CakePHP Cookbook Documentation

Publicación 2.x

Cake Software Foundation

nov. 07, 2016

Índice general

CAPÍTULO 1

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.

CAPÍTULO 2

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`

[1] http://git-scm.com/

```
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.

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. Probabmente 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-rUl3Z',
    '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 ficharo `/app/Config/core.php` línea 208. No importa qué numero 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*

CAPÍTULO 3

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 ficheró 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 variab le '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>
    </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 /views/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 ritas 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 asi 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 `read()` 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 incluír 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 acceder la 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 /views/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 /models/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 excepciónes que no se traten, serán procesadas por CakePHP de forma genérica, mostrando una bonita página de error. Hay muchas excepciones a tu disposición /development/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 incluír 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; ?>

</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 desarroladores 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[2].

Lectura sugerida para continuar desde aquí

1. view-layouts: Personaliza la plantilla *layout* de tu aplicación
2. view-elements Incluír vistas y reutilizar trozos de código
3. /controllers/scaffolding: Prototipos antes de trabajar en el código final
4. /console-and-shells/code-generation-with-bake Generación básica de CRUDs
5. /core-libraries/components/authentication: Gestión de usuarios y permisos

Lectura Adicional

Diseccionando 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] http://api.cakephp.org/2.8/

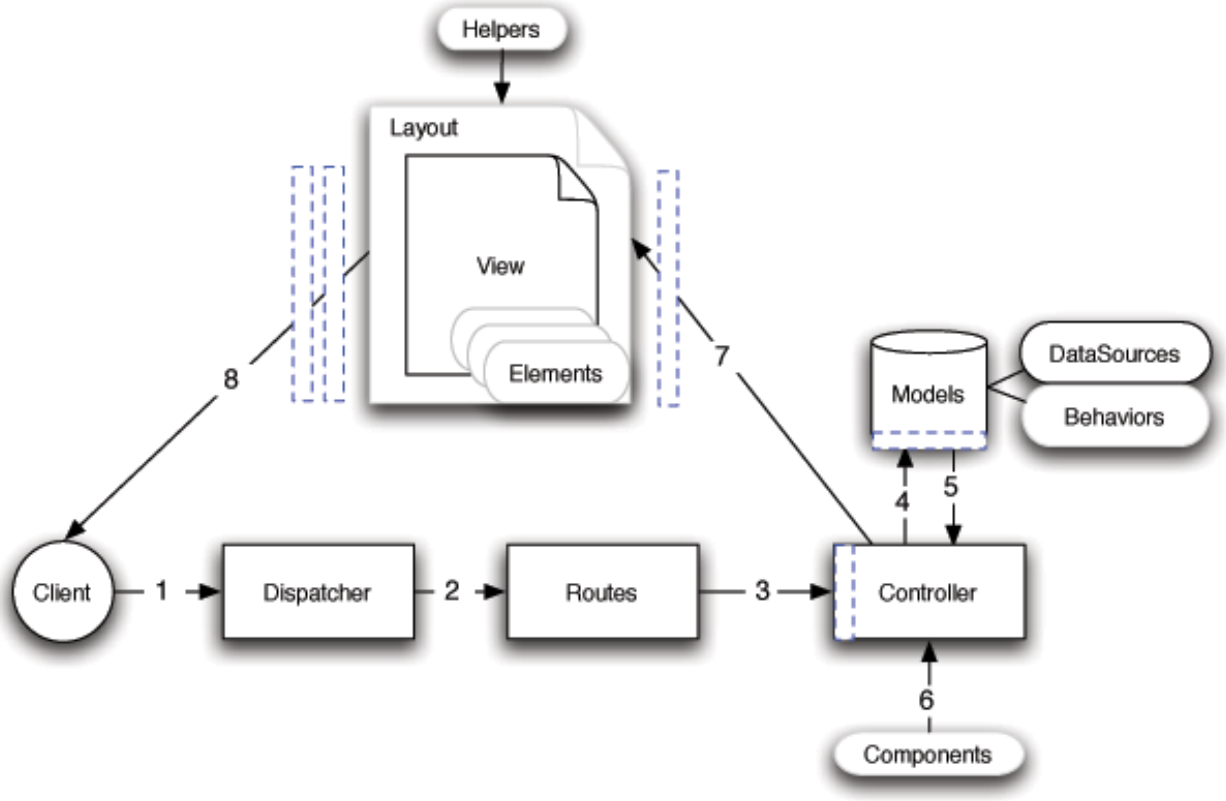

Figura 3.1: Diagrama de flujo que muestra un Request típico en CakePHP

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 incluír 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 sóla 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) **EspeciallyFunkableBehavior** estará en el fichero **EspeciallyFunkableBehavior.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 `Inflector` para comprobar el singular y plural de las palabras. Consulta la documentación aquí /core-utility-libraries/inflector.

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' (belongsTo) 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 exisitr 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 pryectos. 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 componentes 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 incluído 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.

Extendiento 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.

CAPÍTULO 4

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 vistar la página oficial http://cakephp.org y pichar en la opción "Download Now".

Además, todas las versiones de CakePHP están hospedadas en Github[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[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[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, asegúrarte 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 accesible 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 requistos.

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` al 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
-- fallthrough 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",
                ".xtmpl", "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;
    }
}
```

