (`private`, `protected`, `public`) para esto. Los métodos que no sean públicos, no podrán ser accedidos.

Consideraciones para las URL de los controladores

Como acabas de ver, los controladores que tienen nombres de una sola palabra se asignan a una URL en minúscula. Por ejemplo `ApplesController` (que se definirá en el fichero con nombre `ApplesController.php`) se accederá desde la URL `http://example.com/apples`.

Controladores cuyo nombre tiene varias palabras podrían ser asignados de cualquiera de estas formas

- `/redApples`
- `/RedApples`
- `/Red_apples`
- `/red_apples`

todos ellos resolverían al método `index` del controlador `RedApples`. De todos la convención es que esa url sea minúscula y subrayada, de este modo

- `/red_apples/go_pick` sería la url correcta para acceder a `RedApplesController::go_pick`

Para más información sobre URLs y parámetros en CakePHP, consulta [Routes Configuration](#).

Convenciones sobre nombres de fichero y nombres de clases

Como regla general, los nombres de fichero y los nombres de clase serán iguales, en formato CamelCased. Si tienes una clase **MyNiftyClass**, el fichero que la contiene se llamará **MyNiftyClass.php**. En el listado siguiente se muestran algunos ejemplos:

- El controlador con nombre **KissesAndHugsController** estará definido en el fichero **KissesAndHugsController.php**
- El componente con nombre **MyHandyComponent** estará en el fichero **MyHandyComponent.php**
- El modelo con nombre **OptionValue** estará en el fichero **OptionValue.php**
- El comportamiento (behavior) **EspecialyFunkableBehavior** estará en el fichero **EspecialyFunkableBehavior.php**
- La vista **SuperSimpleView** estará en el fichero **SuperSimpleView.php**
- El helper **BestEverHelper** estará, como ya habrás adivinado, en el fichero **BestEverHelper.php**

Cada uno de estos ficheros estará en la carpeta correspondiente bajo el directorio /app.

Convenciones para modelos y bases de datos

Los nombres de clase para los modelos serán CamelCased. Persona, GranPersona y SuperGranPersona, son ejemplos válidos para modelos.

Para las tablas en la base de datos se utiliza plural y el carácter subrayado (underscored) de esta forma: `gran_personas`, `super_gran_personas`. Verás que al leer los plurales en español, no tienen el sentido correcto. Ten en cuenta que esta convención proviene del inglés y si escribes los nombres de tus modelos en inglés, todo tiene mucho más sentido. Puedes saltarte esta convención en cualquier momento y escribir plurales más adecuados al español.

Puedes también usar la clase de utilidad *Inflexor* para comprobar el singular y plural de las palabras. Consulta la documentación aquí *Inflexor*.

Los nombres de los campos con más de una palabra se escriben en minúscula y subrayado, por ejemplo `first_name`.

Las claves foráneas o ajenas (foreign keys) en las relaciones “a muchos” (hasMany), “pertenece a” (belongsToMany) y “a uno” (hasOne) se reconocen por defecto si el nombre del campo se escribe usando el singular de la tabla con la que se relaciona y terminando en `_id`. Por ejemplo el modelo Baker tiene una relación “a muchos” con el modelo Cake. En la tabla `cakes` escribiremos un campo con el nombre `baker_id`. En caso de que el nombre de la tabla tenga varias palabras, como en `category_types`, la clave sería `category_type_id`.

Cuando la tabla es el resultado de una relación de “muchos a muchos” (HABTM o hasAndBelongsToMany), se nombrará utilizando el nombre de cada tabla de la relación, en orden alfabético y plural. Por ejemplo se usará `apples_zebras` en lugar de `zebras_apples`.

Todas las tablas que utilicemos en CakePHP, salvo las tablas de unión de las relaciones “muchos a muchos”, requieren una clave primaria en un único campo para identificar cada fila. Si necesitas que algún modelo no tenga clave primaria en un único campo, la convención es que añadas este campo a la tabla.

CakePHP no soporta claves primarias compuestas. Si quieres manipular directamente los datos de una tabla de unión, usa *query* y construye una query manualmente, o añade una clave primaria a la tabla para poder trabajar con ella como con un modelo normal. Ejemplo:

```
CREATE TABLE posts_tags (  
id INT(10) NOT NULL AUTO_INCREMENT,  
post_id INT(10) NOT NULL,  
tag_id INT(10) NOT NULL,  
PRIMARY KEY(id));
```

En vez de usar un campo numérico autoincremental como clave primaria, también puedes usar un `char(36)`. Si has definido así tu clave primaria, CakePHP gestionará esta clave añadiendo un UUID (`String::uuid`), que es un código único que identificará a cada registro, cada vez que realices un `Model::save` en ese modelo.

Convenciones en la vistas

Los nombres de las vistas son iguales a los del método del controlador al que hacen referencia, en formato subrayado. Por ejemplo el método `getReady()` del controlador `PeopleController` buscará el fichero de vista en la ruta `/app/View/People/get_ready.ctp`.

El patrón para nombrar las vistas es `/app/View/Controller/underscored_function_name.ctp`.

Si usas las convenciones de CakePHP para tu aplicación, ganas inmediatamente funcionalidad gratis, que se mantiene sola y no necesita tocar la configuración. Sirva para ilustrar esto un ejemplo:

- Tabla en la base de datos: «`people`»
- Nombre de Modelo: «`Person`» (es el singular de `people` para CakePHP), en el fichero `/app/Model/Person.php`
- Nombre del Controlador: «`PeopleController`», en el fichero `/app/Controller/PeopleController.php`
- Plantilla para la vista en el fichero `/app/View/People/index.ctp`

Si usas estas convenciones, CakePHP sabrá que una llamada a <http://example.com/people/> se mapeará a una llamada al método `index()` del controlador `PeopleController`, donde el modelo `Person` será instanciado automáticamente para su uso (leerá los datos de la tabla “`people`” en la base de datos). Ninguna de estas relaciones necesita ser creada ni configurada si nombras de la forma correcta los ficheros que de todos modos tienes que crear para que tu aplicación funcione.

Ahora conoces los fundamentos y convenciones que debes utilizar en CakePHP, te recomendamos que le eches un vistazo al [tutorial para hacer un blog](#) para ver cómo encajan estas piezas en una aplicación completa.

Estructura de directorios de CakePHP

Una vez descargado y cuando hayas descomprimido el fichero, estos son los directorios y ficheros que verás:

- `app`
- `lib`
- `vendors`
- `plugins`
- `.htaccess`
- `index.php`
- `README`

Verás que tienes tres directorios principales:

- `app` que es donde harás tu magia: aquí guardarás los ficheros de tu aplicación.
- `lib` que es donde nosotros hemos hecho «nuestra» magia. Haz una promesa ahora mismo: que nunca modificarás ficheros en esta carpeta. Si lo haces no podremos ayudarte ya que estás modificando el núcleo de CakePHP por tu cuenta.
- `vendors` que es donde puedes colocar los recursos externos que necesites para que tu aplicación funcione.

El directorio APP

En este directorio es donde realizarás la mayor parte del desarrollo de tu aplicación. Veamos el contenido de esta carpeta:

Config

Aquí están los (pocos) ficheros de configuración que usa CakePHP, concretamente la conexión con la base de datos, «bootstrapping» o el fichero de arranque, la configuración del núcleo y otros ficheros también de configuración estarán aquí.

Controller

Contiene los ficheros donde están definidos los Controladores de tu aplicación y los componentes.

Lib

Contiene recursos que no son de terceros o externos a tu aplicación. Esto te ayuda a separar tus librerías internas de las externas que estarán en la carpeta vendors.

Locale

Aquí están los ficheros de traducción para facilitar la internacionalización de tu proyecto.

Model

Contiene los modelos de tu aplicación, comportamientos (behaviors) y fuentes de datos (datasources).

Plugins

Contiene los plugins, increíble ¿ verdad ?

tmp

Aquí guarda CakePHP los datos temporales que se generan en ejecución. Los datos que se guardan dependen de tu configuración. Normalmente se almacenan las descripciones de los modelos, ficheros de registro (logs) y ficheros que almacenan las sesiones activas.

Este directorio debe existir y el usuario que ejecuta el servidor web debe ser capaz de escribir en esta ruta, de otro modo el rendimiento de tu aplicación puede reducirse enormemente. Cuando el parámetro *debug* está activo, CakePHP te advertirá si no se puede escribir en este directorio.

Vendors

Cualquier recurso de terceros o librerías PHP deben colocarse aquí. Si lo haces así, podrás importarlas luego cómodamente usando la función `App::import("vendor", "name")`. Si eres atento, te habrás fijado en que hay dos carpetas «Vendors», una aquí y otra en el directorio raíz de esta estructura. Entraremos en detalle sobre las diferencias cuando hablemos de configuraciones complejas de instalación de CakePHP. Por ahora, ten en cuenta que no nos gusta repetirnos, cada carpeta tiene un cometido distinto.

View

Los ficheros de presentación (vistas) se colocan aquí: elementos, páginas de error, helpers y plantillas (templates).

webroot

Este será el directorio raíz de tu aplicación web. Aquí habrá varias carpetas que te permitirán servir ficheros CSS, imágenes y JavaScript.

Estructura de CakePHP

CakePHP implementa las clases para controladores, modelos y vistas, pero también incluye otras clases y objetos que aceleran y facilitan el desarrollo en un framework MVC y lo hacen más ameno. Componentes, comportamientos y helpers, son clases que favorecen la extensibilidad y reutilización de tu código entre proyectos. Empezaremos con una visión de alto nivel y luego examinaremos los detalles de cada parte.

Extensiones para las Aplicaciones

Los controladores, helpers, modelos tienen todos una clase padre que puedes emplear para definir cambios en toda tu aplicación. `AppController`, que está en `/app/Controller/AppController.php`, `AppHelper` en `/app/View/Helper/AppHelper.php` y `AppModel` en `/app/Model/AppModel.php` son lugares apropiados para colocar métodos que quieras compartir entre todos tus controladores, helpers y modelos.

Aunque no sean clases ni ficheros, las rutas juegan un papel importante en las peticiones que se realizan a CakePHP. Las definiciones de rutas le indican al sistema cómo debe mapear las URLs a las acciones de los controladores. El comportamiento por defecto es asumir que la ruta `/controller/action/var1/var2` se mapea a `Controller::action($var1, $var2)`, pero puedes usar las rutas para personalizar esto y definir cómo quieres que se interprete cada URL.

Algunas funcionalidades de tu aplicación se merecen ser empaquetadas para ser usadas como un conjunto. Un plugin es un paquete de modelos, controladores, vistas, etc. que siguen un objetivo común que puede reutilizarse en otros proyectos. Un sistema de gestión de usuarios o un blog podrían ser buenos candidatos para escribir un plugin y utilizarlo en múltiples proyectos.

Extendiendo los controladores («Components»)

Un componente es una clase que da soporte a la lógica de los controladores. Si tienes lógica que quieres reutilizar entre controladores, o proyectos, un componente es el lugar ideal para hacerlo. Por ejemplo, `EmailComponent` es un componente de CakePHP que te permite crear y enviar emails de forma sencilla. En vez de escribir el código para ello en un controlador o varios, se ha empaquetado en un componente reutilizable.

Los controladores poseen «callbacks». Estos callbacks te permiten inyectar funcionalidad en el proceso normal de CakePHP. Los callbacks disponibles incluyen:

- `beforeFilter()`, se ejecuta antes de cualquier otra función.
- `beforeRender()`, se ejecuta tras la función del controlador, y antes de que se genere la vista.
- `afterFilter()`, se ejecuta después de toda la lógica del controlador, incluso después de que se haya generado la vista. No hay diferencia entre `afterRender()` y `afterFilter()` a no ser que llames manualmente al método `render()` en tu controlador y hayas incluido algún código después de esta llamada.

Extensiones para los modelos («Behaviors»)

De forma similar, los comportamientos o «behaviors» son formas de compartir funcionalidades entre los modelos. Por ejemplo, si guardas datos de usuario en una estructura tipo árbol, puedes especificar que tu modelo `Usuario` se comporte como un árbol, y obtener gratis las funciones para eliminar, añadir, e intercambiar nodos en tu estructura.

Los modelos también son potenciados por otra clase llamada fuente de datos o `DataSource`. Las fuentes de datos son una abstracción que permite a los modelos manipular diferentes tipos de datos de manera consistente. La fuente de datos habitual en una aplicación CakePHP es una base de datos relacional. Puedes escribir fuentes de datos adicionales para representar «feeds» RSS, ficheros CSV, servicios LDAP o eventos iCal. Las fuentes de datos te permiten asociar registros de diferentes orígenes: en vez de estar limitado a consultas SQL, las fuentes de datos te permitirían decirle a tu modelo LDAP que está asociado a múltiples eventos iCal.

Igual que los controladores, los modelos poseen callbacks:

- beforeFind()
- afterFind()
- beforeValidate()
- beforeSave()
- afterSave()
- beforeDelete()
- afterDelete()

Los nombres de estos métodos deberían ser descriptivos por sí mismos. Encontrarás todos los detalles en el capítulo que habla sobre los modelos.

Extendiendo las vistas («Helpers»)

Un helper es una clase que sirve de apoyo a las vistas. De forma similar a los componentes y los controladores, los helpers permiten que la lógica que usas en las vistas se pueda acceder desde otras vistas o en otras aplicaciones. Uno de los helpers que se distribuye con CakePHP es AjaxHelper, permite que se realicen llamadas Ajax desde las vistas de forma mucho más sencilla.

La mayoría de aplicaciones tiene trozos de código que se usan una y otra vez. CakePHP facilita la reutilización de código con plantillas y elementos. Por defecto cada vista que se genera por un controlador se incrusta dentro de una plantilla. Los elementos se usan como un modo sencillo de empaquetar código para poder ser usado en cualquier vista.

Instalación

CakePHP se instala de forma fácil y rápidamente. Los requisitos mínimos son: un servidor web y una copia de los archivos de CakePHP ¡Eso es todo! Aunque este manual se centra en la configuración de Apache, ya que es el servidor web más común, se puede configurar CakePHP para que funcione en otros servidores como lighttpd o Microsoft IIS.

Vamos a preparar el proceso de instalación, que consta de los siguientes pasos:

- Descargar copia de CakePHP
- Configurar el servidor web para que use PHP
- Comprobar que los permisos de ficheros y carpetas son correctos

Descargar CakePHP

Hay dos opciones: descargar un archivo comprimido con todo el código (zip/tar.gz/tar.bz2) de la web oficial o realizar un *checkout* del código directamente desde el repositorio de git.

Para descargar la última versión estable, puedes visitar la página oficial <https://cakephp.org> y pinchar en la opción «Download Now».

Además, todas las versiones de CakePHP están hospedadas en [Github](https://github.com/cakephp/cakephp)³. Github almacena tanto el código de CakePHP como muchos otros plugins para el sistema. Las versiones *release* de CakePHP están disponibles aquí [Github tags](https://github.com/cakephp/cakephp/tags)⁴.

También se puede obtener la última versión, con las últimas correcciones de errores y mejoras de última hora (o al menos en ese día). Para ello puedes clonar el repositorio. [Github](https://github.com/cakephp/cakephp)⁵.

³ <https://github.com/cakephp/cakephp>

⁴ <https://github.com/cakephp/cakephp/tags>

⁵ <https://github.com/cakephp/cakephp>

Permisos

CakePHP usa el directorio `/app/tmp` para varias cosas, como guardar las descripciones de los modelos, la cache de las vistas y la información de sesión de los usuarios.

Por ello debes, asegurarte de que el directorio `/app/tmp` de tu instalación de CakePHP puede ser escrito por el usuario que ejecuta tu servidor web. Ten en cuenta que cuando tu servidor web se inicia, define un usuario como propietario del servicio. Este usuario suele llamarse “apache” o “www-data” en algunas versiones de sistemas *nix. Por lo tanto la carpeta `/app/tmp` debe tener permisos de escritura para que el usuario propietario del servidor web pueda escribir dentro de ella.

Configuración

Configurar CakePHP es tan sencillo como copiar la carpeta en la carpeta raíz de tu servidor web (*document root*) o tan complejo y flexible como quieras para que se adapte a tu sistema. En esta sección cubriremos los 3 modos más frecuentes: desarrollo, producción, avanzado.

- Desarrollo: fácil y rápido. Las URL de tu aplicación incluyen la carpeta de instalación de CakePHP. Es menos seguro.
- Producción: Requiere poder cambiar el *document root* de su servidor web, proporciona URL amigables y es muy seguro.
- Avanzado: Te permite colocar la carpeta de CakePHP en otras partes de tu sistema de archivos, posiblemente para compartir el núcleo con varias aplicaciones web basadas en CakePHP.

Desarrollo

Instalar CakePHP para desarrollo es el método más rápido de empezar. Este ejemplo te ayudará a instalar una aplicación CakePHP y configurarla para que se accese desde http://www.example.com/cake_2_0/. Asumiremos que tu *document root* (la carpeta raíz de tu servidor web) es `/var/www/html`.

Descomprime los contenidos del archivo que contiene CakePHP en la carpeta `/var/www/html`. Ahora tendrás un nuevo directorio con el nombre de la versión que te has descargado (por ejemplo `cake_2.0.0`). Cambia el nombre de este directorio a algo más sencillo, por ejemplo a `cake20`. La estructura de directorios debería ser ahora similar a esta:

- `/var/www/html`
- `/cake20`
 - `/app`
 - `/lib`
 - `/vendors`
 - `/plugins`
 - `/.htaccess`
 - `/index.php`
 - `/README`

Si la configuración de tu servidor web es correcta, ahora podrás acceder a tu aplicación accediendo a: <http://localhost/cake20>.

Usando una misma instalación de CakePHP para múltiples aplicaciones

Si estás desarrollando varias aplicaciones a la vez, muchas veces tiene sentido compartir la misma versión del núcleo de CakePHP. Hay varias formas de conseguirlo. Una de las más sencillas es usar la directiva PHP `include_path`. Para empezar, clona CakePHP en un directorio. Por ejemplo, en `/home/mark/projects`:

```
git clone git://github.com/cakephp/cakephp.git /home/mark/projects/cakephp
```

Este comando clonará CakePHP en tu directorio `/home/mark/projects`. Si no quieres usar git, puedes descargar el archivo zip del repositorio, todos los demás pasos serán los mismos. Lo siguiente es modificar el archivo de configuración de PHP `php.ini`. En sistemas *nix, este archivo suele estar ubicado en la ruta `/etc/php.ini`, pero puedes localizarlo fácilmente mediante el comando `php -i`, busca la ruta bajo el epígrafe “Loaded Configuration File”. Cuando hayas localizado el fichero correcto, modifica el parámetro `include_path` y añade el directorio `/home/mark/projects/cakephp/lib`. Ejemplo:

```
include_path = ./home/mark/projects/cakephp/lib:/usr/local/php/lib/php
```

Reinicia el servidor web, deberías ver los cambios aplicados en la salida de `phpinfo()`.

