CakePHP Cookbook Documentation

Versión 4.next

Cake Software Foundation

26 de noviembre de 2024

Índice general

CAPÍTULO 1

CakePHP de un vistazo

CakePHP está diseñado para hacer tareas habituales de desarrollo web simples y fáciles. Proporciona una caja de herramientas todo-en-uno y para que puedas empezar rápidamente, las diferentes partes de CakePHP trabajan correctamente de manera conjunta o separada.

El objetivo de este artículo es introducirte en los conceptos generales de CakePHP y darte un rápido vistazo sobre como esos conceptos están implementados en CakePHP. Si estás deseando comenzar un proyecto puedes *empezar con el tutorial*, o profundizar en la documentación.

Convenciones sobre configuración

CakePHP proporciona una estructura organizativa básica que cubre los nombres de las clases, archivos, tablas de base de datos y otras convenciones más. Aunque lleva algo de tiempo aprender las convenciones, siguiéndolas CakePHP evitará que tengas que hacer configuraciones innecesarias y hará que la estructura de la aplicación sea uniforme y que el trabajo con varios proyectos sea sencillo. El capítulo de *convenciones* muestra las que son utilizadas en CakePHP.

La capa Modelo

La capa Modelo representa la parte de tu aplicación que implementa la lógica de negocio. Es la responsable de obtener datos y convertirlos en los conceptos que utiliza tu aplicación. Esto incluye procesar, validar, asociar u otras tareas relacionadas con el manejo de datos.

En el caso de una red social la capa modelo se encargaría de tareas como guardar los datos del usuario, las asociaciones de amigos, almacenar y obtener fotos, buscar sugerencias de amistad, etc. Los objetos modelo serían «Amigo», «Usuario», «Comentario» o «Foto». Si quisieramos obtener más datos de nuestra tabla `usuarios` podríamos hacer lo siguiente:

```
use Cake\ORM\TableRegistry;

// Prior to 3.6 use TableRegistry::get('Usuarios')
$usuarios = TableRegistry::getTableLocator()->get('Usuarios');
$query = $usuarios->find();
foreach ($query as $row) {
    echo $row->nombreusuario;
}
```

Como te habrás dado cuenta no hemos necesitado escribir ningún código previo para empezar a trabajar con nuestros datos. Al utilizar las convenciones CakePHP usará clases estándar para tablas y clases de entidad que no hayan sido definidas.

Si queremos crear un nuevo usuario y guardarlo (con validaciones) podríamos hacer algo como:

```
use Cake\ORM\TableRegistry;

// Prior to 3.6 use TableRegistry::get('Usuarios')
$usuarios = TableRegistry::getTableLocator()->get('Usuarios');
$usuario = $usuarios->newEntity(['email' => 'mark@example.com']);
$usuarios->save($usuario);
```

La capa Vista

La capa Vista renderiza una presentación de datos modelados. Separada de los objetos Modelo, es la responsable de usar la información disponible para producir cualquier interfaz de presentación que pueda necesitar tu aplicación.

Por ejemplo, la vista podría usar datos del modelo para renderizar una plantilla HTML que los contenga o un resultado en formato XML:

```
// En un archivo de plantilla de vista renderizaremos un 'element' para cada usuario.
<?php foreach ($usuarios as $usuario): ?>
    <li class="usuario">
        <?= $this->element('usuario', ['usuario' => $usuario]) ?>
    </li>
<?php endforeach; ?>
```

La capa Vista proporciona varias extensiones como *Plantillas de vistas*, *Elementos* y *View Cells* que te permiten reutilizar tu lógica de presentación.

Esta capa no se limita a representaciones HTML o texto de los datos. Puede utilizarse para otros formatos habituales como JSON, XML y a través de una arquitectura modular, cualquier otro formato que puedas necesitar como CSV.

La capa Controlador

La capa Controlador maneja peticiones de usuarios. Es la responsable de elaborar una respuesta con la ayuda de las capas Modelo y Vista.