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>
```

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 neceites 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

CAPÍTULO 5

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 teconolgías como SQL, JavaScript y XML y este manual no trata de explicarlas, sino sólo la forma en que se utilizados en su contexto.

¿Qué es CakePHP? y ¿Por qué usarlo?

CakePHP[7] es un famework[8] libre[9], de código abierto[10], para el desarrollo rápido de aplicaciones[11] para PHP[12]. 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, obten una copia de CakePHP y empieza con el verdadero corazón de tu aplicación.

CakePHP tiene un equido de desarollo 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:

[7] http://www.cakephp.org/
[8] http://en.wikipedia.org/wiki/Application_framework
[9] http://en.wikipedia.org/wiki/MIT_License
[10] http://en.wikipedia.org/wiki/Open_source
[11] http://en.wikipedia.org/wiki/Rapid_application_development
[12] http://www.php.net/
[13] http://github.com/cakephp/cakephp/contributors

- Licencia flexible[14]
- 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.

[14] http://en.wikipedia.org/wiki/MIT_License
[15] http://en.wikipedia.org/wiki/Create,_read,_update_and_delete
[16] http://en.wikipedia.org/wiki/Scaffold_(programming)
[17] http://en.wikipedia.org/wiki/Model-view-controller
[18] http://en.wikipedia.org/wiki/Data_validation
[19] http://en.wikipedia.org/wiki/Web_template_system
[20] http://en.wikipedia.org/wiki/Access_control_list
[21] http://en.wikipedia.org/wiki/Web_cache
[22] http://en.wikipedia.org/wiki/Model-view-controller

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 amacenamiento 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 “Amigo”, “Usuario”, “Comentario” y “Foto”.

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

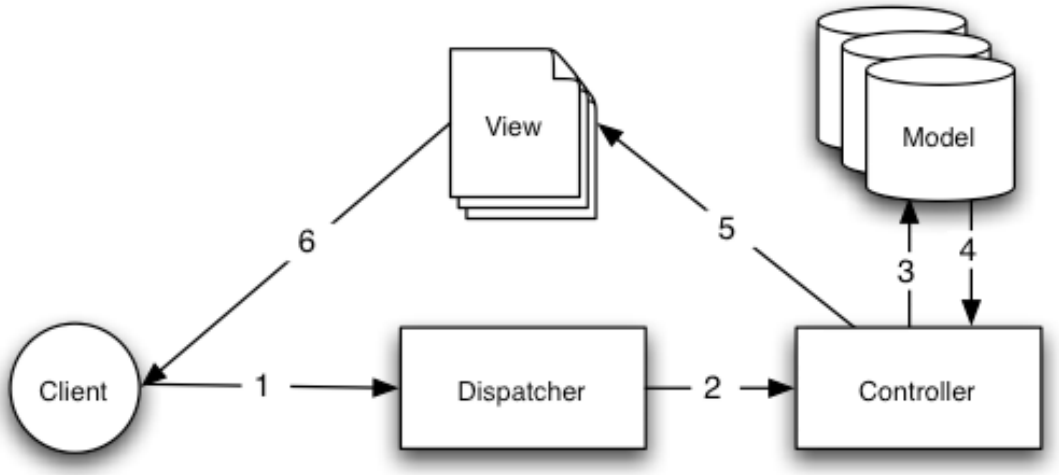

Figure: 1: Una petición MVC típica

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 rápidos[23].

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

[23] http://en.wikipedia.org/wiki/Software_prototyping

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

http://www.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

http://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

http://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.

El API

http://api.cakephp.org/2.8/

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 alli 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: http://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] http://cakephp-fr.org

CAPÍTULO 6

ES - Deployment

Steps to deploy on a Hosting Server

CAPÍTULO 7

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[25] y en Bakery[26] 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.

¡ Vamos allá !

[25] http://plugins.cakephp.org/
[26] http://bakery.cakephp.org/

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[27]. `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());
```