Nota: Si utilizas Windows, separa las rutas en el include path con `;` en vez de `:`

Tras modificar este parámetro y reiniciar el servidor web, tus aplicaciones podrán utilizar CakePHP automáticamente.

Producción

Se llama entorno de Producción porque es el lugar al que accederán los usuarios finales de la aplicación web. Una instalación en Producción es una forma más flexible de configurar CakePHP. Usando este método permite que un dominio completo actúe como una única aplicación CakePHP. El siguiente ejemplo permitirá ayudar a instalar CakePHP en cualquier parte del sistema de archivos y tener la aplicación disponible en <http://www.example.com>. Ten en cuenta que esta instalación requiere que tengas permiso de escritura en el directorio raíz de tu servidor web *document root*.

Descomprime los contenidos del paquete que has descargado con la última versión de CakePHP en el directorio que prefieras. No es necesario que sea una carpeta de tu *document root*. Por ejemplo vamos a suponer que quieres tener tus archivos de CakePHP en la ruta `/cake_install`. Tu sistema de archivos sería entonces:

- `/cake_install/`
 - `/app`
 - `/webroot` (este directorio es el que configuraremos como `DocumentRoot` en el servidor web)
 - `/lib`
 - `/vendors`
 - `/.htaccess`
 - `/index.php`
 - `/README`

Si usas Apache, ahora es el momento de configurar la directiva de configuración `DocumentRoot` de tu servidor web para que apunte a la carpeta `/app/webroot` de tu instalación.

```
DocumentRoot /cake_install/app/webroot
```

Si tu servidor está correctamente configurado, podrás acceder a tu aplicación utilizando la url <http://www.example.com>.

Instalación avanzada y configuración flexible

Instalación Avanzada

Hay muchas situaciones en las que te gustaría colocar los archivos de CakePHP en otro directorio de tu sistema de ficheros. Esto puede pasarte por restricciones en tu hosting compartido, o simplemente quieres que varias aplicaciones compartan la misma versión de CakePHP. Esta sección describe cómo configurar los directorios de CakePHP para que se ajusten a tus requisitos.

Lo primero, ten en cuenta que hay tres partes en toda aplicación CakePHP:

1. Los ficheros propios del framework, en /cake
2. El código específico de tu aplicación, en /app
3. El directorio raíz de tu aplicación, habitualmente en /app/webroot

Cada uno de estos directorios puede estar donde quieras dentro de tu sistema de ficheros, con la excepción del directorio raíz (*webroot*), que tiene que ser accesible por tu servidor web. Puedes moverlo fuera de /app siempre que le digas a CakePHP dónde está.

Cambia los siguientes ficheros si quieres configurar CakePHP para que funcione con una estructura de directorios diferente.

- /app/webroot/index.php
- /app/webroot/test.php (si usas tests [Testing](#) .)

Hay 3 constantes que necesitarás cambiar: `ROOT`, `APP_DIR` y `CAKE_CORE_INCLUDE_PATH`.

- `ROOT` debe apuntar a la carpeta que contiene tu directorio app.
- `APP_DIR` debería ser el nombre base de tu directorio app.
- `CAKE_CORE_INCLUDE_PATH` debe apuntar al directorio que contiene CakePHP.

Veamos todo esto con un ejemplo. Imagina que quiero crear una estructura de directorios como sigue:

- La instalación de CakePHP la quiero en /usr/lib/cake.
- Mi directorio raíz *webroot* lo colocaré en /var/www/mysite/.
- Mi directorio app con el código de mi aplicación lo colocaré en /home/me/myapp.

Para llevar esto a cabo, necesitaré editar el fichero /var/www/mysite/index.php para que se parezca a este:

```
// /app/webroot/index.php (partial, comments removed)

if (!defined('ROOT')) {
    define('ROOT', DS . 'home' . DS . 'me');
}

if (!defined('APP_DIR')) {
    define ('APP_DIR', 'myapp');
}

if (!defined('CAKE_CORE_INCLUDE_PATH')) {
    define('CAKE_CORE_INCLUDE_PATH', DS . 'usr' . DS . 'lib');
}
```

Recomendamos utilizar la constante `DS` en vez del caracter “/” para delimitar las rutas de directorios. Esto permite que tu código sea más portable ya que este caracter cambia en algunos sistemas operativos. Usa `DS`.

Apache, mod_rewrite y .htaccess

CakePHP está escrito para funcionar con mod_rewrite sin tener que realizar ningún cambio. Normalmente ni te darás cuenta de que ya está funcionando, aunque hemos visto que para algunas personas es un poco más complicado configurarlo para que funcione bien en su sistema.

Te proponemos algunas cosas que te pueden ayudar a que quede bien configurado.

Lo primero: echa un vistazo a tu fichero de configuración de Apache httpd.conf (asegúrate de estar editando el fichero correcto, ya que puede haber ficheros de este tipo por usuario o por sitio web. Edita el fichero de configuración principal).

1. Debe estar permitido la reescritura de ficheros .htaccess (*override*), y el parámetro AllowOverride debe estar fijado a "All" para el DocumentRoot en el que reside tu aplicación web. Deberías ver algo similar a esto:

```
# Cada directorio al que tiene acceso Apache debe ser configurado
# para indicar qué funciones están habilitadas y deshabilitadas
#
# Primero se configura un directorio por defecto restringido por seguridad
#
<Directory />
    Options FollowSymLinks
    AllowOverride All
#   Order deny,allow
#   Deny from all
</Directory>
```

1. Comprueba que efectivamente se está cargando mod_rewrite ya que en algunos sistemas viene desactivado por defecto en Apache. Para ello deberías ver la siguiente línea *sin* comentario (“#”) al principio:

```
LoadModule rewrite_module libexec/apache2/mod_rewrite.so
```

Si ves que tiene un comentario al principio de la línea, quítalo. Si has tenido que hacer algún cambio a este fichero, necesitarás reiniciar el servicio Apache.

Verifica que los ficheros .htaccess están ahí.

A veces, al copiar archivos de un lugar a otro los ficheros con un nombre que empieza por “.” se consideran ocultos y no se copian. Hay que forzar la copia de estos ficheros.

1. Asegúrate de que tu copia de CakePHP es de nuestro sitio oficial o del repositorio oficial de GIT, y que la has descomprimido correctamente.

En el directorio raíz de CakePHP (necesita ser copiado a tu carpeta, esto redirige todo a tu aplicación CakePHP):

```
<IfModule mod_rewrite.c>
    RewriteEngine on
    RewriteRule ^$ app/webroot/ [L]
    RewriteRule (.* ) app/webroot/$1 [L]
</IfModule>
```

En el directorio app (será copiado en tu directorio de aplicación por bake):

```
<IfModule mod_rewrite.c>
    RewriteEngine on
    RewriteRule ^$ webroot/ [L]
    RewriteRule (.* ) webroot/$1 [L]
</IfModule>
```

En el directorio raíz *webroot* (será copiado allí por bake):

```
<IfModule mod_rewrite.c>
  RewriteEngine On
  RewriteCond %{REQUEST_FILENAME} !-d
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteRule ^(.*)$ index.php [QSA,L]
</IfModule>
```

Muchos de las empresas de hosting (GoDaddy, 1and1) ya tienen `mod_rewrite` activo y su servidor web ya utiliza un directorio de usuario para servir el contenido. Si estás instalando CakePHP en un directorio de usuario, por ejemplo (<http://example.com/~username/cakephp/>) o cualquier otra ruta que ya utilice `mod_rewrite` necesitarás añadir una directiva `RewriteBase` a los ficheros `.htaccess` que se utilizan (todos).

Nota: Si al cargar la página de bienvenida de CakePHP ves que no se aplican bien los estilos, puede que necesites esta directiva `RewriteBase` en tus ficheros `.htaccess`.

Para añadir la directiva, abre los 3 ficheros `.htaccess` y escribe la nueva directiva bajo la línea `RewriteEngine` (dentro del `IfModule` para que tu fichero de configuración sólo se aplique si `mod_rewrite` está cargado):

```
<IfModule mod_rewrite.c>
  RewriteEngine On
  RewriteBase /path/to/cake/app
  RewriteCond %{REQUEST_FILENAME} !-d
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteRule ^(.*)$ index.php [QSA,L]
</IfModule>
```

Este cambio dependerá de tu configuración. Puede que debas realizar otros cambios en función de tu servidor. Para aclarar dudas, consulta la documentación de Apache.

URLs amigables y Lighttpd

Aunque `Lighttpd` incluye un módulo de redirección, no es igual que `mod_rewrite` de Apache. Para que funcionen las URLs del mismo modo, tienes dos opciones:

- Usar `mod_rewrite`
- Usar un script LUA y `mod_magnet`

Usando `mod_rewrite` La manera más sencilla es añadir este script a la configuración de `lighty`. Sólo edita la URL y todo debería ir bien. Ten en cuenta que esto no funciona si CakePHP ha sido instalado en subdirectorios.

```
$HTTP["host"] =~ "^(\www\.)?example.com$" {
  url.rewrite-once = (
    # if the request is for css/files etc, do not pass on to Cake
    "^/(css|files|img|js)/(.*)" => "/$1/$2",
    "^([\^?]*)(\?(.+))?$" => "/index.php/$1&$3",
  )
  evhost.path-pattern = "/home/%2-%1/www/www/%4/app/webroot/"
}
```

Usando `mod_magnet` Coloca este script lua en `/etc/lighttpd/cake`.

```

-- little helper function
function file_exists(path)
    local attr = lighty.stat(path)
    if (attr) then
        return true
    else
        return false
    end
end
function removePrefix(str, prefix)
    return str:sub(1,#prefix+1) == prefix.."/" and str:sub(#prefix+2)
end

-- prefix without the trailing slash
local prefix = ''

-- the magic ;)
if (not file_exists(lighty.env["physical.path"])) then
    -- file still missing. pass it to the fastcgi backend
    request_uri = removePrefix(lighty.env["uri.path"], prefix)
    if request_uri then
        lighty.env["uri.path"] = prefix .. "/index.php"
        local uriquery = lighty.env["uri.query"] or ""
        lighty.env["uri.query"] = uriquery .. (uriquery ~= "" and "&" or "") .. "url=" ..
↪request_uri
        lighty.env["physical.rel-path"] = lighty.env["uri.path"]
        lighty.env["request.orig-uri"] = lighty.env["request.uri"]
        lighty.env["physical.path"] = lighty.env["physical.doc-root"] .. lighty.env[
↪"physical.rel-path"]
    end
end
end
-- fallback will put it back into the lighty request loop
-- that means we get the 304 handling for free. ;)

```

y escribe la nueva directiva bajo la línea RewriteEngine (dentro del IfModule para que tu fichero de configuración sólo se aplique si mod_rewrite está cargado):

```

<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteBase /path/to/cake/app
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^(.*)$ index.php/$1 [QSA,L]
</IfModule>

```

Este cambio dependerá de tu configuración. Puede que debas realizar otros cambios en función de tu servidor. Para aclarar dudas, consulta la documentación de Apache.

Nota: Si has instalado CakePHP en un subdirectorio, debes colocar set prefix = “nombre-del-subdirectorio” el el script anterior.

Luego Lighttpd para tu host virtual:

```

$HTTP["host"] =~ "example.com" {
    server.error-handler-404 = "/index.php"

    magnet.attract-physical-path-to = ( "/etc/lighttpd/cake.lua" )

    server.document-root = "/var/www/cake-1.2/app/webroot/"

    # Think about getting vim tmp files out of the way too
    url.access-deny = (
        "~", ".inc", ".sh", "sql", ".sql", ".tpl.php",
        ".xhtml", "Entries", "Repository", "Root",
        ".ctp", "empty"
    )
}

```

¡ y listo !

URLs amigables en nginx

nginx es un servidor web que está ganando mucha popularidad. Igual que Lighttpd, usa eficientemente los recursos del sistema. En el caso de nginx, no hay ficheros .htaccess, así que es necesario crear esas reglas de redirección directamente en la configuración del servidor. Dependiendo de tu configuración igual tendrás que ajustar un poco este fichero. Como mínimo necesitarás PHP funcionando como instancia FastCGI. Puedes ver los detalles en la documentación de instalación de nginx.

```

server {
    listen 80;
    server_name www.example.com;
    rewrite ^(.*) http://example.com$1 permanent;
}

server {
    listen 80;
    server_name example.com;

    # root directive should be global
    root /var/www/example.com/public/app/webroot/;
    index index.php;

    access_log /var/www/example.com/log/access.log;
    error_log /var/www/example.com/log/error.log;

    location / {
        try_files $uri $uri/ /index.php?$uri&$args;
    }

    location ~ \.php$ {
        include /etc/nginx/fastcgi_params;
        try_files $uri =404;
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    }
}

```

(continué en la próxima página)

(proviene de la página anterior)

```
}
}
```

IIS7 También existe (Windows hosts)

No nos olvidamos de que muchos utilizan IIS como servidor web. IIS no soporta de forma nativa los ficheros .htaccess. Hay algunos “add-ons” que te permiten añadir esta funcionalidad. También puedes importar las reglas de redirección de los ficheros .htaccess en IIS y usar la reescritura nativa de CakePHP. Para hacer esto último, sigue estos pasos:

1. Usa el *Microsoft's Web Platform Installer* para instalar el módulo *URL Rewrite Module 2.0*.
2. Crea un nuevo fichero en la carpeta de CakePHP, llamado web.config.
3. Usa notepad o cualquier otro editor “seguro” para ficheros xml que conserve el formato. Copia el siguiente código:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <system.webServer>
    <rewrite>
      <rules>
        <rule name="Imported Rule 1" stopProcessing="true">
          <match url="^(.*)$" ignoreCase="false" />
          <conditions logicalGrouping="MatchAll">
            <add input="{REQUEST_FILENAME}" matchType="IsDirectory" negate=
↪ "true" />
            <add input="{REQUEST_FILENAME}" matchType="IsFile" negate="true" ↪
↪ />
          </conditions>

          <action type="Rewrite" url="index.php?url={R:1}" appendQueryString="true" />

        </rule>

        <rule name="Imported Rule 2" stopProcessing="true">
          <match url="^$" ignoreCase="false" />
          <action type="Rewrite" url="/" />
        </rule>
        <rule name="Imported Rule 3" stopProcessing="true">
          <match url="(.*)" ignoreCase="false" />
          <action type="Rewrite" url="/{R:1}" />
        </rule>
        <rule name="Imported Rule 4" stopProcessing="true">
          <match url="^(.*)$" ignoreCase="false" />
          <conditions logicalGrouping="MatchAll">
            <add input="{REQUEST_FILENAME}" matchType="IsDirectory" negate=
↪ "true" />
            <add input="{REQUEST_FILENAME}" matchType="IsFile" negate="true" ↪
↪ />
          </conditions>
          <action type="Rewrite" url="index.php/{R:1}" appendQueryString="true" />
        </rule>
      </rules>
    </rewrite>
  </system.webServer>
</configuration>
```

(continué en la próxima página)

(proviene de la página anterior)

```
</rules>
</rewrite>
</system.webServer>
</configuration>
```

También puedes usar la funcionalidad “Importar” en el módulo de reescritura de IIS, para importar directamente las reglas de todos los ficheros .htaccess de CakePHP. Si importas las reglas de este modo, IIS creará el fichero web.config. Es posible que necesites retocar un poco esta configuración hasta que funcione.

Una vez creado el archivo web.config con la configuración correcta de reglas de reescritura para IIS, los links, css, js y enrutado de CakePHP deberían funcionar correctamente.

URL Rewriting

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁶ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

¡ A por todas !

Vamos a ver de qué es capaz tu recientemente instalado CakePHP. Dependiendo de qué opción de configuración hayas elegido, podrás acceder a tu aplicación mediante <http://www.example.com/> o http://example.com/cake_install/. Verás una página de bienvenida por defecto, que mostrará un mensaje que te dice el estado actual de conexión a la Base de Datos.

¡ Enhorabuena ! Estás preparado para empezar.

¿ No funciona ? Bueno, estas cosas pasan. Si aparece un mensaje de error que habla de la Zona Horaria *timezone*, quita el comentario en la siguiente línea del fichero app/Config/core.php:

```
/**
 * If you are on PHP 5.3 uncomment this line and correct your server timezone
 * to fix the date & time related errors.
 */
date_default_timezone_set('UTC');
```

⁶ <https://github.com/cakephp/docs>

Introducción a CakePHP

Bienvenido al Cookbook, el manual web del framework CakePHP hace que el desarrollo de aplicaciones sea pan comido.

Este manual asume que tienes una comprensión general de PHP y un conocimiento básico de programación orientada a objetos (POO). Las diferentes funcionalidades que este framework posee hace uso de tecnologías como SQL, JavaScript y XML y este manual no trata de explicarlas, sino sólo la forma en que se utilizan en su contexto.

¿Qué es CakePHP? y ¿Por qué usarlo?

CakePHP⁷ es un framework⁸ libre⁹, de código abierto¹⁰, para el desarrollo rápido de aplicaciones¹¹ para PHP¹². Es una estructura fundamental para la ayudar a los programadores a crear aplicaciones web. Nuestro objetivo principal es permitirte trabajar de forma estructurada y rápida y sin pérdida de flexibilidad.

CakePHP pone tu disposición todas las herramientas que necesita para empezar a programar lo que realmente hay que hacer: la lógica específica de tu aplicación. En lugar de reinventar la rueda cada vez que te sientas a hacer un nuevo proyecto, obtén una copia de CakePHP y empieza con el verdadero corazón de tu aplicación.

CakePHP tiene un *equido de desarrollo activo*¹³ y una comunidad muy viva, lo que le da un gran valor al proyecto. Además de no tener que reinventar la rueda, usar CakePHP significa que el núcleo de la aplicación estará bien probado y está siendo constantemente mejorado.

He aquí una lista rápida de las características que disfrutarás al utilizar CakePHP:

- *Licencia flexible*¹⁴

⁷ <https://cakephp.org/>

⁸ https://en.wikipedia.org/wiki/Application_framework

⁹ https://en.wikipedia.org/wiki/MIT_License

¹⁰ https://en.wikipedia.org/wiki/Open_source

¹¹ https://en.wikipedia.org/wiki/Rapid_application_development

¹² <https://www.php.net/>

¹³ <https://github.com/cakephp/cakephp/contributors>

¹⁴ https://en.wikipedia.org/wiki/MIT_License

- Compatible con las versiones de PHP 5.2.6 y superiores.
- Contiene **CRUD**¹⁵ para la interacción de la base de datos.
- **Andamiaje de código**¹⁶.
- Generación automática de código.
- **Arquitectura MVC**¹⁷
- URLs personalizadas
- Función de **Validación**¹⁸.
- **Plantillas rápidas y flexibles**¹⁹ (La sintaxis de PHP, con ayudantes).
- Ayudantes para AJAX, JavaScript, formularios HTML y más.
- Componentes de Email, Cookie, Seguridad, Sesión y otros.
- **ACL**²⁰ flexible.
- Sanitización de Datos.
- Poderoso **Caché**²¹.
- Localización e Internacionalización.
- Funciona desde cualquier directorio de sitios web, con poca o ninguna configuración adicional.

Entendiendo el Modelo - Vista - Controlador

CakePHP sigue el patrón diseño de software llamado **MVC**²². Programar usando MVC separa tu aplicación en tres partes principalmente:

La capa del Modelo

El modelo representa la parte de la aplicación que implementa la lógica de negocio. Ésto significa que es responsable de la recuperación de datos convirtiéndolos en conceptos significativos para la aplicación, así como su procesamiento, validación, asociación y cualquier otra tarea relativa a la manipulación de dichos datos.

A primera vista los objetos del modelo puede ser considerados como la primera capa de la interacción con cualquier base de datos que podría estar utilizando tu aplicación. Pero en general representan los principales conceptos en torno a los cuales se desea implementar un programa.

En el caso de una red social, la capa de modelo se haría cargo de tareas tales como guardar datos del usuario, el almacenamiento de asociaciones con amigos, el almacenamiento y la recuperación de fotos de los usuarios, encontrar sugerencias de nuevos amigos, etc. Mientras que los objetos del modelo pueden ser considerados como «Usuario», «Comentario» y «Foto».

¹⁵ https://en.wikipedia.org/wiki/Create,_read,_update_and_delete

¹⁶ [https://en.wikipedia.org/wiki/Scaffold_\(programming\)](https://en.wikipedia.org/wiki/Scaffold_(programming))

¹⁷ <https://en.wikipedia.org/wiki/Model-view-controller>

¹⁸ https://en.wikipedia.org/wiki/Data_validation

¹⁹ https://en.wikipedia.org/wiki/Web_template_system

²⁰ https://en.wikipedia.org/wiki/Access_control_list

²¹ https://en.wikipedia.org/wiki/Web_cache

²² <https://en.wikipedia.org/wiki/Model-view-controller>

La capa de la Vista

La vista hace una presentación de los datos del modelo estando separada de los objetos del modelo. Es responsable del uso de la información de la cual dispone para producir cualquier interfaz de presentación de cualquier petición que se presente.

Por ejemplo, como la capa de modelo devuelve un conjunto de datos, la vista los usaría para hacer una página HTML que los contenga. O un resultado con formato XML para que otras aplicaciones puedan consumir.

La capa de la Vista no se limita únicamente a HTML o texto que represente los datos, sino que puede ser utilizada para ofrecer una amplia variedad de formatos en función de sus necesidades tales como videos, música, documentos y cualquier otro formato que puedas imaginar.

La capa del Controlador

La capa del controlador gestiona las peticiones de los usuarios. Es responsable de responder la información solicitada con la ayuda tanto del modelo como de la vista.

Los controladores pueden ser vistos como administradores cuidando de que todos los recursos necesarios para completar una tarea se deleguen a los trabajadores más adecuados. Espera peticiones de los clientes, comprueba su validez de acuerdo a las normas de autenticación o autorización, delega la búsqueda de datos al modelo y selecciona el tipo de respuesta más adecuado según las preferencias del cliente. Finalmente delega este proceso de presentación a la capa de la Vista.

El ciclo de una petición en CakePHP

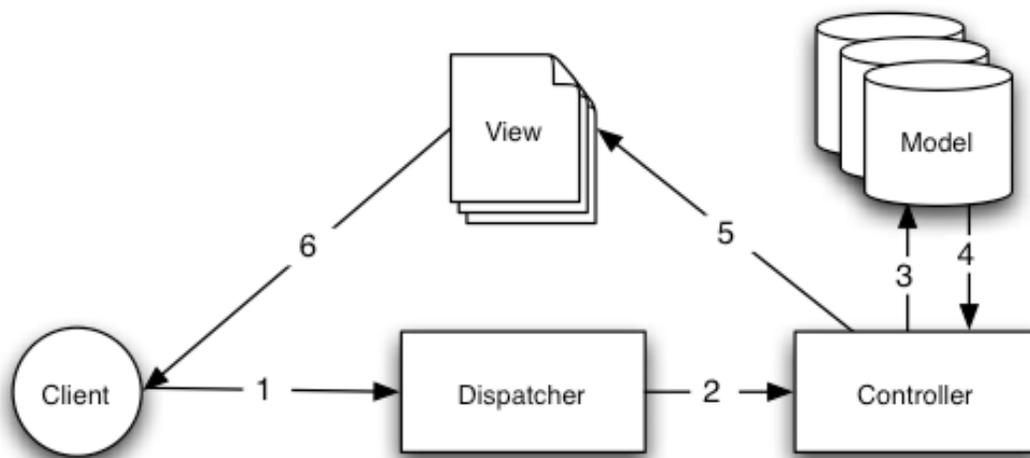


Figure: 1: Una petición MVC típica

Figure: 1 muestra el manejo de una petición típica a una aplicación CakePHP.

El ciclo de una petición típica en CakePHP comienza cuando un usuario solicita una página o un recurso de tu aplicación. Esta solicitud es procesada por un despachador que selecciona el controlador correcto para manejarlo.

Una vez que la solicitud llega al controlador, éste se comunicará con la capa del Modelo para cualquier proceso de captación de datos o el guardado de los mismos según se requiera. Una vez finalizada esta comunicación el controlador procederá a delegar en el objeto de vista correcto la tarea de generar una presentación resultante de los datos proporcionada por el modelo.

Finalmente, cuando esta presentación se genera, se envía de inmediato al usuario.

Casi todas las solicitudes para la aplicación van a seguir este patrón básico. Vamos a añadir algunos detalles más adelante que son específicos a CakePHP, así que mantén esto en mente a medida que avancemos.

Beneficios

¿Por qué utilizar MVC? Debido a que es un patrón de diseño de software verdaderamente probado que convierte una aplicación en un paquete modular fácil de mantener y mejora la rapidez del desarrollo. La separación de las tareas de tu aplicación en modelos, vistas y controladores hace que su aplicación sea además muy ligeras de entender. Las nuevas características se añaden fácilmente y agregar cosas nuevas a código viejo se hace muy sencillo. El diseño modular también permite a los desarrolladores y los diseñadores trabajar simultáneamente, incluyendo la capacidad de hacer prototipos rápidos²³.

La separación también permite a los desarrolladores hacer cambios en una parte de la aplicación sin afectar a los demás.

Si nunca has creado una aplicación de esta forma se necesita algún tiempo para acostumbrarse, pero estamos seguros que una vez que hayas terminado tu primera aplicación con CakePHP no vas a querer hacerlo de cualquier otra manera.

Para iniciarte con tu primera aplicación en CakePHP *haz este tutorial ahora*

Dónde encontrar ayuda

EL sitio oficial de CakePHP

<https://cakephp.org>

EL sitio oficial de CakePHP siempre es un buen lugar para visitar. Contiene enlaces a herramientas comunmente usadas, videos, oportunidades para donar al proyecto y descargas útiles.

El manual

<https://book.cakephp.org>

Este manual debe ser probablemente el primer lugar al que acudir para obtener respuestas. Al igual que con muchos otros proyectos de código abierto, tenemos gente nueva con regularidad. Haz tu mejor esfuerzo para responder tus propias preguntas por cuenta propia en primer lugar, esto te ayudará a entender los conceptos más rápidamente y a nosotros a mejorar la documentación.

The Bakery

<https://bakery.cakephp.org>

The Bakery Es el sitio oficial para publicar todo lo relacionado a CakePHP, desde tutoriales, nuevos plugins, actualizaciones a CakePHP hasta casos de estudio de usuarios del framework.

²³ https://en.wikipedia.org/wiki/Software_prototyping

El API

<https://api.cakephp.org/2.x/>

Directo al grano y directamente de los desarrolladores principales, el API (Application Programming Interface) de CakePHP es el más una amplia documentación en torno a todos los detalles del funcionamiento interno del framework. Si quieres los detalles de cada función, los parámetros, y ver cómo las clases se complementan entre sí, este es el lugar para buscar.

Las pruebas unitarias

Si alguna vez sientes la información proporcionada en la API no es suficiente, echa un vistazo al código de las pruebas unitarias de CakePHP. Pueden servir como ejemplos prácticos para la utilización y los datos parametros de cada clase.:

```
lib/Cake/Test/Case
```

El canal IRC

Canales IRC oficiales en irc.freenode.net:

- [#cakephp](#) – Discusión general en Inglés
- [#cakephp-es](#) – Discusión general en Español

Si no tienes ni idea, nos peagas un grito en el canal de IRC de CakePHP. Alguien del equipo de desarrollo está allí generalmente, sobre todo durante las horas del día de los usuarios del norte y América del Sur. Nos encantaría saber de ti, si necesitas ayuda, quieres encontrar los usuarios en tu área, o si deseas donarnos un nuevo coche.

El grupo de Google

En Español: <https://groups.google.com/group/cakephp-esp>

CakePHP tiene también un Grupo de Google muy activo. Puede ser un gran recurso para encontrar las respuestas archivadas, preguntas frecuentes, y obtener respuestas a los problemas inmediatos.

Where to get Help in your Language

French

- [French CakePHP Community](#)²⁴

²⁴ <https://cakephp-fr.org>

Controllers

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)²⁵ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Controller Methods

Request and Response objects

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)²⁶ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

²⁵ <https://github.com/cakephp/docs>

²⁶ <https://github.com/cakephp/docs>

Sending files

Scaffolding

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)²⁷ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

The Pages Controller

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)²⁸ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Componentes

Los componentes son paquetes de logica que es compartida entre controladores. CakePHP incluye un fantastico set de componentes que puedes usar para ayudarte en tareas comunes. También puedes crear tus propios componentes. Si te encontrás queriendo copiar y pegar cosas entre de un controlador a otro, deberias considerar crear tu propio componente para obtener esa funcionalidad. Crear componentes ayuda a mantener el codigo de tus controladores limpio y facilita la reutilización de código entre proyectos.

Cada uno de los componentes principales esta detallado en su propio capítulo. Ver *Components*. Esta sección describe como configurar y usar los componentes y como crear tus propios componentes.

Configurando componentes

Mucho de los componentes principales requieren configuración. Algunos ejemplos de componentes que requieren configuración son *Authentication* y *Cookie*. La configuración para estos componentes, y para los componentes en general, es realizada usualmente en el array `$components` o en el método `beforeFilter()` de tú controlador:

```
class PostsController extends AppController {
    public $components = array(
        'Auth' => array(
            'authorize' => array('controller'),
            'loginAction' => array(
                'controller' => 'users',
                'action' => 'login'
            )
        )
    )
}
```

(continué en la próxima página)

²⁷ <https://github.com/cakephp/docs>

²⁸ <https://github.com/cakephp/docs>

(proviene de la página anterior)

```

    ),
    'Cookie' => array('name' => 'CookieMonster')
);

```

El fragmento de código anterior sería un ejemplo de configuración de un componente con el array `$components`. Todos los componentes principales permiten que su configuración sea seteada de ésta forma. A su vez, es posible configurar el componente a través del método `beforeFilter()` de tú controlador. Esto es útil cuando necesitas asignar los resultados de una función a la propiedad de un componente. El código anterior también podría ser expresado como:

```

public function beforeFilter() {
    $this->Auth->authorize = array('controller');
    $this->Auth->loginAction = array(
        'controller' => 'users',
        'action' => 'login'
    );

    $this->Cookie->name = 'CookieMonster';
}

```

Sin embargo, es posible que un componente requiera que cierta configuración antes que el método `beforeFilter()` del controlador sea ejecutado. Con éste fin, algunos componentes permiten que las opciones de configuración sean seteadas en el array `$components`:

```

public $components = array(
    'DebugKit.Toolbar' => array('panels' => array('history', 'session'))
);

```

Consulte la documentación relevante para determinar que opción de configuración provee cada componente.

Una opción común es el uso de `className`, la cual le permite ponerle un alias a un componente. Esta característica es útil cuando se quiere reemplazar `$this->Auth` o la referencia de otro componente común con una implementación propia:

```

// app/Controller/PostsController.php
class PostsController extends AppController {
    public $components = array(
        'Auth' => array(
            'className' => 'MyAuth'
        )
    );
}

// app/Controller/Component/MyAuthComponent.php
App::uses('AuthComponent', 'Controller/Component');
class MyAuthComponent extends AuthComponent {
    // Add your code to override the core AuthComponent
}

```

El código anterior aplicaría un *alias* `MyAuthComponent` a `$this->Auth` en tus controladores.

Nota: Usar un alias en un componente reemplaza esa instancia en cualquier lugar que ese componente sea usado, incluyendo dentro de otros componentes.

Usando Componentes

Una vez que has incluido algunos componentes en tus controladores, usarlos es bastante sencillo. Cada componente que use es expuesto como una propiedad en tu controlador. Si ha cargado *SessionComponent* y *CookieComponent* en su controlador, accedería a ellos de esta forma:

```
class PostsController extends AppController {
    public $components = array('Session', 'Cookie');

    public function delete() {
        if ($this->Post->delete($this->request->data('Post.id'))) {
            $this->Session->setFlash('Post deleted. ');
            return $this->redirect(array('action' => 'index'));
        }
    }
}
```

Nota: Ya que tanto modelos como componentes son agregados a los controladores como propiedades, compartan el mismo “namespace”. Asegúrese de no darles el mismo nombre a un modelo y componente.

Cargar modelos en el acto

Tal vez no necesite todos los componentes disponibles en cada acción de su controlador. En situaciones como esta puede cargar un componente en tiempo de ejecución usando *Component Collection*. Desde dentro de un método de un controlador se puede:

```
$this->OneTimer = $this->Components->load('OneTimer');
$this->OneTimer->getTime();
```

Nota: Tenga en cuenta que cargar un componente en tiempo de ejecución no llamara el método initialize de el componente.

Callbacks de componentes

Los componentes también ofrecen varios callbacks que permiten aumentar el ciclo del request. Vea *API Componentes* para más información sobre los callbacks que ofrecen los componentes.

Creando componentes

Suponga que nuestra aplicación online necesita llevar a cabo una operación matemática compleja en diferentes partes de la aplicación. En este caso, crearíamos un componente que albergue esta lógica compartida para usar en diferentes controladores.

El primer paso es crear un nuevo archivo y clase de componente. Cree el archivo en `app/Controller/Component/MathComponent.php`. La estructura básica para un componente es:

```
App::uses('Component', 'Controller');
class MathComponent extends Component {
    public function doComplexOperation($amount1, $amount2) {
```

(continué en la próxima página)

(proviene de la página anterior)

```

        return $amount1 + $amount2;
    }
}

```

Nota: Todos los componentes deben extender *Component*. Si no lo hacen se disparará una excepción.

Incluyendo tus componentes en un controlador

Una vez que nuestro componente está terminado, podemos usarlo en un controlador agregando el nombre del componente (sin la parte de «Component») en el array `$components`. El controlador recibirá automáticamente un nuevo atributo llamado como este componente a través del cual podemos acceder a una instancia del componente:

```

/* Hacer disponible el nuevo componente $this->Math,
y el standard $this->Session */
public $components = array('Math', 'Session');

```

Los componentes declarados en `AppController` serán combinados con aquellos en otros controladores. Por lo que no hay necesidad de re-declarar un componente mas de una vez.

Al incluir componentes en un controlador también podemos declarar un grupo de parametros que serán pasados al constructor del componente. Estos parametros pueden ser manejados por el componente:

```

public $components = array(
    'Math' => array(
        'precision' => 2,
        'randomGenerator' => 'srand'
    ),
    'Session', 'Auth'
);

```

El código anterior pasará un array que contiene `precision` y `randomGenerator` a `MathComponent::__construct()` como el segundo parámetro. Por convención si los keys de un array son iguales a las propiedades públicas de un componente, las propiedades serán actualizadas con los valores de esas keys.

Usando otros componentes en tus componentes

A veces, algunos de tus componentes pueden necesitar usar otros componentes. En estos casos se puede incluir otros componentes en tus componentes de la misma forma que se incluyen en un controlador:

```

// app/Controller/Component/CustomComponent.php
App::uses('Component', 'Controller');
class CustomComponent extends Component {
    // los otros componentes que tu componente usa
    public $components = array('Existing');

    public function initialize(Controller $controller) {
        $this->Existing->foo();
    }
}

```

(continué en la próxima página)

(proviene de la página anterior)