Un controlador puede verse como un gestor que asegura que todos los recursos necesarios para completar una tarea son delegados a los trabajadores oportunos. Espera por las peticiones de los clientes, comprueba la validez de acuerdo con las reglas de autenticación y autorización, delega la búsqueda o procesado de datos al modelo, selecciona el tipo de presentación que el cliente acepta y finalmente delega el proceso de renderizado a la capa Vista. Un ejemplo de controlador para el registro de un usuario sería:

```
public function add()
{
    $usuario = $this->Usuarios->newEntity();
    if ($this->request->is('post')) {
        $usuario = $this->Usuarios->patchEntity($usuario, $this->request->getData());
        if ($this->Usuarios->save($usuario, ['validate' => 'registration'])) {
            $this->Flash->success(__('Ahora estás registrado.'));
        } else {
            $this->Flash->error(__('Hubo algunos problemas.'));
        }
    }
    $this->set('usuario', $usuario);
}
```

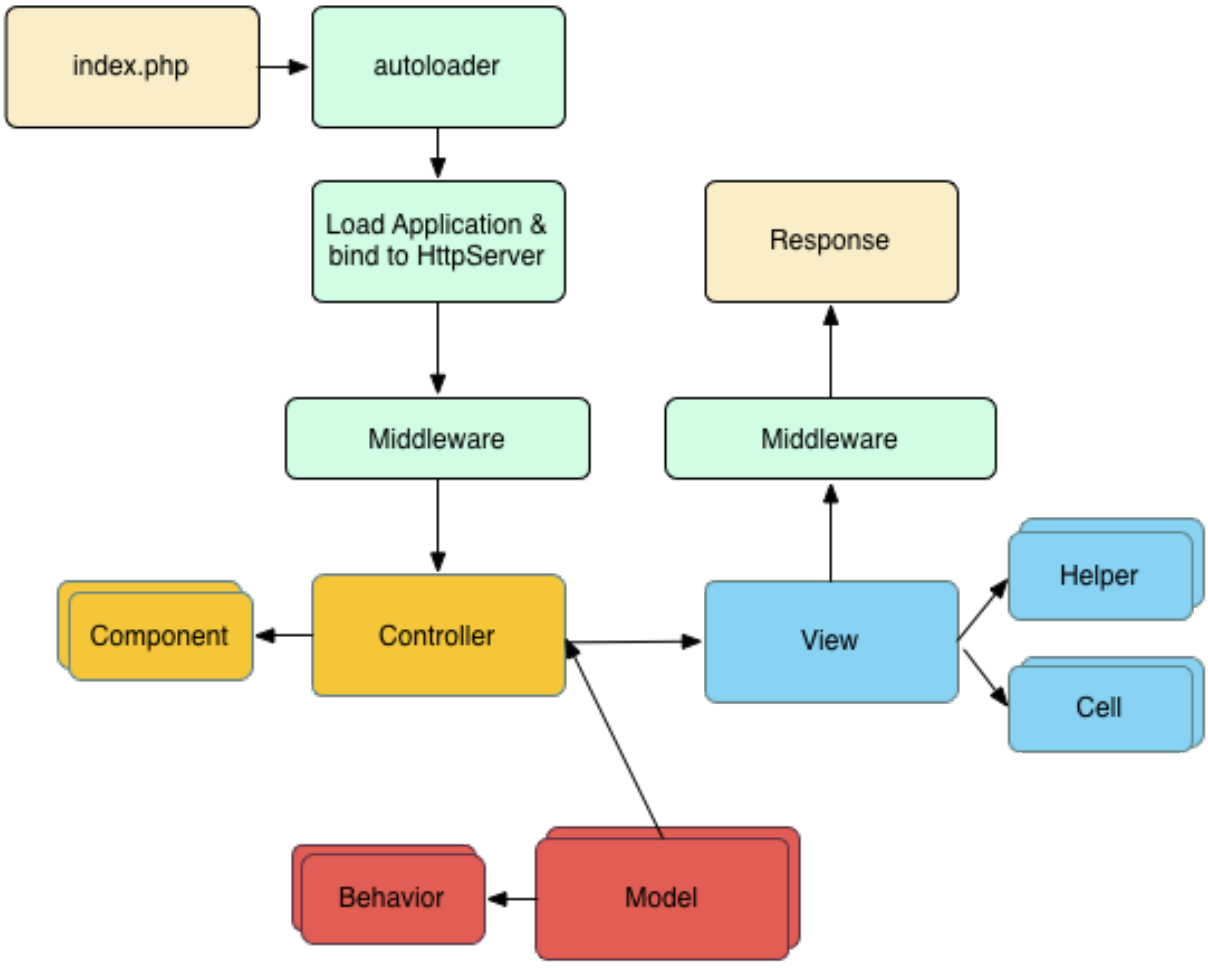
Puedes fijarte en que nunca renderizamos una vista explícitamente. Las convenciones de CakePHP se harán cargo de seleccionar la vista correcta y de renderizarla con los datos que preparemos con `set()`.