[27] http://git-scm.com/

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.

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. Probabmente 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-rUl3Z',
    '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 ficharo `/app/Config/core.php` línea 208. No importa qué numero 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 ficheró 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 variab le '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>
    </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 /views/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 ritas 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 asi 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 `read()` 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 incluír 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 acceder la 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 /views/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 /models/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 excepciónes que no se traten, serán procesadas por CakePHP de forma genérica, mostrando una bonita página de error. Hay muchas excepciones a tu disposición /development/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 incluír 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; ?>

</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[28].

Lectura sugerida para continuar desde aquí

1. view-layouts: Personaliza la plantilla *layout* de tu aplicación
2. view-elements Incluír vistas y reutilizar trozos de código
3. /controllers/scaffolding: Prototipos antes de trabajar en el código final
4. /console-and-shells/code-generation-with-bake Generación básica de CRUDs
5. /core-libraries/components/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),
```

[28] http://api.cakephp.org/2.8/

```
    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 mayoria 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',
                '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 incluído 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.')
            );
        }
    }

    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 version 2.5, usamos `CakeRequest::allowMethod()` en lugar de `CakeRequest::onlyAllow()` (deprecated).

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(
            '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 estan autorizados a utilizar.

Para agregar este componente a tú aplicación, abrí `app/Controller/AppController.php` y agregá las siguientes lineas:

```
// 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 despues 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 funcion 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 lineas:

```
// 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 lineas:

```
//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 explicitamente

autorizada, por ejemplo `/posts/add`, la aplicación te redireccionará automaticamente al la pagina 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` asi 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 exito redirige a la URL puesta en AppController, dentro de la configuracion 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 agregá 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 queriamos, 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 deberian permitirle a los autores crear posts, pero prevenir que editen posts que no le pertenezcan. Abrí el archivo `PostsController.php` y agregá las siguientes lineas:

```
// 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'))) {
        $postId = (int) $this->request->params['pass'][0];
        if ($this->Post->isOwnedBy($postId, $user['id'])) {
            return true;
        }
    }
```

```
    return parent::isAuthorized($user);
}
```

Estamos sobreescribiendo 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 autorizador 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 guia completa sobre Auth en /core-libraries/components/authentication, donde encontrarás mas información para configurar el componente y crear clases de autorizacion a tú medida.

Lectura sugerida

1. /console-and-shells/code-generation-with-bake Generar código CRUD básico
2. /core-libraries/components/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[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.

Simple Acl controlled Application - part 2

[29] https://github.com/cakephp/docs

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.

[30] https://github.com/cakephp/docs

CAPÍTULO 8

Indices and tables

- genindex