```

    public function bar() {
        // ...
    }
}

// app/Controller/Component/ExistingComponent.php
App::uses('Component', 'Controller');
class ExistingComponent extends Component {

    public function foo() {
        // ...
    }
}

```

Nota: En contraste con los componentes incluidos en un controlador ningún callback va a ser disparado en el componente de un componente.

API Componentes

class Component

La clase base Component ofrece varios métodos para cargar otros componentes a través de ComponentCollection así como para tratar con configuraciones comunes. También provee prototipos para todos los callbacks de los componentes.

Component::__construct(*ComponentCollection \$collection*, *\$settings = array()*)

Constructor para la clase base component. Todos los *\$settings* que también tengan propiedades públicas serán actualizados para coincidir con *\$settings*.

Callbacks

Component::initialize(*Controller \$controller*)

Es llamado antes que el método `beforeFilter` de un controlador.

Component::startup(*Controller \$controller*)

Es llamado después de el método `beforeFilter` pero antes que el controlador ejecute la acción.

Component::beforeRender(*Controller \$controller*)

Es llamada luego que el controlador ejecute la acción de la vista solicitada, pero antes que el controlador renderice la vista.

Component::shutdown(*Controller \$controller*)

Es llamado antes de que el output sea enviado al navegador.

Component::beforeRedirect(*Controller \$controller*, *\$url*, *\$status=null*, *\$exit=true*)

Es llamado cuando el método de redirección del controlador es llamado, pero antes de acción alguna. Si este método devuelve `false` el controlador no continuará hacia el redireccionamiento.

Views

Las Views (Vistas) son la **V** en MVC. Las vistas son responsables de generar la salida requerida por la solicitud. Comúnmente es en un formato de HTML, XML o JSON, pero transmitir archivos o crear archivos PDF también son responsabilidades de la capa de las vistas.

CakePHP incluye clases de vistas predefinidas para encargarse de los escenarios más comunes de renderizado:

- Para crear servicios web de XML o JSON, puedes usar las *JSON and XML views*.
- Para retornar archivos protegidos o generado dinámicamente, puedes usar *Sending files*.
- Para crear vistas con múltiples temas, puedes usar *Themes*.

Plantillas de vistas

La capa de vistas de CakePHP es cómo le hablas a tus usuarios. La mayoría del tiempo las vistas mostrarán documentos de (X)HTML a los exploradores, pero pueda que también tengas que proveerle data un objeto de Flash, responderle a una aplicación remote a través de SOAP o desplegar un archivo CSV para un usuario.

Por defecto los archivos de las vistas de CakePHP están escritas en PHP puro and tienen una extensión por defecto `.ctp` (CakePHP Template). Estos archivos contienen toda la lógica de la presentación necesaria para transformar los datos recibidos del controlador a un formato que esté listo para la audiencia destinada. Si prefieres usar un lenguaje para hacer plantillas, como Twig o Smarty, una subclase de View creará un puente entre tu lenguaje y CakePHP.

Un archivo de vista está almacenado en `/app/View/`, en un subdirectorio cuyo nombre está basado en el nombre del controlador que use el archivo. Tiene un nombre que corresponde a su acción. Por ejemplo, la vista para la acción «view()» del controlador de productos se encontraría normalmente en `/app/View/Products/view.ctp`.

La capa de vista en CakePHP puede estar formada de un número de diferentes partes. Cada parte tiene distintos usos y será explicada en este capítulo:

- **views:** Las vistas son la parte de la página que es única para la acción ejecutándose. Forman la base de la respuesta de la aplicación.

- **elements:** piezas más pequeñas y reutilizables de código. Los elementos son renderizados típicamente en las vistas.
- **layouts:** archivos de vistas que contienen code de presentación que envuelven muchas interfaces de la aplicación. La mayoría de vistas son renderizadas dentro de un layout.
- **helpers:** estas clases encapsulan la logica de las vistas que es necesaria en muchos lugares de la capa de vistas. Los helpers en CakePHP puede ayudar a construir formularios, funcionalidad de AJAX, paginar datos de los modelos, o proveer transmisiones de RSS, entre otros.

Extender vistas

Nuevo en la versión 2.1.

Extender vistas te permite envolver una vista dentro de otra. Combinando esto con *view blocks* te da una manera poderosa de mantener tus vistas *DRY*. Por ejemplo, tu aplicación tiene una barra lateral que necesita cambiar dependiendo de la vista renderizada. Al extender un archivo de vista común, puedes evitar que se repita el markup común de la barra lateral y sólo definir las partes que cambien:

```
// app/View/Common/view.ctp
<h1><?php echo $this->fetch('title'); ?></h1>
<?php echo $this->fetch('content'); ?>

<div class="actions">
  <h3>Related actions</h3>
  <ul>
    <?php echo $this->fetch('sidebar'); ?>
  </ul>
</div>
```

La vista de arriba puede usarse como una vista padre. Espera que la vista que la extienda defina los bloques `sidebar` y `title`. El bloque de `content` es un bloque especial que CakePHP crea. Esta contiene todo el contenido no capturado por la vista extendida. Asumiendo que nuestra vista tiene una variable `$post` con datos de nuestra publicación, la vista podría verse así:

```
<?php
// app/View/Posts/view.ctp
$this->extend('/Common/view');

$this->assign('title', $post);

$this->start('sidebar');
?>
<li>
<?php
echo $this->Html->link('edit', array(
  'action' => 'edit',
  $post['Post']['id']
)); ?>
</li>
<?php $this->end(); ?>
// El resto del contenido estará disponible como el bloque de
// contenido en la vista padre.
<?php echo h($post['Post']['body']);
```

La publicación de arriba muestra como puedes extender una vista e ingresar datos de bloques. Cualquier contenido que no esté definido en un bloque será capturado y puesto en el bloque especial llamado `content`. Cuando una vista contiene una llamada a `extend()`, la ejecución continúa hasta el final del archivo. Una vez terminada la vista extendida será renderizada. Llamar `extend()` más de una vez en una vista se antepone a la siguiente vista padre que será procesada:

```
$this->extend('/Common/view');
$this->extend('/Common/index');
```

Lo de arriba resultará en `/Common/index.ctp` siendo renderizada como la vista padre de la vista actual.

Puedes anidar vistas extendidas cuantas veces sea necesario. Cada vista puede extender otra vista si es deseado. Cada vista padre tendrá el contenido de la vista anterior como el bloque `content`.

Nota: Evita usar `content` como un nombre de bloque en tu aplicación. CakePHP usa este nombre para el contenido no capturado en vistas extendidas.

Usar bloques de vistas

Nuevo en la versión 2.1.

Los bloques de vista reemplazan a `$scripts_for_layout` y proveen un API flexible que permite definir los puestos o bloques en tus vistas/layouts que serán definidos en otro lugar. Por ejemplo, los bloques son ideales para implementar cosas como barras laterales o regiones que carguen recursos en la parte superior o inferior del layout. Los bloques pueden ser definidos de dos formas: usando captura de bloques o asignación directa. Los métodos `start()`, `append()` and `end` permiten trabajar capturando bloques:

```
// crear el bloque de la barra lateral
$this->start('sidebar');
echo $this->element('sidebar/recent_topics');
echo $this->element('sidebar/recent_comments');
$this->end();

// Adjuntar la barra más tarde
$this->append('sidebar');
echo $this->element('sidebar/popular_topics');
$this->end();
```

También puedes adjuntar un bloque usando `start()` multiples veces. `assign()` también puede ser usado para quitar un bloque o sobrescribirlo en cualquier momento:

```
// Quita el contenido previo del bloque de la barra lateral.
$this->assign('sidebar', '');
```

En la versión 2.3 unos métodos nuevos fueron añadidos para trabajar con bloques. El método `prepend()` fue añadido para anteponerle contenido a un bloque existente:

```
// Anteponer a la barra lateral
$this->prepend('sidebar', 'this content goes on top of sidebar');
```

El método `startIfEmpty()` puede ser usado para empezar un bloque **sólo** si este está vacío o no definido. Si el bloque ya existe, el contenido capturado será descartado. Esto es útil cuando se quiere definir condicionalmente contenido predeterminado para un bloque si no existe todavía:

```
// En un archivo de vista.  
// Crear un bloque navbar  
$this->startIfEmpty('navbar');  
echo $this->element('navbar');  
echo $this->element('notifications');  
$this->end();
```

```
// En una vista/layout padre  
<?php $this->startIfEmpty('navbar'); ?>  
<p>If the block is not defined by now - show this instead</p>  
<?php $this->end(); ?>  
  
// Después en la vista/layout padre  
echo $this->fetch('navbar');
```

En el ejemplo de arriba, el bloque navbar sólo tendrá contenido si es agregado después de la primera sección. Ya que el bloque está definido en la vista hija, el contenido predeterminado con `<p>` será descartado.

Nuevo en la versión 2.3: `startIfEmpty()` y `prepend()` fueron añadidos en la versión 2.3

Nota: Evita usar `content` como un nombre de bloque en tu aplicación. CakePHP usa este nombre para el contenido no capturado en vistas extendidas.

Desplegar bloques

Nuevo en la versión 2.1.

Los bloques pueden ser desplegados con el método `fetch()`. `fetch()` desplegará de forma segura, retornando "" si el bloque no existe:

```
echo $this->fetch('sidebar');
```

`fetch()` también puede ser usado para desplegar condicionalmente contenido que debe envolver un bloque si este existe. Esto es útil en layouts o vistas extendidas donde quieras mostrar condicionalmente encabezados u otro markup:

```
// en app/View/Layouts/default.ctp  
<?php if ($this->fetch('menu')): ?>  
<div class="menu">  
  <h3>Menu options</h3>  
  <?php echo $this->fetch('menu'); ?>  
</div>  
<?php endif; ?>
```

A partir de la versión 2.3.0 también se puede proveer un valor predeterminado para un bloque si este no tiene contenido. Esto permite añadir contenido predeterminado a ser reemplazado para estados vacíos. Puedes proveer un valor predeterminado usando el segundo argumento:

```
<div class="shopping-cart">  
  <h3>Your Cart</h3>  
  <?php echo $this->fetch('cart', 'Your cart is empty'); ?>  
</div>
```

Distinto en la versión 2.3: El argumento `$default` fue añadido en la versión 2.3.

Usar bloques para archivos de CSS o scripts

Nuevo en la versión 2.1.

Los bloques reemplazan la variable desvalorizada `$scripts_for_layout`. Es recomendado usar bloques en su lugar. El *HtmlHelper* se enlaza con los bloques de vista y sus métodos `script()`, `css()` y `meta()` actualizan a un bloque con el mismo nombre si se usan con la opción `inline = false`:

```
<?php
// en tu archivo de vista
$this->Html->script('carousel', array('inline' => false));
$this->Html->css('carousel', array('inline' => false));
?>

// En tu archivo de layout
<!DOCTYPE html>
<html lang="en">
  <head>
    <title><?php echo $this->fetch('title'); ?></title>
    <?php echo $this->fetch('script'); ?>
    <?php echo $this->fetch('css'); ?>
  </head>
  // el resto del layout continúa
```

HtmlHelper también permite controlar a que bloque van los scripts y CSS:

```
// en tu vista
$this->Html->script('carousel', array('block' => 'scriptBottom'));

// en tu layout
echo $this->fetch('scriptBottom');
```

Layouts

Un layout contiene el código de presentación que envuelve a la vista. Todo lo que quieras ver en una vista debe ser puesto en un layout.

El layout predeterminado de CakePHP está localizado en `/app/View/Layouts/default.ctp`. Si quieres cambiar el aspecto general de la aplicación, este es el lugar indicado para comenzar, ya que el código de la vista renderizada por el controlador está colocado dentro del layout predeterminado cuando la página es renderizada.

Otros archivos de layout deberán ser puestos en `app/View/Layouts`. Cuando creas un layout, necesitas decirle a CakePHP en dónde se deben desplegar tus vistas. Para hacer esto, asegurate que tu layout incluya en algún lugar `$this->fetch('content')`. Un ejemplo de cómo se podría ver un layout predeterminado:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title><?php echo $this->fetch('title'); ?></title>
    <link rel="shortcut icon" href="favicon.ico" type="image/x-icon">
    <!-- Include external files and scripts here (See HTML helper for more info.) -->
    <?php
echo $this->fetch('meta');
```

(continué en la próxima página)

(proviene de la página anterior)

```

echo $this->fetch('css');
echo $this->fetch('script');
?>
</head>
<body>

<!-- Si deseas que menús aparezcan
en todas las vistas, incluyelos aquí -->
<div id="header">
    <div id="menu">...</div>
</div>

<!-- Aquí es donde quieres tus vistas desplegadas -->
<?php echo $this->fetch('content'); ?>

<!-- Añade un pie de página a todas las páginas -->
<div id="footer">...</div>

</body>
</html>

```

Nota: Antes de la versión 2.1, el método `fetch()` no estaba disponible, `fetch('content')` es un reemplazo de `$content_for_layout` y las líneas `fetch('meta')`, `fetch('css')` y `fetch('script')` están contenidas en la variable `$scripts_for_layout` en la versión 2.0

Los bloques `script`, `css` y `meta` contienen todo el contenido definido en las vistas usando un helper de HTML incluido. Útil para incluir archivos de JavaScript y CSS de las vistas.

Nota: Cuando se usen `HtmlHelper::css()` o `HtmlHelper::script()` in archivos de vistas, especifica “false” en la opción “inline” para colocar la fuente de HTML en un bloque con el mismo nombre. (Ve el API para más detalles de uso).

El bloque `content` contiene los contenidos de la vista renderizada.

`$title_for_layout` contiene el título de la página. Esta variable es generada automáticamente, pero puede ser reasignar como cualquier otra variable en tu controlador o vista.

Nota: La variable `$title_for_layout` estará depreciada a partir de la versión 2.5, usa `$this->fetch('title')` en tu layout y `$this->assign('title', 'page title')` en su lugar.

Puedes crear cuantos layouts quieras: sólo colócalas en el directorio `app/View/Layouts` y alterna entre ellas dentro de tus acciones de controlador usando `$layout` property:

```

// desde un controlador
public function admin_view() {
    // stuff
    $this->layout = 'admin';
}

```

(continué en la próxima página)

(proviene de la página anterior)

```
// desde un archivo de vista
$this->layout = 'loggedin';
```

Por ejemplo, si una sección de un sitio incluye un espacio pequeño para publicidad, se podría un nuevo layout con el pequeño espacio publicitario y aplicarlo para todas las acciones del controlador usando algo como esto:

```
class UsersController extends AppController {
    public function view_active() {
        $this->set('title_for_layout', 'View Active Users');
        $this->layout = 'default_small_ad';
    }

    public function view_image() {
        $this->layout = 'image';
        //despliegue de imagen de usuario
    }
}
```

CakePHP tiene dos layouts base (sin contar el layout predeterminado de CakePHP) que puedes usar en tu propia aplicación: “ajax” y “flash”. El layout Ajax es útil para generar respuestas AJAX, es un layout vacío. (La mayoría de llamadas AJAX sólo requieren un poco de markup en su retorno, en lugar de una interfaz completa.) El layout de flash es usado por mensajes mostrados por el método `Controller::flash()`.

Otros tres layouts existen (xml, js y rss) existen en la base para una forma rápida y fácil de generar contenido que no es texto o html.

Usar layouts desde los plugins

Nuevo en la versión 2.1.

Si quieres usar un layout que existe en un plugin, puedes usar *plugin syntax*. Por ejemplo, para usar el layout de contacto desde el plugin Contacts:

```
class UsersController extends AppController {
    public function view_active() {
        $this->layout = 'Contacts.contact';
    }
}
```

Elementos

Muchas aplicaciones tienen pequeños bloques de código de presentación que necesitan repetirse en varias páginas, a veces en distintos lugares en el layout. CakePHP puede ayudarte a repetir las partes de tu sitio web que necesitan reutilizarse. Estas partes reutilizables son llamadas Elementos (Elements). Espacios publicitarios, de ayuda, controles de navegación, menus adicionales, formularios de inicio de sesión son generalmente implementados como elementos en CakePHP. Un elemento es una vista miniatura que puede ser incluida en otras vistas, en layouts o incluso otros elementos. Los elementos pueden ser usados para hacer una vista más legible, colocando la renderización de los elementos que se repiten en su propio archivo. También pueden ayudarte a reutilizar fragmentos de contenido en tu aplicación.

Los elementos viven en el directorio `/app/View/Elements`, y tienen la extensión `.ctp`. Son desplegados utilizando el método de elementos de la vista:

```
echo $this->element('helpbox');
```

Pasar variables a un elemento

Puedes pasar datos a un elemento a través del segundo argumento del elemento:

```
echo $this->element('helpbox', array(
    "helptext" => "Oh, this text is very helpful."
));
```

Dentro del archivo del elemento estarán todas variables pasadas como miembros del arreglo de parametros (de la misma forma que `:php:meth`Controller::set()`` funciona en el controlador con los archivos de vistas). En el ejemplo anterior el archivo `/app/View/Elements/helpbox.ctp` puede usar la variable `$helptext` variable:

```
// dentro de app/View/Elements/helpbox.ctp
echo $helptext; //despliega "Oh, this text is very helpful."
```

El método `View::element()` también soporta opciones para el elemento. Las opciones suportadas son “cache” y “callbacks”. Un ejemplo:

```
echo $this->element('helpbox', array(
    "helptext" => "This is passed to the element as $helptext",
    "foobar" => "This is passed to the element as $foobar",
),
array(
    // usa la configuración de cache "long_view"
    "cache" => "long_view",
    // puesto en true para hacer llamadas a before/afterRender del elemento
    "callbacks" => true
)
);
```

El almacenamiento de los elementos es facilitado por la clase `Cache`. Se puede configurar que los elementos se almacenen en cualquier `Cache` que se haya configurado. Esto te da una gran flexibilidad para decidir dónde y por cuánto tiempo se almacenan los elementos. Para almacenar distintas versiones del mismo elemento en una aplicación debes proveer una llave única de cache usando el siguiente formato:

```
$this->element('helpbox', array(), array(
    "cache" => array('config' => 'short', 'key' => 'valor único')
)
);
```

Puedes tomar completa ventaja de los elementos al usar `requestAction()`, el cual obtiene las variables de la vista desde una acción de controlador y las retorna como un arreglo. Esto le permite a tus elementos operar en la forma pura del estilo MVC. Crea una acción de controlador que prepare las variables de vista para tus elementos, luego llama `requestAction()` dentro del segundo parametro de `element()` para alimentar al elemento las variables de vista desde tu controlador.

Para hacer esto, en tu controlador, agrega algo similar a lo siguiente para el ejemplo de `Post`:

```
class PostsController extends AppController {
    // ...
    public function index() {
```

(continué en la próxima página)

(proviene de la página anterior)

```

    $posts = $this->paginate();
    if ($this->request->is('requested')) {
        return $posts;
    }
    $this->set('posts', $posts);
}
}

```

Y después en el elemento podemos acceder al modelo de publicaciones paginado. Para obtener las últimas 5 publicaciones en una lista ordenada tendríamos que hacer algo similar a lo siguiente:

```

<h2>Latest Posts</h2>
<?php
    $posts = $this->requestAction(
        'posts/index/sort:created/direction:asc/limit:5'
    );
?>
<ol>
<?php foreach ($posts as $post): ?>
    <li><?php echo $post['Post']['title']; ?></li>
<?php endforeach; ?>
</ol>

```

Almacenar elementos

Puedes tomar ventaja de el almacenamiento de vistas de CakePHP si provees un parametro de cache. Si se le asigna true almacenará el elemento en el cache “default” (predeterminado) de la configuración. De lo contrario puedes configurar cuál cache deberá usarse en la configuración. Ver [Caching](#) para más información de cómo configurar la clase Cache. Un ejemplo simple de almacenamiento un elemento sería así:

```
echo $this->element('helpbox', array(), array('cache' => true));
```

Si renderizas un mismo elemento más de una vez en una vista y tienes activado el almacenamiento, asegurate de asignarle a “key” un valor diferente cada vez. Esto prevendrá que cada llamada sobrescriba el resultado anterior de las llamadas anteriores de element(). Por ejemplo:

```

echo $this->element(
    'helpbox',
    array('var' => $var),
    array('cache' => array('key' => 'primer_uso', 'config' => 'view_long'))
);

echo $this->element(
    'helpbox',
    array('var' => $varDiferente),
    array('cache' => array('key' => 'segundo_uso', 'config' => 'view_long'))
);

```

Lo de arriba asegurará que los elementos se almacenen por separado. Si quieres que todo almacenamiento use la misma configuración, puedes evitar un poco de repetición usando `View::$elementCache` a la configuración de cache que quieres usar. CakePHP usará esta configuración cuando no se provea una.