Ciclo de una petición CakePHP

Ahora que te has familiarizado con las diferentes capas en CakePHP, revisemos como funciona el ciclo de una petición:

El ciclo de petición típico de CakePHP comienza con un usuario solicitando una página o recurso en tu aplicación. A un alto nivel cada petición sigue los siguientes pasos:

1. Las reglas de rescritura del servidor web envían la petición a **webroot/index.php**.
2. Tu aplicación es cargada y ligada a un `HttpServer`.
3. Se inicializa el `midleware` de tu aplicación.
4. Una petición y respuesta son precesadas a través del `Middleware PSR-7` que tu aplicación utiliza. Normalmente esto incluye la captura de errores y enrutamiento.
5. Si no recibe ninguna respuesta del `middleware` y la petición contiene información de enrutamiento, se selecciona un controlador y una acción.
6. La acción del controlador es ejecutada y el controlador interactúa con los Modelos y Componentes necesarios.
7. El controlador delega la creación de la respuesta a la Vista para generar la salida a partir de los datos del modelo.
8. La vista utiliza `Helpers` y `Cells` para generar el cuerpo y las cabeceras de la respuesta.
9. La respuesta es devuelta a través del /controllers/middleware.
10. El `HttpServer` envía la respuesta al servidor web.


index.php
autoloader
Load Application & bind to HttpServer
Middleware
Controller
Component
Behavior
Model
View
Helper
Cell
Middleware
Response

Esto es solo el comienzo

Ojalá este repaso rápido haya despertado tu curiosidad. Otras funcionalidades geniales de CakePHP son:

- Un *framework para caché* que se integra con Memcached, Redis y otros métodos de caché.
- Poderosas herramientas de generación de código para que puedas comenzar inmediatamente.
- *Framework para la ejecución de pruebas integrado* para que puedas asegurarte de que tu código funciona perfectamente.

Los siguientes pasos obvios son *descargar CakePHP* y leer el *tutorial y crear algo asombroso*.

Lecturas complementarias

Donde obtener ayuda

La página oficial de CakePHP

https://cakephp.org

La página oficial de CakePHP es siempre un gran lugar para visitar. Proporciona enlaces a las herramientas más utilizadas por desarrolladores, `screencasts`, oportunidades para hacer una donación y descargas.

El Cookbook

https://book.cakephp.org

Este manual probablemente debería ser el primer lugar al que debas acudir para obtener respuestas. Como muchos otros proyectos de código libre, nuevos colaborades se unen regularmente. Intenta encontrar por ti mismo las respuestas a tus preguntas primero, puede que así tardes más en encontrar las respuestas pero permanecerán durante más tiempo - y además aliviarás nuestra carga de soporte. Tanto el manual como la API tienen una versión online.

La Bakery

https://bakery.cakephp.org

La «panadería» (`bakery`) de CakePHP es un lugar de intercambio para todo lo relacionado con CakePHP. Consúltala para tutoriales, casos de estudio y ejemplos de código. Cuando estés familiarizado con CakePHP, accede y comparte tus conocimientos con la comunidad y gana fortuna y fama de forma instantánea.

La API

https://api.cakephp.org/

Directo al punto y directo para desarrolladores del núcleo de CakePHP, la API (`Application Programming Interface`) es la documentación más completa para todos los detalles esenciales del funcionamiento interno del `framework`. Es referencia directa al código, asi que trae tu sombrero de hélice.