Solicitar elementos de un plugin

2.0

Para cargar un elemento de un plugin, usa la opción de *plugin* (movida fuera la opción de *data* en la versión 1.x):

```
echo $this->element('helpbox', array(), array('plugin' => 'Contacts'));
```

2.1

Si estás usando algún plugin y deseas usar elementos dentro del plugin simplemente utiliza *plugin syntax*. Si la vista está siendo renderizada por un controlador o acción de plugin, el nombre del plugin será antepuesto automáticamente a todos los elementos usados, a menos que otro nombre de un plugin esté presente. Si el elemento no existe en el plugin, buscará en el directorio principal APP:

```
echo $this->element('Contacts.helpbox');
```

Si tu vista es parte de un plugin, puedes omitir el nombre del plugin. Por ejemplo, si estás en `ContactsController` del plugin de `Contacts`:

```
echo $this->element('helpbox');
// y
echo $this->element('Contacts.helpbox');
```

son equivalentes y resultarán en que el mismo elemento cargue.

Distinto en la versión 2.1: La opción `$options[plugin]` fue depreciada y soporte para `Plugin.element` fue añadida.

Crear tus propias clases de vistas

Puede que necesites crear clases personalizadas para activar nuevos tipos de datos o agregar lógica de renderización adicional a tu aplicación. Como la mayoría de componentes de CakePHP, las clases de vista tienen un par de estándares:

- Archivos de clases de vistas deben estar en `App/View`. Por ejemplo: `App/View/PdfView.php`
- Las clases deben tener el sufijo `View`. Por ejemplo: `PdfView`.
- Al hacer referencia a una vista, se debe omitir el sufijo `View`. Por ejemplo: `$this->viewClass = 'Pdf'`;

También deberás extender `View` para asegurar que las cosas funcionen bien:

```
// en App/View/PdfView.php

App::uses('View', 'View');
class PdfView extends View {
    public function render($view = null, $layout = null) {
        // lógica personalizada aquí.
    }
}
```

Reemplazar el método de renderizado te deja control total sobre cómo se renderiza el contenido.

API de Vistas

class View

Los métodos de vistas son accesibles en todos los archivos de vistas, elementos y layouts. Para llamar a cualquier método usa `$this->method()`

View::set(*string \$var, mixed \$value*)

Las vistas tienen un método `set()` que es análogo al `set()` de los objetos «Controller». Usar `set()` desde el archivo de vista agregará variables al layout y elementos que serán renderizados después. Ver *Controller Methods* para más información de cómo usar `set()`.

En tu archivo de vista puedes hacer:

```
$this->set('activeMenuButton', 'posts');
```

Luego, en tu layout, la variable `$activeMenuButton` estará disponible y contendrá el valor “posts”.

View::get(*string \$var, \$default = null*)

Obtiene el valor de la `viewVar` con el nombre `$var`.

A partir de la versión 2.5 se puede proveer un valor predeterminando en caso de que la variable no tenga un valor asignado.

Distinto en la versión 2.5: El argumento `$default` fue agregado en la versión 2.5.

View::getVar(*string \$var*)

Obtiene el valor de la `viewVar` con el nombre `$var`.

Obsoleto desde la versión 2.3: Usa `View::get()` en su lugar.

View::getVars()

Obtiene una lista de todas las variables de vista en el alcance actual de renderizado. Retorna un arreglo de nombres de variables.

View::element(*string \$elementPath, array \$data, array \$options = array()*)

Renderiza un elemento o vista parcial. Ver la sección de *Elementos* para más información y ejemplos.

View::uuid(*string \$object, mixed \$url*)

Genera un ID de DOM no aleatorio para un objeto basándose en el tipo de objeto y URL. Este método es regularmente usado por helpers que necesitan generar IDs de DOM para elementos como la clase `:php:class`JsHelper``:

```
$uuid = $this->uuid(
    'form',
    array('controller' => 'posts', 'action' => 'index')
);
//$uuid contiene 'form0425fe3bad'
```

View::addScript(*string \$name, string \$content*)

Agrega contenido al búfer interno de scripts. Este búfer está disponible en el layout como `$scripts_for_layout`. Este método es útil cuando se crean helpers que necesiten agregarle javascript or CSS directamente al layout. No olvides que los scripts agregados desde el layout y elementos en el layout no serán añadidos a `$scripts_for_layout`. Este método es usado generalmente desde helpers, como *JsHelper* y *HtmlHelper*.

Obsoleto desde la versión 2.1: Usar *Usar bloques de vistas* en su lugar.

View::blocks()

Coloca los nombres de todos los bloques definidos en un arreglo.

View::start(\$name)

Empezar la captura de un bloque para un bloque de vista. Ver la sección de *Usar bloques de vistas* para más detalles y ejemplos.

Nuevo en la versión 2.1.

View::end()

Terminar el bloque más recientemente abierto. Ver la sección de *Usar bloques de vistas* para más detalles y ejemplos.

Nuevo en la versión 2.1.

View::append(\$name, \$content)

Añadir al final de bloques con \$name. Ver la sección de *Usar bloques de vistas* para más detalles y ejemplos.

Nuevo en la versión 2.1.

View::prepend(\$name, \$content)

Anteponer a bloques con \$name. Ver la sección de *Usar bloques de vistas* para más detalles y ejemplos.

Nuevo en la versión 2.3.

View::startIfEmpty(\$name)

Empezar un bloque si está vacío. Todo el contenido del bloque será capturado y descartado si el bloque ya estaba definido.

Nuevo en la versión 2.3.

View::assign(\$name, \$content)

Asignar el valor a un bloque. Esto sobrescribirá cualquier contenido existente. Ver la sección de *Usar bloques de vistas* para más detalles y ejemplos.

Nuevo en la versión 2.1.

View::fetch(\$name, \$default = "")

Obtener el valor de un bloque. Si un bloque está vacío o no definido, se retornará "". Ver la sección de *Usar bloques de vistas* para más detalles y ejemplos.

Nuevo en la versión 2.1.

View::extend(\$name)

Extender la vista, elemento o layout actual con la vista con este nombre. Ver la sección de *Extender vistas* para más detalles y ejemplos.

Nuevo en la versión 2.1.

property View::\$layout

Asignar el layout que envolverá a la vista actual.

property View::\$elementCache

La configuración que se usa para almacenar elementos. Darle valor a esta propiedad cambiará el comportamiento predeterminado usado para almacenar elementos. Lo predeterminado puede ser modificado usando la opción "cache" en el método del elemento.

property View::\$request

Una instancia de CakeRequest. Usa esta instancia para acceder información acerca de la solicitud (request) actual.

property View::\$output

Contiene el último contenido renderizado de una vista, ya sea el archivo de vista o el contenido del layout.

Obsoleto desde la versión 2.1: Usa `$view->Blocks->get('content')`; en su lugar.

property View::\$Blocks

Una instancia de `ViewBlock`. Usado para proporcionar la funcionalidad de bloque de vista en el renderizado de vista.

Nuevo en la versión 2.1.

Más acerca de vistas

Themes

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)²⁹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Media Views

class MediaView

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)³⁰ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

JSON and XML views

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)³¹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

²⁹ <https://github.com/cakephp/docs>

³⁰ <https://github.com/cakephp/docs>

³¹ <https://github.com/cakephp/docs>

Helpers

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)³² o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

³² <https://github.com/cakephp/docs>

Models

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)³³ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Associations: Linking Models Together

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)³⁴ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

³³ <https://github.com/cakephp/docs>

³⁴ <https://github.com/cakephp/docs>

Retrieving Your Data

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)³⁵ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Model::query()

Saving Your Data

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)³⁶ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Deleting Data

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)³⁷ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Data Validation

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)³⁸ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

³⁵ <https://github.com/cakephp/docs>

³⁶ <https://github.com/cakephp/docs>

³⁷ <https://github.com/cakephp/docs>

³⁸ <https://github.com/cakephp/docs>

Validating Data from the Controller

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)³⁹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Callback Methods

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁴⁰ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Behaviors

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁴¹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

DataSources

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁴² o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

³⁹ <https://github.com/cakephp/docs>

⁴⁰ <https://github.com/cakephp/docs>

⁴¹ <https://github.com/cakephp/docs>

⁴² <https://github.com/cakephp/docs>

Model Attributes

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁴³ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Additional Methods and Properties

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁴⁴ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Virtual fields

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁴⁵ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Transactions

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁴⁶ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

⁴³ <https://github.com/cakephp/docs>

⁴⁴ <https://github.com/cakephp/docs>

⁴⁵ <https://github.com/cakephp/docs>

⁴⁶ <https://github.com/cakephp/docs>

Core Libraries

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁴⁷ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

CakePHP comes with a plethora of built-in functions and classes. These classes and functions try to cover some of the most common features required in web applications.

General Purpose

General purpose libraries are available and reused in many places across CakePHP.

General Purpose

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁴⁸ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

⁴⁷ <https://github.com/cakephp/docs>

⁴⁸ <https://github.com/cakephp/docs>

Global Constants and Functions

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁴⁹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

App Class

```
class App
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁵⁰ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Events System

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁵¹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Collections

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁵² o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

⁴⁹ <https://github.com/cakephp/docs>

⁵⁰ <https://github.com/cakephp/docs>

⁵¹ <https://github.com/cakephp/docs>

⁵² <https://github.com/cakephp/docs>

Behaviors

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁵³ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Behaviors add extra functionality to your models. CakePHP comes with a number of built-in behaviors such as [TreeBehavior](#) and [ContainableBehavior](#).

To learn about creating and using behaviors, read the section on [Behaviors](#).

Behaviors

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁵⁴ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Behaviors add extra functionality to your models. CakePHP comes with a number of built-in behaviors such as [TreeBehavior](#) and [ContainableBehavior](#).

To learn about creating and using behaviors, read the section on [Behaviors](#).

ACL

`class AclBehavior`

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁵⁵ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

⁵³ <https://github.com/cakephp/docs>

⁵⁴ <https://github.com/cakephp/docs>

⁵⁵ <https://github.com/cakephp/docs>

Containable

class ContainableBehavior

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁵⁶ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Translate

class TranslateBehavior

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁵⁷ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Tree

class TreeBehavior

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁵⁸ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Components

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁵⁹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

⁵⁶ <https://github.com/cakephp/docs>

⁵⁷ <https://github.com/cakephp/docs>

⁵⁸ <https://github.com/cakephp/docs>

⁵⁹ <https://github.com/cakephp/docs>

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

CakePHP has a selection of components to help take care of basic tasks in your controllers. See the section on *Components* for how to configure and use components.

Components

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁶⁰ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

CakePHP has a selection of components to help take care of basic tasks in your controllers. See the section on *Components* for how to configure and use components.

Pagination

```
class PaginatorComponent(ComponentCollection $collection, array $settings = array())
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁶¹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Flash

```
class FlashComponent(ComponentCollection $collection, array $config = array())
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁶² o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

⁶⁰ <https://github.com/cakephp/docs>

⁶¹ <https://github.com/cakephp/docs>

⁶² <https://github.com/cakephp/docs>

Sessions

```
class SessionComponent(ComponentCollection $collection, array $settings = array())
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁶³ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Authentication

```
class AuthComponent(ComponentCollection $collection, array $settings = array())
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁶⁴ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Security

```
class SecurityComponent(ComponentCollection $collection, array $settings = array())
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁶⁵ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Request Handling

```
class RequestHandlerComponent(ComponentCollection $collection, array $settings = array())
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁶⁶ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

⁶³ <https://github.com/cakephp/docs>

⁶⁴ <https://github.com/cakephp/docs>

⁶⁵ <https://github.com/cakephp/docs>

⁶⁶ <https://github.com/cakephp/docs>

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Cookie

```
class CookieComponent(ComponentCollection $collection, array $settings = array())
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁶⁷ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Access Control Lists

```
class AclComponent(ComponentCollection $collection, array $settings = array())
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁶⁸ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Helpers

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁶⁹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

CakePHP features a number of helpers that aid in view creation. They assist in creating well-formed markup (including forms), aid in formatting text, times and numbers, and can even integrate with popular JavaScript libraries. Here is a summary of the built-in helpers.

Read [Helpers](#) to learn more about helpers, their API, and how you can create and use your own helpers.

⁶⁷ <https://github.com/cakephp/docs>

⁶⁸ <https://github.com/cakephp/docs>

⁶⁹ <https://github.com/cakephp/docs>

Helpers

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁷⁰ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

CakePHP features a number of helpers that aid in view creation. They assist in creating well-formed markup (including forms), aid in formatting text, times and numbers, and can even integrate with popular JavaScript libraries. Here is a summary of the built-in helpers.

Read [Helpers](#) to learn more about helpers, their API, and how you can create and use your own helpers.

CacheHelper

```
class CacheHelper(View $view, array $settings = array())
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁷¹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Flash

```
class FlashHelper(View $view, array $config = array())
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁷² o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

⁷⁰ <https://github.com/cakephp/docs>

⁷¹ <https://github.com/cakephp/docs>

⁷² <https://github.com/cakephp/docs>

FormHelper

```
class FormHelper(View $view, array $settings = array())
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁷³ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

HtmlHelper

```
class HtmlHelper(View $view, array $settings = array())
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁷⁴ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

JsHelper

```
class JsHelper(View $view, array $settings = array())
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁷⁵ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

NumberHelper

```
class NumberHelper(View $view, array $settings = array())
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁷⁶ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

⁷³ <https://github.com/cakephp/docs>

⁷⁴ <https://github.com/cakephp/docs>

⁷⁵ <https://github.com/cakephp/docs>

⁷⁶ <https://github.com/cakephp/docs>

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Paginator

```
class PaginatorHelper(View $view, array $settings = array())
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁷⁷ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

RSS

```
class RssHelper(View $view, array $settings = array())
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁷⁸ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

SessionHelper

```
class SessionHelper(View $view, array $settings = array())
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁷⁹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

⁷⁷ <https://github.com/cakephp/docs>

⁷⁸ <https://github.com/cakephp/docs>

⁷⁹ <https://github.com/cakephp/docs>

TextHelper

```
class TextHelper(View $view, array $settings = array())
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁸⁰ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁸¹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

TimeHelper

```
class TimeHelper(View $view, array $settings = array())
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁸² o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁸³ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

⁸⁰ <https://github.com/cakephp/docs>

⁸¹ <https://github.com/cakephp/docs>

⁸² <https://github.com/cakephp/docs>

⁸³ <https://github.com/cakephp/docs>

Utilities

Beyond the core MVC components, CakePHP includes a great selection of utility classes that help you do everything from webservice requests, to caching, to logging, internationalization and more.

Utilities

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁸⁴ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Caching

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁸⁵ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

CakeEmail

```
class CakeEmail(mixed $config = null)
```

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁸⁶ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

⁸⁴ <https://github.com/cakephp/docs>

⁸⁵ <https://github.com/cakephp/docs>

⁸⁶ <https://github.com/cakephp/docs>

Folder & File

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁸⁷ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Hash

class Hash

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁸⁸ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

HttpSocket

class HttpSocket(*mixed \$config = array()*)

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁸⁹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Inflector

class Inflector

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁹⁰ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

⁸⁷ <https://github.com/cakephp/docs>

⁸⁸ <https://github.com/cakephp/docs>

⁸⁹ <https://github.com/cakephp/docs>

⁹⁰ <https://github.com/cakephp/docs>

Internationalization & Localization

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁹¹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Logging

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁹² o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

CakeNumber

`class CakeNumber`

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁹³ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Router

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁹⁴ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

⁹¹ <https://github.com/cakephp/docs>

⁹² <https://github.com/cakephp/docs>

⁹³ <https://github.com/cakephp/docs>

⁹⁴ <https://github.com/cakephp/docs>

Data Sanitization

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁹⁵ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Security

class Security

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁹⁶ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Set

class Set

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁹⁷ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

String

class String

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁹⁸ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

⁹⁵ <https://github.com/cakephp/docs>

⁹⁶ <https://github.com/cakephp/docs>

⁹⁷ <https://github.com/cakephp/docs>

⁹⁸ <https://github.com/cakephp/docs>

CakeTime

class CakeTime

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)⁹⁹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Xml

class Xml

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁰⁰ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

⁹⁹ <https://github.com/cakephp/docs>

¹⁰⁰ <https://github.com/cakephp/docs>

Plugins

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁰¹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

How To Install Plugins

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁰² o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

¹⁰¹ <https://github.com/cakephp/docs>

¹⁰² <https://github.com/cakephp/docs>

How To Use Plugins

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁰³ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

How To Create Plugins

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁰⁴ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

¹⁰³ <https://github.com/cakephp/docs>

¹⁰⁴ <https://github.com/cakephp/docs>

Shells, Tasks & Console Tools

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁰⁵ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Shell Helpers

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁰⁶ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

¹⁰⁵ <https://github.com/cakephp/docs>

¹⁰⁶ <https://github.com/cakephp/docs>

Running Shells as cronjobs

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁰⁷ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Completion Shell

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁰⁸ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Code Generation with Bake

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁰⁹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Schema management and migrations

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹¹⁰ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

¹⁰⁷ <https://github.com/cakephp/docs>

¹⁰⁸ <https://github.com/cakephp/docs>

¹⁰⁹ <https://github.com/cakephp/docs>

¹¹⁰ <https://github.com/cakephp/docs>

I18N shell

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹¹¹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

ACL Shell

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹¹² o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Test shell

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹¹³ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Upgrade shell

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹¹⁴ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

¹¹¹ <https://github.com/cakephp/docs>

¹¹² <https://github.com/cakephp/docs>

¹¹³ <https://github.com/cakephp/docs>

¹¹⁴ <https://github.com/cakephp/docs>

Development

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹¹⁵ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

In this section we'll cover the various aspects of developing a CakePHP application. Topics like Configuration, handling errors & exceptions, debugging, and testing will be covered.