Los casos de prueba

Si crees que la información proporcionada en la API no es suficiente, comprueba el código de los casos de prueba proporcionados con CakePHP. Pueden servirte como ejemplos prácticos de funciones y datos de una clase.

```
tests/TestCase/
```

El canal IRC

Canales IRC en irc.freenode.net:

- #cakephp – Discusión general
- #cakephp-docs – Documentación
- #cakephp-bakery – Bakery
- #cakephp-fr – Canal francés.

Si estás atascado, péganos un grito en el canal IRC de CakePHP. Alguién del equipo de desarrollo[4] está normalmente, especialmente durante las horas de día para usuarios de América del Norte y del Sur. Estaremos encantados de escucharte, tanto si necesitas ayuda como si quieres encontrar usuarios en tu zona o si quieres donar tu nuevo coche deportivo de marca.

Foro oficial de CakePHP

Foro oficial de CakePHP[5]

Nuestro foro oficial donde puedes pedir ayuda, sugerir ideas y conversar sobre CakePHP. Es un lugar perfecto para encontrar rápidamente respuestas y ayudar a otros. Únete a la familia CakePHP registrándote.

Stackoverflow

https://stackoverflow.com/[6]

Etiqueta tus preguntas con `cakephp` y la versión específica que utilizas para permitir encontrar a los usuarios de stackoverflow tus preguntas.

Donde encontrar ayuda en tu idioma

Portugúes de Brasil

- Comunidad brasileña de CakePHP[7]

[4] https://cakephp.org/team
[5] https://discourse.cakephp.org
[6] https://stackoverflow.com/questions/tagged/cakephp/
[7] https://cakephp-br.org

Danés

- Canal danés de CakePHP en Slack[8]

Francés

- Comunidad francesa de CakePHP[9]

Alemán

- Canal alemán de CakePHP en Slack[10]
- Grupo alemán de CakePHP en Facebook[11]

Iraní

- Comunidad iraní de CakePHP[12]

Holandés

- Canal holandés de CakePHP en Slack[13]

Japonés

- Canal japonés de CakePHP en Slack[14]
- Grupo japonés de CakePHP en Facebook[15]

Portugués

- Grupo portugés de CakePHP en Google[16]

[8] https://cakesf.slack.com/messages/denmark/
[9] https://cakephp-fr.org
[10] https://cakesf.slack.com/messages/german/
[11] https://www.facebook.com/groups/146324018754907/
[12] https://cakephp.ir
[13] https://cakesf.slack.com/messages/netherlands/
[14] https://cakesf.slack.com/messages/japanese/
[15] https://www.facebook.com/groups/304490963004377/
[16] https://groups.google.com/group/cakephp-pt

Español

- Canal español de CakePHP en Slack[17]
- Canal IRC de CakePHP en español
- Grupo español de CakePHP en Google[18]

Convenciones CakePHP

Somos muy fans de la convención por encima de la configuración. A pesar de que toma algo de tiempo aprender las convenciones de CakePHP, ahorrarás tiempo a la larga. Siguiendo las convenciones obtendrás funcionalidades gratuitas y te liberarás de la pesadilla de mantener archivos de configuración. Las convenciones también hacen que el desarrollo sea uniforme, permitiendo a otros desarrolladores intervenir y ayudar facilmente.

Convenciones de Controlador

Los nombres de las clases Controlador son en plurar, en formato `CamelCase`, y finalizan con `Controller`. Ejemplos de nombres son: `UsuariosController` y `CategoriasArticulosController`.

Los métodos publicos de los Controladores a menudo se exponen como "acciones" accesibles a través de un navegador web. Por ejemplo, `/users/view` mapea al método `view()` de `UsersController` sin tener que hacer nada en el enrutamiento de la aplicación. Los métodos protegidos o privados no son accesibles con el enrutamiento.

Consideraciones URL para los nombres de Controladores

Como acabas de ver, los controladores de una sola palabra mapean a una dirección URL en minúscula. Por ejemplo: a `UsuariosController` (que debería estar definido en **UsuariosController.php**) se puede acceder desde http://example.com/usuarios.