Configuration

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹¹⁶ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

¹¹⁵ <https://github.com/cakephp/docs>

¹¹⁶ <https://github.com/cakephp/docs>

Routing

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹¹⁷ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Routes Configuration

Sessions

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹¹⁸ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Exceptions

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹¹⁹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Error Handling

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹²⁰ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

¹¹⁷ <https://github.com/cakephp/docs>

¹¹⁸ <https://github.com/cakephp/docs>

¹¹⁹ <https://github.com/cakephp/docs>

¹²⁰ <https://github.com/cakephp/docs>

Debugging

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹²¹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Testing

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹²² o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

REST

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹²³ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Dispatcher Filters

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹²⁴ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

¹²¹ <https://github.com/cakephp/docs>

¹²² <https://github.com/cakephp/docs>

¹²³ <https://github.com/cakephp/docs>

¹²⁴ <https://github.com/cakephp/docs>

ES - Deployment

Steps to deploy on a Hosting Server

Tutoriales y Ejemplos

En esta sección puedes encontrar varias aplicaciones completas construidas en CakePHP que te ayudarán a comprender el framework y ver cómo se relacionan todas las piezas.

También puedes ver otros ejemplos en: [CakePackages](#)¹²⁵ y en [Bakery](#)¹²⁶ encontrarás también componentes listos para usar.

Parte 1: Tutorial para desarrollar el Blog

Bienvenido a CakePHP. Probablemente estás consultando este tutorial porque quieres aprender cómo funciona CakePHP. Nuestro objetivo es potenciar tu productividad y hacer más divertido el desarrollo de aplicaciones. Esperamos que puedas comprobarlo a medida que vas profundizando en el código.

En este tutorial vamos a crear un blog sencillo desde cero. Empezaremos descargando e instalando CakePHP, luego crearemos una base de datos y el código necesario para listar, añadir, editar o borrar artículos del blog.

Esto es lo que necesitas:

1. Servidor web funcionando. Asumiremos que estás usando Apache, aunque las instrucciones para otros servidores son similares. Igual tendremos que ajustar un poco la configuración inicial, pero todos los pasos son sencillos. La mayor parte de nosotros podrá tener CakePHP funcionando sin tocar nada en su configuración.
2. Base de datos funcionando. Usaremos MySQL en este tutorial. Necesitarás saber cómo crear una base de datos nueva. CakePHP se encargará del resto.
3. Nivel básico de PHP. Si estás familiarizado con la programación orientada a objetos, mucho mejor. Aún así puedes seguir desarrollando con tu estilo procedimental si lo prefieres.
4. Conocimiento sobre patrón MVC. Puedes encontrar una definición rápida aquí: *Entendiendo el Modelo - Vista - Controlador*. No tengas miedo, sólo es media página.

¹²⁵ <https://plugins.cakephp.org/>

¹²⁶ <https://bakery.cakephp.org/>

¡ Vamos allá !

Descargar CakePHP

Vamos a descargar la última versión de CakePHP.

Para ello, visita la web del proyecto en GitHub: <https://github.com/cakephp/cakephp/tags> y descargar / descomprimir la última versión de la rama 2.0

También puedes clonar el repositorio usando `git`¹²⁷. `git clone git://github.com/cakephp/cakephp.git`

Usa el método que prefieras y coloca la carpeta que has descargado bajo la ruta de tu servidor web (dentro de tu DocumentRoot). Una vez terminado, tu directorio debería tener esta estructura:

```
/path_to_document_root
/app
/lib
/plugins
/vendors
.htaccess
index.php
README
```

Es buen momento para aprender algo sobre cómo funciona esta estructura de directorios: echa un vistazo a «Directorios en CakePHP», Sección: *Estructura de directorios de CakePHP*.

Creando la base de datos para nuestro blog

Vamos a crear una nueva base de datos para el blog. Puedes crear una base de datos en blanco con el nombre que quieras. De momento vamos a definir sólo una tabla para nuestros artículos («posts»). Además crearemos algunos artículos de test para usarlos luego. Una vez creada la tabla, ejecuta el siguiente código SQL en ella:

```
/* tabla para nuestros articulos */
CREATE TABLE posts (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(50),
  body TEXT,
  created DATETIME DEFAULT NULL,
  modified DATETIME DEFAULT NULL
);

/* algunos valores de test */
INSERT INTO posts (title,body,created)
  VALUES ('The title', 'This is the post body.', NOW());
INSERT INTO posts (title,body,created)
  VALUES ('A title once again', 'And the post body follows.', NOW());
INSERT INTO posts (title,body,created)
  VALUES ('Title strikes back', 'This is really exciting! Not.', NOW());
```

La elección de los nombres para el nombre de la tabla y de algunas columnas no se ha hecho al azar. Si sigues las convenciones para nombres en la Base de Datos, y las demás convenciones en tus clases (ver más sobre convenciones aquí: *Convenciones en CakePHP*), aprovecharás la potencia del framework y ahorrarás mucho trabajo de configuración.

¹²⁷ <https://git-scm.com/>

CakePHP es flexible, si no quieres usar las convenciones puedes configurar luego cada elemento para que funcione con tu Base de Datos legada. Te recomendamos que utilices estas convenciones ya que te ahorrarán tiempo.

Al llamar “posts” a nuestra tabla de artículos, estamos diciendo a CakePHP que vincule esta tabla por defecto al Modelo “Post”, e incluir los campos “modified” y “created” con ese nombre, serán automáticamente administrados por CakePHP.

Configurando la Base de Datos

Rápido y sencillo, vamos a decirle a CakePHP dónde está la Base de Datos y cómo conectarnos a ella. Probablemente ésta será la primera y última vez que lo hagas en cada proyecto.

Hay un fichero de configuración preparado para que sólo tengas que copiarlo y modificarlo con tu propia configuración.

Cambia el nombre del fichero `/app/Config/database.php.default` por `/app/Config/database.php` (hemos eliminado el “.default” del final).

Edita ahora este fichero y verás un array definido en la variable `$default` que contiene varios campos. Modifica esos campos para que se correspondan con tu configuración actual de acceso a la Base de Datos. Debería quedarte algo similar a esto:

```
public $default = array(
    'datasource' => 'Database/Mysql',
    'persistent' => false,
    'host' => 'localhost',
    'port' => '',
    'login' => 'cakeBlog',
    'password' => 'c4k3-rU13Z',
    'database' => 'cake_blog_tutorial',
    'schema' => '',
    'prefix' => '',
    'encoding' => ''
);
```

Ten en cuenta que los campos “login”, “password”, “database” tendrás que cambiarlos por tu usuario de MySQL, tu contraseña de MySQL y el nombre que le diste a la Base de Datos.

Guarda este fichero.

Ahora ya podrás acceder a la página inicial de bienvenida de CakePHP en tu máquina. Esta página podrás accederla normalmente en <http://localhost/cakeblog> si has llamado a la carpeta raíz del proyecto “cakeblog”. Verás una página de bienvenida que muestra varias informaciones de configuración y te indica si tienes correctamente instalado CakePHP.

Configuración Opcional

Hay otras tres cosas que puedes querer configurar, aunque no son requeridas para este tutorial no está mal echarles un vistazo. Para ello abre el fichero `/app/Config/core.php` que contiene todos estos parámetros.

1. Configurar un string de seguridad “salt” para usarlo al realizar los “hash”.
2. Configurar un número semilla para el encriptado “seed”.
3. Definir permisos de escritura en la carpeta `Tmp`. El servidor web (normalmente “apache”) debe poder escribir dentro de esta carpeta y subcarpetas.

El string de seguridad se utiliza en la generación de “hashes”. Cambia el valor inicial y escribe cualquier cosa diferente. Cualquier cosa vale. Para cambiarlo vete a la línea 203 del fichero `/app/Config/core.php` y verás algo así:

```
/**
 * A random string used in security hashing methods.
 */
Configure::write('Security.salt', 'pl345e-P45s_7h3*S@l7!');
```

El número semilla se utiliza para encriptar y desencriptar cadenas. Cambia el valor por defecto en el fichero `/app/Config/core.php` línea 208. No importa qué número pongas, que sea difícil de adivinar.

```
/**
 * A random numeric string (digits only) used to encrypt/decrypt strings.
 */
Configure::write('Security.cipherSeed', '7485712659625147843639846751');
```

Para dar permisos al directorio `app/Tmp`, la mejor forma es ver qué usuario está ejecutando el servidor web (`<?php echo `whoami` ; ?>`) y cambiar el directorio para que el nuevo propietario sea el usuario que ejecuta el servidor web.

En un sistema `*nix` esto se hace así:

```
$ chown -R www-data app/tmp
```

Suponiendo que `www-data` sea el usuario que ejecuta tu servidor web (en otras versiones de `*unix` como `fedora`, el usuario suele llamarse “`apache`”).

Si `CakePHP` no puede escribir en este directorio, te informará de ello en la página de bienvenida, siempre que tengas activado el modo depuración, por defecto está activo.

Sobre `mod_rewrite`

Si eres nuevo usuario de `Apache`, puedes encontrar alguna dificultad con `mod_rewrite`, así que lo trataremos aquí.

Si al cargar la página de bienvenida de `CakePHP` ves cosas raras (no se cargan las imágenes ni los estilos y se ve todo en blanco y negro), esto significa que probablemente la configuración necesita ser revisada en el servidor `Apache`. Prueba lo siguiente:

1. Asegúrate de que existe la configuración para procesar los ficheros `.htaccess`. En el fichero de configuración de `Apache`: “`httpd.conf`” debería existir una sección para cada “`Directory`” de tu servidor. Asegúrate de que `AllowOverride` está fijado a `All` para el directorio que contiene tu aplicación web. Para tu seguridad, es mejor que no asignes `All` a tu directorio raíz `<Directory />` sino que busques el bloque `<Directory>` que se refiera al directorio en el que tienes instalada tu aplicación web.
2. Asegúrate que estás editando el fichero `httpd.conf` correcto, ya que en algunos sistemas hay ficheros de este tipo por usuario o por aplicación web. Consulta la documentación de `Apache` para tu sistema.
3. Comprueba que existen los ficheros `.htaccess` en el directorio en el que está instalada tu aplicación web. A veces al descomprimir el archivo o al copiarlo desde otra ubicación, estos ficheros no se copian correctamente. Si no están ahí, obtén otra copia de `CakePHP` desde el servidor oficial de descargas.
4. Asegúrate de tener activado el módulo `mod_rewrite` en la configuración de `Apache`. Deberías tener algo así:

```
LoadModule rewrite_module      libexec/httpd/mod_rewrite.so

(para Apache 1.3)::

AddModule      mod_rewrite.c

en tu fichero httpd.conf
```

Si no puedes (o no quieres) configurar `mod_rewrite` o algún otro módulo compatible, necesitarás activar las url amigables en CakePHP. En el fichero `/app/Config/core.php`, quita el comentario a la línea:

```
Configure::write('App.baseUrl', env('SCRIPT_NAME'));
```

Borra también los ficheros `.htaccess` que ya no serán necesarios:

```
/.htaccess
/app/.htaccess
/app/webroot/.htaccess
```

Esto hará que tus url sean así: `www.example.com/index.php/nombredelcontrolador/nombredelaaccion/parametro` en vez de `www.example.com/nombredelcontrolador/nombredelaaccion/parametro`.

Si estás instalando CakePHP en otro servidor diferente a Apache, encontrarás instrucciones para que funcione la reescritura de URLs en la sección *Instalación Avanzada*

Parte 2: Tutorial para desarrollar el Blog

Creando un modelo para los artículos (*Post Model*)

Los modelos son una parte fundamental en CakePHP. Cuando creamos un modelo, podemos interactuar con la base de datos para crear, editar, ver y borrar con facilidad cada ítem de ese modelo.

Los ficheros en los que se definen los modelos se ubican en la carpeta `/app/Model`, y el fichero que vamos a crear debe guardarse en la ruta `/app/Model/Post.php`. El contenido de este fichero será:

```
class Post extends AppModel {
    public $name = 'Post';
}
```

Los convenios usados para los nombres son importantes. Cuando llamamos a nuestro modelo *Post*, CakePHP deducirá automáticamente que este modelo se utilizará en el controlador `PostsController`, y que se vinculará a una tabla en nuestra base de datos llamada `posts`.

Nota: CakePHP creará dinámicamente un objeto para el modelo si no encuentra el fichero correspondiente en `/app/Model`. Esto significa que si te equivocas al nombrar el fichero (por ejemplo lo llamas `post.php` con la primera p minúscula o `posts.php` en plural) CakePHP no va a reconocer la configuración que escribas en ese fichero y utilizará valores por defecto.

Para más información sobre modelos, como prefijos para las tablas, validación, etc. puedes visitar *Models* en el Manual.

Crear un Controlador para nuestros Artículos (*Posts*)

Vamos a crear ahora un controlador para nuestros artículos. En el controlador es donde escribiremos el código para interactuar con nuestros artículos. Es donde se utilizan los modelos para llevar a cabo el trabajo que queramos hacer con nuestros artículos. Vamos a crear un nuevo fichero llamado `PostsController.php` dentro de la ruta `/app/Controller`. El contenido de este fichero será:

```
class PostsController extends AppController {
    public $helpers = array('Html', 'Form');
}
```

Y vamos a añadir una acción a nuestro nuevo controlador. Las acciones representan una función concreta o interfaz en nuestra aplicación. Por ejemplo, cuando los usuarios recuperan la url `www.example.com/posts/index` (que CakePHP también asigna por defecto a la ruta `www.example.com/posts/` ya que la acción por defecto de cada controlador es `index` por convención) esperan ver un listado de *posts*. El código para tal acción sería este:

```
class PostsController extends AppController {
    public $helpers = array ('Html', 'Form');

    function index() {
        $this->set('posts', $this->Post->find('all'));
    }
}
```

Si examinamos el contenido de la función `index()` en detalle, podemos ver que ahora los usuarios podrán acceder a la ruta `www.example.com/posts/index`. Además si creáramos otra función llamada `foobar()`, los usuarios podrían acceder a ella en la url `www.example.com/posts/foobar`.

Advertencia: Puede que tengas la tentación de llamar tus controladores y acciones de forma determinada para que esto afecte a la ruta final, y así puedas predeterminar estas rutas. No te preocupes por esto ya que CakePHP incorpora un potente sistema de configuración de rutas. Al escribir los ficheros, te recomendamos seguir las convenciones de nombres y ser claro. Luego podrás generar las rutas que te convengan utilizando el componente de rutas (*Route*).

La función `index` tiene sólo una instrucción `set()` que sirve para pasar información desde el controlador a la vista (*view*) asociada. Luego crearemos esta vista. Esta función `set()` asigna una nueva variable “posts” igual al valor retornado por la función `find('all')` del modelo `Post`. Nuestro modelo `Post` está disponible automáticamente en el controlador y no hay que importarlo ya que hemos usado las convenciones de nombres de CakePHP.

Para aprender más sobre los controladores, puedes visitar el capítulo [Controllers](#)

Creando una vista para los artículos (*View*)

Ya tenemos un modelo que define nuestros artículos y un controlador que ejecuta alguna lógica sobre ese modelo y envía los datos recuperados a la vista. Ahora vamos a crear una vista para la acción `index()`.

Las vistas en CakePHP están orientadas a cómo se van a presentar los datos. Las vistas encajan dentro de *layouts* o plantillas. Normalmente las vistas son una mezcla de HTML y PHP, aunque pueden ser también XML, CSV o incluso datos binarios.

Las plantillas (*layouts*) sirven para recubrir las vistas y reutilizar código. Además pueden crearse tantos layouts como se deseen y se puede elegir cuál utilizar en cada momento. Por el momento vamos a usar el la plantilla por defecto `default`.

¿ Recuerdas que el controlador envió a la vista una variable `posts` que contiene todos los posts mediante el método `set()` ? Esto nos generará una variable en la vista con esta pinta:

```
// print_r($posts) output:
```

```
Array
(
    [0] => Array
        (
            [Post] => Array
                (
```

(continué en la próxima página)

(proviene de la página anterior)

```

        [id] => 1
        [title] => The title
        [body] => This is the post body.
        [created] => 2008-02-13 18:34:55
        [modified] =>
    )
)
[1] => Array
(
    [Post] => Array
        (
            [id] => 2
            [title] => A title once again
            [body] => And the post body follows.
            [created] => 2008-02-13 18:34:56
            [modified] =>
        )
)
[2] => Array
(
    [Post] => Array
        (
            [id] => 3
            [title] => Title strikes back
            [body] => This is really exciting! Not.
            [created] => 2008-02-13 18:34:57
            [modified] =>
        )
)
)

```

Las vistas en CakePHP se almacenan en la ruta `/app/View` y en un directorio con el mismo nombre que el controlador al que pertenecen, en nuestro caso `Posts`, así que para mostrar estos elementos formateados mediante una tabla tendremos algo como esto:

```

<!-- File: /app/View/Posts/index.ctp -->

<h1>Blog posts</h1>
<table>
  <tr>
    <th>Id</th>
    <th>Title</th>
    <th>Created</th>
  </tr>

  <!-- Here is where we loop through our $posts array, printing out post info -->

  <?php foreach ($posts as $post): ?>
  <tr>
    <td><?php echo $post['Post']['id']; ?></td>
    <td>
      <?php echo $this->Html->link($post['Post']['title'],

```

(continué en la próxima página)

(proviene de la página anterior)

```

array('controller' => 'posts', 'action' => 'view', $post['Post']['id'])); ?>
    </td>
    <td><?php echo $post['Post']['created']; ?></td>
</tr>
<?php endforeach; ?>
</table>

```

Esto debería ser sencillo de comprender.

Como habrás notado, hay una llamada a un objeto `$this->Html`. Este objeto es una instancia de una clase *Helper HtmlHelper*. CakePHP proporciona un conjunto de *Helpers* para ayudarte a completar acciones habituales, como por ejemplo realizar un link, crear un formulario, utilizar Javascript y Ajax de forma sencilla, etc. Puedes aprender más sobre esto en *Helpers* en otro momento. Basta con saber que la función `link()` generará un link HTML con el título como primer parámetro y la URL como segundo parámetro.

Cuando crees URLs en CakePHP te recomendamos emplear el formato de array. Se explica con detenimiento en la sección de *Routes*. Si utilizas estas rutas, podrás aprovecharte de las potentes funcionalidades de generación inversa de rutas de CakePHP en el futuro. Además puedes especificar rutas relativas a la base de tu aplicación de la forma “/controlador/accion/param1/param2”.

Llegados a este punto, deberías poder ver esta página si escribes la ruta a tu aplicación en el navegador, normalmente será algo así `http://localhost/blog/posts/index`. Deberías ver los posts correctamente formateados en una tabla.

Verás que si pinchas sobre alguno de los enlaces que aparecen en esta página (que van a una URL “/posts/view/some_id”, verás una página de error que te indica que la acción `view()` no ha sido definida todavía, y que debes definirla en el fichero `PostsController`. Si no ves ese error, algo ha ido mal, ya que esa acción no está definida y debería mostrar la página de error correspondiente. Cosa muy rara. Creemos esta acción para evitar el error:

```

class PostsController extends AppController {
    public $helpers = array('Html', 'Form');
    public $name = 'Posts';

    public function index() {
        $this->set('posts', $this->Post->find('all'));
    }

    public function view($id = null) {
        $this->set('post', $this->Post->findById($id));
    }
}

```

Si observas la función `view()`, ahora el método `set()` debería serte familiar. Verás que estamos usando `findById()` en vez de `find('all')` ya que sólo queremos un post concreto.

Verás que nuestra función `view` toma un parámetro (`$id`), que es el ID del artículo que queremos ver. Este parámetro se gestiona automáticamente al llamar a la URL `/posts/view/3`, el valor “3” se pasa a la función `view` como primer parámetro `$id`.

Vamos a definir la vista para esta nueva función, como hicimos antes para `index()` salvo que el nombre ahora será `/app/View/Posts/view.ctp`.

```

<!-- File: /app/View/Posts/view.ctp -->

<h1><?php echo $post['Post']['title']; ?></h1>

```

(continué en la próxima página)

(proviene de la página anterior)

```
<p><small>Created: <?php echo $post['Post']['created']?></small></p>
<p><?php echo $post['Post']['body']?></p>
```

Verifica que ahora funciona el enlace que antes daba un error desde `/posts/index` o puedes ir manualmente si escribes `/posts/view/1`.

Añadiendo artículos (*posts*)

Ya podemos leer de la base de datos nuestros artículos y mostrarlos en pantalla, ahora vamos a ser capaces de crear nuevos artículos y guardarlos.

Lo primero, añadir una nueva acción `add()` en nuestro controlador `PostsController`:

```
class PostsController extends AppController {
    public $helpers = array('Html', 'Form', 'Flash');
    public $components = array('Flash');

    public function index() {
        $this->set('posts', $this->Post->find('all'));
    }

    public function view($id) {
        $this->set('post', $this->Post->findById($id));
    }

    public function add() {
        if ($this->request->is('post')) {
            if ($this->Post->save($this->request->data)) {
                $this->Flash->success('Your post has been saved.');
```

Nota: Necesitas incluir el `FlashComponent` y `FlashHelper` en el controlador para poder utilizarlo. Si lo prefieres, puedes añadirlo en `AppController` y será compartido para todos los controladores que hereden de él.

Lo que la función `add()` hace es: si el formulario enviado no está vacío, intenta guardar un nuevo artículo utilizando el modelo `Post`. Si no se guarda bien, muestra la vista correspondiente, así podremos mostrar los errores de validación si el artículo no se ha guardado correctamente.

Cuando un usuario utiliza un formulario y efectúa un POST a la aplicación, esta información puedes accederla en `$this->request->data`. Puedes usar la función `pr()` o `debug()` para mostrar el contenido de esa variable y ver la pinta que tiene.

Utilizamos el `FlashComponent`, concretamente el método `FlashComponent::success()` para guardar el mensaje en la sesión y poder recuperarlo posteriormente en la vista y mostrarlo al usuario, incluso después de haber redirigido a otra página mediante el método `redirect()`. Esto se realiza a través de la función `FlashHelper::render()` que está en el layout, que muestra el mensaje y lo borra de la sesión para que sólo se vea una vez. El método

`Controller::redirect` del controlador nos permite redirigir a otra página de nuestra aplicación, traduciendo el parámetro `array('action' => 'index')` a la URL `/posts`, y la acción `index`. Puedes consultar la documentación de este método aquí `Router::url()`. Verás los diferentes modos de indicar la ruta que quieres construir.

Al llamar al método `save()`, comprobará si hay errores de validación primero y si encuentra alguno, no continuará con el proceso de guardado. Veremos a continuación cómo trabajar con estos errores de validación.

Validando los datos

CakePHP te ayuda a evitar la monotonía al construir tus formularios y su validación. Todos odiamos teclear largos formularios y gastar más tiempo en reglas de validación de cada campo. CakePHP está aquí para echarnos una mano.

Para aprovechar estas funciones es conveniente que utilices el `FormHelper` en tus vistas. La clase `FormHelper` está disponible en tus vistas por defecto mediante llamadas del estilo `$this->Form`.

Nuestra vista sería así

```
<!-- File: /app/View/Posts/add.ctp -->

<h1>Add Post</h1>
<?php
echo $this->Form->create('Post');
echo $this->Form->input('title');
echo $this->Form->input('body', array('rows' => '3'));
echo $this->Form->end('Save Post');
?>
```

Hemos usado `FormHelper` para generar la etiqueta “form”. Esta llamada al `FormHelper` : `$this->Form->create()` generaría el siguiente código

```
<form id="PostAddForm" method="post" action="/posts/add">
```

Si `create()` no tiene parámetros al ser llamado, asume que estás creando un formulario que realiza el `submit` al método del controlador `add()` o al método `edit()` si hay un `id` en los datos del formulario. Por defecto el formulario se enviará por POST.

Las llamadas `$this->Form->input()` se usan para crear los elementos del formulario con el nombre que se pasa por parámetro. El primer parámetro indica precisamente el nombre del campo del modelo para el que se quiere crear el elemento de entrada. El segundo parámetro te permite definir muchas otras variables sobre la forma en la que se generará este *input field*. Por ejemplo, al enviar `array('rows' => '3')` estamos indicando el número de filas para el campo *textarea* que vamos a generar. El método `input()` está dotado de introspección y un poco de magia, ya que tiene en cuenta el tipo de datos del modelo al generar cada campo.

Una vez creados los campos de entrada para nuestro modelo, la llamada `$this->Form->end()` genera un botón de *submit* en el formulario y cierra el tag `<form>`. Puedes ver todos los detalles aquí [Helpers](#).

Volvamos atrás un minuto para añadir un enlace en `/app/View/Post/index.ctp` que nos permita agregar nuevos artículos. Justo antes del tag `<table>` añade la siguiente línea:

```
echo $this->Html->link('Add Post', array('controller' => 'posts', 'action' => 'add'));
```

Te estarás preguntando: ¿Cómo le digo a CakePHP la forma en la que debe validar estos datos? Muy sencillo, las reglas de validación se escriben en el modelo. Abre el modelo `Post` y vamos a escribir allí algunas reglas sencillas

```
class Post extends AppModel {
    public $name = 'Post';
```

(continué en la próxima página)

(proviene de la página anterior)

```

public $validate = array(
    'title' => array(
        'rule' => 'notEmpty'
    ),
    'body' => array(
        'rule' => 'notEmpty'
    )
);
}

```

El array `$validate` contiene las reglas definidas para validar cada campo, cada vez que se llama al método `save()`. En este caso vemos que la regla para ambos campos es que no pueden ser vacíos `notEmpty`. El conjunto de reglas de validación de CakePHP es muy potente y variado. Podrás validar direcciones de email, codificación de tarjetas de crédito, incluso añadir tus propias reglas de validación personalizadas. Para más información sobre esto [Data Validation](#).

Ahora que ya tienes las reglas de validación definidas, usa tu aplicación para crear un nuevo artículo con un título vacío y verás cómo funcionan. Como hemos usado el método `FormHelper::input()`, los mensajes de error se construyen automáticamente en la vista sin código adicional.

Editando Posts

Seguro que ya le vas cogiendo el truco a esto. El método es siempre el mismo: primero la acción en el controlador, luego la vista. Aquí está el método `edit()`:

```

public function edit($id = null) {
    if (!$id) {
        throw new NotFoundException(__('Invalid post'));
    }

    $post = $this->Post->findById($id);
    if (!$post) {
        throw new NotFoundException(__('Invalid post'));
    }

    if ($this->request->is(array('post', 'put'))) {
        $this->Post->id = $id;
        if ($this->Post->save($this->request->data)) {
            $this->Flash->success(__('Your post has been updated.'));
            return $this->redirect(array('action' => 'index'));
        }
        $this->Flash->error(__('Unable to update your post.'));
    }

    if (!$this->request->data) {
        $this->request->data = $post;
    }
}

```

Esta acción primero comprueba que se trata de un GET request. Si lo es, buscamos un `Post` con el id proporcionado como parámetro y lo ponemos a disposición para usarlo en la vista. Si la llamada no es GET, usaremos los datos que se envíen por POST para intentar actualizar nuestro artículo. Si encontramos algún error en estos datos, lo enviaremos a la vista sin guardar nada para que el usuario pueda corregirlos.

La vista quedará así:

```
<!-- File: /app/View/Posts/edit.ctp -->

<h1>Edit Post</h1>
<?php
    echo $this->Form->create('Post', array('action' => 'edit'));
    echo $this->Form->input('title');
    echo $this->Form->input('body', array('rows' => '3'));
    echo $this->Form->input('id', array('type' => 'hidden'));
    echo $this->Form->end('Save Post');
```

Mostramos el formulario de edición (con los valores actuales de ese artículo), junto a los errores de validación que hubiese.

Una cosa importante, CakePHP asume que estás editando un modelo si su id está presente en su array de datos. Si no hay un “id” presente, CakePHP asumirá que es un nuevo elemento al llamar a la función save(). Puedes actualizar un poco tu vista “index” para añadir los enlaces de edición de un artículo específico:

```
<!-- File: /app/View/Posts/index.ctp (edit links added) -->

<h1>Blog posts</h1>
<p><?php echo $this->Html->link("Add Post", array('action' => 'add')); ?></p>
<table>
    <tr>
        <th>Id</th>
        <th>Title</th>
        <th>Action</th>
        <th>Created</th>
    </tr>

<!-- Here's where we loop through our $posts array, printing out post info -->

<?php foreach ($posts as $post): ?>
    <tr>
        <td><?php echo $post['Post']['id']; ?></td>
        <td>
            <?php echo $this->Html->link($post['Post']['title'], array('action' => 'view
→', $post['Post']['id']));?>
            </td>
        <td>
            <?php echo $this->Form->postLink(
                'Delete',
                array('action' => 'delete', $post['Post']['id']),
                array('confirm' => 'Are you sure?')
            )?>
            <?php echo $this->Html->link('Edit', array('action' => 'edit', $post['Post']['
→id']));?>
            </td>
        <td><?php echo $post['Post']['created']; ?></td>
    </tr>
<?php endforeach; ?>

</table>
```

Borrando Artículos

Vamos a permitir a los usuarios que borren artículos. Primero, el método en nuestro controlador:

```
function delete($id) {
    if (!$this->request->is('post')) {
        throw new MethodNotAllowedException();
    }
    if ($this->Post->delete($id)) {
        $this->Flash->success('The post with id: ' . $id . ' has been deleted.');
```

```
        $this->redirect(array('action' => 'index'));
    }
}
```

Este método borra un artículo cuyo “id” enviamos como parámetro y usa `$this->Flash->success()` para mostrar un mensaje si ha sido borrado. Luego redirige a “/posts/index”. Si el usuario intenta borrar un artículo mediante una llamada GET, generaremos una excepción. Las excepciones que no se traten, serán procesadas por CakePHP de forma genérica, mostrando una bonita página de error. Hay muchas excepciones a tu disposición *Exceptions* que puedes usar para informar de diversos problemas típicos.

Como estamos ejecutando algunos métodos y luego redirigiendo a otra acción de nuestro controlador, no es necesaria ninguna vista (nunca se usa). Lo que si querrás es actualizar la vista `index.ctp` para incluir el ya habitual enlace:

```
<!-- File: /app/View/Posts/index.ctp -->

<h1>Blog posts</h1>
<p><?php echo $this->Html->link('Add Post', array('action' => 'add')); ?></p>
<table>
    <tr>
        <th>Id</th>
        <th>Title</th>
        <th>Actions</th>
        <th>Created</th>
    </tr>

<!-- Here's where we loop through our $posts array, printing out post info -->

    <?php foreach ($posts as $post): ?>
    <tr>
        <td><?php echo $post['Post']['id']; ?></td>
        <td>
            <?php echo $this->Html->link($post['Post']['title'], array('action' => 'view',
→$post['Post']['id']));?>
        </td>
        <td>
            <?php echo $this->Form->postLink(
                'Delete',
                array('action' => 'delete', $post['Post']['id']),
                array('confirm' => 'Are you sure?'));
            ?>
        </td>
        <td><?php echo $post['Post']['created']; ?></td>
    </tr>
    <?php endforeach; ?>
```

(continué en la próxima página)

(proviene de la página anterior)

```
</table>
```

Nota: Esta vista utiliza el FormHelper para pedir confirmación al usuario antes de borrar un artículo. Además el enlace para borrar el artículo se construye con Javascript para que se realice una llamada POST.

Rutas (*Routes*)

En muchas ocasiones, las rutas por defecto de CakePHP funcionan bien tal y como están. Los desarrolladores que quieren rutas diferentes para mejorar la usabilidad apreciarán la forma en la que CakePHP relaciona las URLs con las acciones de los controladores. Vamos a hacer cambios ligeros para este tutorial.

Para más información sobre las rutas, visita esta referencia [Routes Configuration](#).

Por defecto CakePHP responde a las llamadas a la raíz de tu sitio (por ejemplo `www.example.com/`) usando el controlador `PagesController`, y la acción `display`/"home". Esto muestra la página de bienvenida con información de CakePHP que ya has visto. Vamos a cambiar esto mediante una nueva regla.

Las reglas de enrutamiento están en `/app/Config/routes.php`. Comentaremos primero la regla de la que hemos hablado:

```
Router::connect('/', array('controller' => 'pages', 'action' => 'display', 'home'));
```

Como habíamos dicho, esta regla conecta la URL `/` con el controlador `pages` la acción `display` y le pasa como parámetro `home`, así que reemplazaremos esta regla por esta otra:

```
Router::connect('/', array('controller' => 'posts', 'action' => 'index'));
```

Ahora la URL `/` nos llevará al controlador `posts` y la acción `index`.

Nota: CakePHP también calcula las rutas a la inversa. Si en tu código pasas el array `array('controller' => 'posts', 'action' => 'index')` a una función que espera una url, el resultado será `/`. Es buena idea usar siempre arrays para configurar las URL, lo que asegura que los links irán siempre al mismo lugar.

Conclusión

Creando aplicaciones de este modo te traerá paz, amor, dinero a carretas e incluso te conseguirá lo demás que puedas querer. Así de simple.

Ten en cuenta que este tutorial es muy básico, CakePHP tiene *muchas* otras cosas que harán tu vida más fácil, y es flexible aunque no hemos cubierto aquí estos puntos para que te sea más simple al principio. Usa el resto de este manual como una guía para construir mejores aplicaciones (recuerda todo los los beneficios que hemos mencionado un poco más arriba)

Ahora ya estás preparado para la acción. Empieza tu propio proyecto, lee el resto del manual y el [API Manual API](#)¹²⁸.

¹²⁸ <https://api.cakephp.org/2.x/>

Lectura sugerida para continuar desde aquí

1. *Layouts*: Personaliza la plantilla *layout* de tu aplicación
2. *Elementos* Incluir vistas y reutilizar trozos de código
3. *Scaffolding*: Prototipos antes de trabajar en el código final
4. *Code Generation with Bake* Generación básica de CRUDs
5. *Authentication*: Gestión de usuarios y permisos

Aplicación con autenticación y autorizacion

Siguiendo con nuestro ejemplo de aplicacion *Parte 1: Tutorial para desarrollar el Blog*, imaginá que necesitamos proteger ciertas URLs, dependiendo del usuario logeado. También tenemos otro requisito, permitir que nuestro blog tenga varios autores, cada uno habilitado para crear sus posts, editar y borrarlos a voluntad, evitando que otros autores puedan cambiarlos.

Creando el codigo para usuarios

Primero, vamos a crear una tabla en nuestra base de datos para guardar los datos de usuarios:

```
CREATE TABLE users (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50),
    password VARCHAR(255),
    role VARCHAR(20),
    created DATETIME DEFAULT NULL,
    modified DATETIME DEFAULT NULL
);
```

Siguimos las convenciones de CakePHP para nombrar tablas pero también estamos aprovechando otra convencion: al usar los campos username y password en nuestra tabla CakePHP configurará automáticamente la mayoría de las cosas al momento de implementar el login.

El siguiente paso es crear el modelo para la tabla User que será responsable de buscar, guardar y validar los datos de usuario:

```
// app/Model/User.php
App::uses('AppModel', 'Model');

class User extends AppModel {
    public $validate = array(
        'username' => array(
            'required' => array(
                'rule' => 'notBlank',
                'message' => 'A username is required'
            )
        ),
        'password' => array(
            'required' => array(
                'rule' => 'notBlank',
```

(continué en la próxima página)

(proviene de la página anterior)

```

        'message' => 'A password is required'
    )
),
'role' => array(
    'valid' => array(
        'rule' => array('inList', array('admin', 'author')),
        'message' => 'Please enter a valid role',
        'allowEmpty' => false
    )
)
);
}

```

También vamos a crear UsersController; el siguiente contenido fue generado usando *baked* UsersController con el generador de código incluido con CakePHP:

```

// app/Controller/UsersController.php
App::uses('AppController', 'Controller');

class UsersController extends AppController {

    public function beforeFilter() {
        parent::beforeFilter();
        $this->Auth->allow('add');
    }

    public function index() {
        $this->User->recursive = 0;
        $this->set('users', $this->paginate());
    }

    public function view($id = null) {
        $this->User->id = $id;
        if (!$this->User->exists()) {
            throw new NotFoundException(__('Invalid user'));
        }
        $this->set('user', $this->User->findById($id));
    }

    public function add() {
        if ($this->request->is('post')) {
            $this->User->create();
            if ($this->User->save($this->request->data)) {
                $this->Session->setFlash(__('The user has been saved'));
                return $this->redirect(array('action' => 'index'));
            }
            $this->Session->setFlash(
                __('The user could not be saved. Please, try again.')
            );
        }
    }
}

```

(continué en la próxima página)

(proviene de la página anterior)

```

public function edit($id = null) {
    $this->User->id = $id;
    if (!$this->User->exists()) {
        throw new NotFoundException(__('Invalid user'));
    }
    if ($this->request->is('post') || $this->request->is('put')) {
        if ($this->User->save($this->request->data)) {
            $this->Session->setFlash(__('The user has been saved'));
            return $this->redirect(array('action' => 'index'));
        }
        $this->Session->setFlash(
            __('The user could not be saved. Please, try again.')
        );
    } else {
        $this->request->data = $this->User->findById($id);
        unset($this->request->data['User']['password']);
    }
}

public function delete($id = null) {
    // Prior to 2.5 use
    // $this->request->onlyAllow('post');

    $this->request->allowMethod('post');

    $this->User->id = $id;
    if (!$this->User->exists()) {
        throw new NotFoundException(__('Invalid user'));
    }
    if ($this->User->delete()) {
        $this->Session->setFlash(__('User deleted'));
        return $this->redirect(array('action' => 'index'));
    }
    $this->Session->setFlash(__('User was not deleted'));
    return $this->redirect(array('action' => 'index'));
}
}

```

Distinto en la versión 2.5: A partir de la versión 2.5, usamos `CakeRequest::allowMethod()` en lugar de `CakeRequest::onlyAllow()` (depreciado).

De la misma forma que creamos las vistas para los posts del blog o usando la herramienta de generación de código, creamos las vistas. Para los objetivos de este tutorial, mostraremos solamente `add.ctp`:

```

<!-- app/View/Users/add.ctp -->
<div class="users form">
<?php echo $this->Form->create('User'); ?>
<fieldset>
    <legend><?php echo __('Add User'); ?></legend>
    <?php echo $this->Form->input('username');
    echo $this->Form->input('password');
    echo $this->Form->input('role', array(

```

(continué en la próxima página)

(proviene de la página anterior)

```

        'options' => array('admin' => 'Admin', 'author' => 'Author')
    ));
?>
</fieldset>
<?php echo $this->Form->end(__('Submit')); ?>
</div>

```

Autenticación (login y logout)

Ya estamos listos para agregar nuestra autenticación. En CakePHP esto es manejado por la clase `AuthComponent`, responsable de requerir login para ciertas acciones, de manejar el sign-in y el sign-out y también de autorizar usuarios logeados a ciertas acciones que están autorizados a utilizar.

Para agregar este componente a tu aplicación, abrí `app/Controller/AppController.php` y agregó las siguientes líneas:

```

// app/Controller/AppController.php
class AppController extends Controller {
    //...

    public $components = array(
        'Session',
        'Auth' => array(
            'loginRedirect' => array(
                'controller' => 'posts',
                'action' => 'index'
            ),
            'logoutRedirect' => array(
                'controller' => 'pages',
                'action' => 'display',
                'home'
            ),
            'authenticate' => array(
                'Form' => array(
                    'passwordHasher' => 'Blowfish'
                )
            )
        )
    );

    public function beforeFilter() {
        $this->Auth->allow('index', 'view');
    }
    //...
}

```

No hay mucho que configurar, al haber utilizado convenciones para la tabla de usuarios. Simplemente asignamos las URLs que serán cargadas después del login y del logout, en nuestro caso `/posts/` y `/` respectivamente.

Lo que hicimos en `beforeFilter` fue decirle al `AuthComponent` que no requiera login para las acciones `index` y `view` en cada controlador. Queremos que nuestros visitantes puedan leer y listar las entradas sin registrarse.

Ahora necesitamos poder registrar nuevos usuarios, guardar el nombre de usuario y contraseña, y hashear su contraseña

para que no sea guardada como texto plano. Vamos a decirle al AuthComponent que deje usuarios sin autenticar acceder a la función add del controlador users e implementemos las acciones de login y logout:

```
// app/Controller/UsersController.php

public function beforeFilter() {
    parent::beforeFilter();
    // Allow users to register and logout.
    $this->Auth->allow('add', 'logout');
}

public function login() {
    if ($this->request->is('post')) {
        if ($this->Auth->login()) {
            return $this->redirect($this->Auth->redirectUrl());
        }
        $this->Session->setFlash(__('Invalid username or password, try again'));
    }
}

public function logout() {
    return $this->redirect($this->Auth->logout());
}
```

El hasheo del password aún no esta hecho, abrí app/Model/User.php y agregá la siguientes líneas:

```
// app/Model/User.php

App::uses('AppModel', 'Model');
App::uses('BlowfishPasswordHasher', 'Controller/Component/Auth');

class User extends AppModel {

    // ...

    public function beforeSave($options = array()) {
        if (isset($this->data[$this->alias]['password'])) {
            $passwordHasher = new BlowfishPasswordHasher();
            $this->data[$this->alias]['password'] = $passwordHasher->hash(
                $this->data[$this->alias]['password']
            );
        }
        return true;
    }

    // ...
}
```

Nota: El BlowfishPasswordHasher usa un algoritmo de hasheo mas fuerte (bcrypt) que SimplePasswordHasher (sha1) y provee salts por usuario. SimplePasswordHasher será removido en la version 3.0 de CakePHP

Entonces, cada vez que un usuario sea guardado, el password es hasheado usando la clase BlowfishPasswordHasher. Solamente nos falta una vista para la función de login. Abrí app/View/Users/login.ctp y agregá las siguientes líneas:

```
//app/View/Users/login.ctp

<div class="users form">
<?php echo $this->Session->flash('auth'); ?>
<?php echo $this->Form->create('User'); ?>
    <fieldset>
        <legend>
            <?php echo __('Please enter your username and password'); ?>
        </legend>
        <?php echo $this->Form->input('username');
        echo $this->Form->input('password');
    ?>
    </fieldset>
<?php echo $this->Form->end(__('Login')); ?>
</div>
```

Ya podés registrar un nuevo usuario accediendo a `/users/add` e iniciar sesión con las nuevas credenciales ingresando a `/users/login`. También al intentar acceder a alguna otra URL que no fue explícitamente autorizada, por ejemplo `/posts/add`, la aplicación te redireccionará automáticamente a la página de login.

Y eso es todo! Se ve demasiado simple para ser verdad. Volvamos un poco para explicar que pasa. La función `beforeFilter` le dice al `AuthComponent` que no requiera login para la acción `add` así como para `index` y `view`, autorizadas en el `beforeFilter` del `AppController`.

La función `login` llama a `$this->Auth->login()` del `AuthComponent`, y funciona sin ninguna otra configuración ya que seguimos la convención. Es decir, tener un modelo llamado `User` con los campos `username` y `password`, y usar un formulario que hace `post` a un controlador con los datos del usuario. Esta función devuelve si el login fue exitoso o no, y en caso de que tenga éxito redirige a la URL puesta en `AppController`, dentro de la configuración del `AuthComponent`.

El `logout` funciona simplemente al acceder a `/users/logout` y redirecciona al usuario a la URL configurada.

Autorización (quién está autorizado a acceder qué)

Como mencionamos antes, estamos convirtiendo este blog en una herramienta de autoría multiusuario, y para hacer esto necesitamos modificar la tabla de `posts` para agregar referencia al modelo `User`:

```
ALTER TABLE posts ADD COLUMN user_id INT(11);
```

También, un pequeño cambio en `PostsController` es necesario para guardar el usuario logeado como referencia en los `posts` creados:

```
// app/Controller/PostsController.php
public function add() {
    if ($this->request->is('post')) {
        //Added this line
        $this->request->data['Post']['user_id'] = $this->Auth->user('id');
        if ($this->Post->save($this->request->data)) {
            $this->Session->setFlash(__('Your post has been saved.'));
            return $this->redirect(array('action' => 'index'));
        }
    }
}
```

La función `user()` del `AuthComponent` devuelve datos del usuario actualmente logeado. Usamos este método para agregar datos a la información que será guardada.

Vamos a prevenir que autores puedan editar o eliminar los posts de otros autores. La regla básica para nuestra aplicación es que los usuarios admin pueden acceder todas las URL, mientras que los usuarios normales (autores) solamente pueden acceder las acciones permitidas. Abrí nuevamente AppController y agregó las siguientes opciones en la configuración del Auth:

```
// app/Controller/AppController.php

public $components = array(
    'Session',
    'Auth' => array(
        'loginRedirect' => array('controller' => 'posts', 'action' => 'index'),
        'logoutRedirect' => array(
            'controller' => 'pages',
            'action' => 'display',
            'home'
        ),
        'authenticate' => array(
            'Form' => array(
                'passwordHasher' => 'Blowfish'
            )
        ),
        'authorize' => array('Controller') // Added this line
    )
);

public function isAuthorized($user) {
    // Admin can access every action
    if (isset($user['role']) && $user['role'] === 'admin') {
        return true;
    }

    // Default deny
    return false;
}
```

Creamos un mecanismo de autorización muy simple. En este caso, los usuarios con el rol admin podrán acceder a cualquier URL del sitio cuando esten logeados, pero el resto de los usuarios no podrán hacer más que los usuarios no logeados.

Esto no es exactamente lo que queríamos, por lo que tendremos que agregar mas reglas a nuestro método `isAuthorized()`. Pero en lugar de hacerlo en AppController, vamos a delegar a cada controlador. Las reglas que vamos a agregar a PostsController deberían permitirle a los autores crear posts, pero prevenir que editen posts que no le pertenezcan. Abrí el archivo `PostsController.php` y agregó las siguientes líneas:

```
// app/Controller/PostsController.php

public function isAuthorized($user) {
    // All registered users can add posts
    if ($this->action === 'add') {
        return true;
    }

    // The owner of a post can edit and delete it
    if (in_array($this->action, array('edit', 'delete'))) {
```

(continué en la próxima página)

(proviene de la página anterior)

```
        $postId = (int) $this->request->params['pass'][0];
        if ($this->Post->isOwnedBy($postId, $user['id'])) {
            return true;
        }
    }

    return parent::isAuthorized($user);
}
```

Estamos sobrescribiendo el método `isAuthorized()` de `AppController` y comprobando si la clase padre autoriza al usuario. Si no lo hace entonces solamente autorizarlo a acceder a la acción `add` y condicionalmente acceder a `edit` y `delete`. Una última cosa por implementar, decidir si el usuario está autorizado a editar el post o no, estamos llamando la función `isOwnedBy()` del modelo `Post`. Es en general una buena practica mover la mayor parte de la logica posible hacia los modelos:

```
// app/Model/Post.php

public function isOwnedBy($post, $user) {
    return $this->field('id', array('id' => $post, 'user_id' => $user)) !== false;
}
```

Esto concluye nuestro simple tutorial de autenticación y autorización. Para proteger el `UsersController` se puee seguir la misma técnica utilizada para `PostsController`. También es posible implementar una solución mas general en `AppController`, de acuerdo a tus reglas.

En caso de necesitar más control, sugerimos leer la guía completa sobre Auth en [Authentication](#), donde encontrarás mas información para configurar el componente y crear clases de autorizacion a tú medida.

Lectura sugerida

1. *Code Generation with Bake* Generar código CRUD básico
2. *Authentication*: Registro y login de usuarios

Simple Acl controlled Application

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹²⁹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

¹²⁹ <https://github.com/cakephp/docs>

Simple Acl controlled Application - part 2

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹³⁰ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

¹³⁰ <https://github.com/cakephp/docs>

Contributing

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹³¹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

There are a number of ways you can contribute to CakePHP. The following sections cover the various ways you can contribute to CakePHP:

Documentation

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹³² o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

¹³¹ <https://github.com/cakephp/docs>

¹³² <https://github.com/cakephp/docs>

Tickets

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹³³ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Code

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹³⁴ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Coding Standards

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹³⁵ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Backwards Compatibility Guide

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹³⁶ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

¹³³ <https://github.com/cakephp/docs>

¹³⁴ <https://github.com/cakephp/docs>

¹³⁵ <https://github.com/cakephp/docs>

¹³⁶ <https://github.com/cakephp/docs>

CakePHP Development Process

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹³⁷ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

¹³⁷ <https://github.com/cakephp/docs>

Appendices

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹³⁸ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Appendices contain information regarding the new features introduced in 2.x, and the migration path from 1.3 to 2.0.

2.10 Migration Guide

2.10 Migration Guide

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹³⁹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

¹³⁸ <https://github.com/cakephp/docs>

¹³⁹ <https://github.com/cakephp/docs>

2.9 Migration Guide

2.9 Migration Guide

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁴⁰ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

2.8 Migration Guide

2.8 Migration Guide

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁴¹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

2.7 Migration Guide

2.7 Migration Guide

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁴² o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

¹⁴⁰ <https://github.com/cakephp/docs>

¹⁴¹ <https://github.com/cakephp/docs>

¹⁴² <https://github.com/cakephp/docs>

2.6 Migration Guide

2.6 Migration Guide

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁴³ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

2.5 Migration Guide

2.5 Migration Guide

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁴⁴ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

2.4 Migration Guide

2.4 Migration Guide

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁴⁵ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

¹⁴³ <https://github.com/cakephp/docs>

¹⁴⁴ <https://github.com/cakephp/docs>

¹⁴⁵ <https://github.com/cakephp/docs>

2.3 Migration Guide

2.3 Migration Guide

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁴⁶ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

2.2 Migration Guide

2.2 Migration Guide

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁴⁷ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

2.1 Migration Guide

2.1 Migration Guide

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁴⁸ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

¹⁴⁶ <https://github.com/cakephp/docs>

¹⁴⁷ <https://github.com/cakephp/docs>

¹⁴⁸ <https://github.com/cakephp/docs>

New Features in CakePHP 2.1

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁴⁹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

2.0 Migration Guide

2.0 Migration Guide

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁵⁰ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

New Features in CakePHP 2.0

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁵¹ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

PHPUnit Migration Hints

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁵² o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

¹⁴⁹ <https://github.com/cakephp/docs>

¹⁵⁰ <https://github.com/cakephp/docs>

¹⁵¹ <https://github.com/cakephp/docs>

¹⁵² <https://github.com/cakephp/docs>

Migration from 1.2 to 1.3

Migrating from CakePHP 1.2 to 1.3

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁵³ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

New features in CakePHP 1.3

Nota: La documentación no es compatible actualmente con el idioma español en esta página.

Por favor, siéntase libre de enviarnos un pull request en [Github](#)¹⁵⁴ o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

General Information

Glossary

routing array

(Enrutamiento de array) Un array de atributos que son pasados a `Router::url()`. Usualmente se tipean de la siguiente manera:

```
array('controller' => 'posts', 'action' => 'view', 5)
```

HTML attributes

(Atributos HTML) Un array de llaves => valores que se compone en HTML attributes(atributos HTML). Por ejemplo:

```
// Dado
array('class' => 'my-class', 'target' => '_blank')

// Generara
class="my-class" target="_blank"
```

Si un atributo acepta su nombre(de atributo) como valor, se puede usar `true`:

¹⁵³ <https://github.com/cakephp/docs>

¹⁵⁴ <https://github.com/cakephp/docs>

```
// Dado
array('checked' => true)

// Generara
checked="checked"
```

plugin syntax

(Sintaxis de plugin) Plugin syntax(Sintaxis de plugin) se refiere al punto separador del nombre de la clase indicación que la clase es parte de un plugin. Por ej. ``DebugKit.Toolbar``. El plugin es DebugKit y el nombre de la clase es Toolbar.

dot notation

(Notación por puntos) Dot notation(notación por puntos) define la ruta dentro un array, separando valores anidados con . Por ejemplo:

```
Asset.filter.css
```

Apuntara al siguiente valor:

```
array(
    'Asset' => array(
        'filter' => array(
            'css' => 'got me'
        )
    )
)
```

CSRF

Cross Site Request Forgery(falsificación de petición en sitios cruzados). Previene ataques de reproducción,doble envío y peticiones falsas de otros dominios.

routes.php

Un archivo en APP/Config que contiene configuración de enrutamiento. Este archivo es incluido antes de que cualquier petición se procese. Debe conectar todas las rutas que tu aplicación necesita para que las peticiones puedan ser enrutadas al controlador + acción que corresponda.

DRY

Don't repeat yourself(No te repitas a ti mismo). Es un principio del desarrollo de software apuntado a reducir la repetición de información de cualquier tipo. En CakePHP DRY es usado para permitirte programar to código una vez, y re-usarlo en distintas partes de tu aplicación.

Indices and tables

- `genindex`

Símbolos

`__construct()` (método de Component), 48

A

`AclBehavior` (class), 69

`AclComponent` (class), 73

`addScript()` (método de View), 59

`App` (class), 68

`append()` (método de View), 60

`assign()` (método de View), 60

`AuthComponent` (class), 72

B

`beforeRedirect()` (método de Component), 48

`beforeRender()` (método de Component), 48

`Blocks` (View propiedad), 61

`blocks()` (método de View), 59

C

`CacheHelper` (class), 74

`CakeEmail` (class), 78

`CakeNumber` (class), 80

`CakeTime` (class), 82

`Component` (class), 48

`ContainableBehavior` (class), 70

`CookieComponent` (class), 73

CSRF, 129

D

dot notation, 129

DRY, 129

E

`element()` (método de View), 59

`elementCache` (View propiedad), 60

`end()` (método de View), 60

`extend()` (método de View), 60

F

`fetch()` (método de View), 60

`FlashComponent` (class), 71

`FlashHelper` (class), 74

`FormHelper` (class), 75

G

`get()` (método de View), 59

`getVar()` (método de View), 59

`getVars()` (método de View), 59

H

`Hash` (class), 79

HTML attributes, 128

`HtmlHelper` (class), 75

`HttpSocket` (class), 79

I

`Inflector` (class), 79

`initialize()` (método de Component), 48

J

`JsHelper` (class), 75

L

`layout` (View propiedad), 60

M

`MediaView` (class), 61

N

`NumberHelper` (class), 75

O

`output` (View propiedad), 60

P

PaginatorComponent (class), [71](#)
PaginatorHelper (class), [76](#)
plugin syntax, [129](#)
prepend() (método de View), [60](#)

R

request (View propiedad), [60](#)
RequestHandlerComponent (class), [72](#)
routes.php, [129](#)
routing array, [128](#)
RssHelper (class), [76](#)

S

Security (class), [81](#)
SecurityComponent (class), [72](#)
SessionComponent (class), [72](#)
SessionHelper (class), [76](#)
Set (class), [81](#)
set() (método de View), [59](#)
shutdown() (método de Component), [48](#)
start() (método de View), [60](#)
startIfEmpty() (método de View), [60](#)
startup() (método de Component), [48](#)
String (class), [81](#)

T

TextHelper (class), [77](#)
TimeHelper (class), [77](#)
TranslateBehavior (class), [70](#)
TreeBehavior (class), [70](#)

U

uuid() (método de View), [59](#)

V

View (class), [59](#)

X

Xml (class), [82](#)