Aunque puedes enrutar controladores de múltiples palabaras de la forma que desees, la convención es que tus URLs separen las palabras con guiones utilizando la clase `DashedRoute`, de este modo `/categorias-articulos/ver-todas` es la forma correcta para acceder a la acción `CategoriasArticulosController::verTodas()`.

Cuando creas enlaces utilizando `this->Html->link()` puedes utilizar las siguientes convenciones para el array url:

```
$this->Html->link('titulo-enlace', [
    'prefix' => 'MiPrefijo' // CamelCase
    'plugin' => 'MiPlugin', // CamelCase
    'controller' => 'NombreControlador', // CamelCase
    'action' => 'nombreAccion' // camelBack
]
```

Para más información sobre URLs de CakePHP y el manejo de sus parámetros puedes consultar *Connecting Routes*.

[17] https://cakesf.slack.com/messages/spanish/
[18] https://groups.google.com/group/cakephp-esp

Convenciones de nombre de clase y archivo

En general los nombres de los archivos coinciden con los nombres de las clases y siguen los estándares PSR-0 o PSR-4 para cargarse automáticamente. Los siguientes son ejemplos de nombres de clases y de sus archivos:

- La clase Controlador `LatestArticlesController` debería estar en un archivo llamado **LatestArticlesController.php**
- La clase Componente `MyHandyComponent` debería estar en un archivo llamado **MyHandyComponent.php**
- La clase Tabla `OptionValuesTable` debería estar en un archivo llamado **OptionValuesTable.php**.
- La clase Entidad `OptionValue` debería estar en un archivo llamado **OptionValue.php**.
- La clase Behavior `EspeciallyFunkableBehavior` debería estar en un archivo llamado **EspeciallyFunkableBehavior.php**
- La clase Vista `SuperSimpleView` debería estar en un archivo llamado **SuperSimpleView.php**
- La clase Helper `BestEverHelper` debería estar en un archivo llamado **BestEverHelper.php**

Cada archivo deberá estar ubicado en la carpeta/namespace correcta dentro de tu carpeta de tu aplicación.

Convenciones de Modelo y Base de datos

Los nombres de las clases `table` son en plural, `CamelCase` y terminan en `Table`. Algunos ejemplos de convención de nombres son: `UsersTable`, `ArticleCategoriesTable` y `UserFavoritePagesTable`.

Los nombres de las tablas correspondientes a los modelos de CakePHP son en plural y con "_". Los nombres de las tablas para los modelos arriba mencionados serían `users`, `article_categories` y `user_favorite_pages` respectivamente.

La convención es utilizar palabras en inglés para los nombres de las tablas y de las columnas. Si utilizas otro idioma CakePHP puede que no sea capaz de procesar correctamente las conversiones (de singular a plural y viceversa). Si necesitas añadir reglas para tu idioma para algunas palabras, puedes utilizar la clase *`Cake\Utility\Inflector`*. Además de definir tus reglas de conversión personalizadas, esta clase te permite comprobar que CakePHP comprenda tu sintaxis personalizada para palabras en plural y singular. Mira la documentación sobre *Inflector* para más información.

Los nombres de campos con dos o más palabras se escriben con "_", por ejemplo: `first_name`.

Las claves foráneas en relaciones `1-n` (`hasMany`) y `1-1` (`belongsTo/hasOne`) son reconocidas por defecto mediante el nombre (en singular) de la tabla relacionada seguido de `_id`. De este modo si `Users` tiene varios `Articles` (relación `hasMany`), la tabla `articles` se relacionará con la tabla `users` a través de la clave foránea `user_id`. Para una tabla como `article_categories` cuyo nombre está formado por varias palabras, la clave foránea sería `article_category_id`.

Las tablas de unión, usadas en las relaciones `n-n` (`BelongsToMany`) entre modelos, deberían ser nombradas después de las tablas que unirán y en orden alfabético (`articles_tags` en lugar de `tags_articles`).

Además de utilizar claves auto-incrementales como claves primarias, también puedes utilizar columnas UUID. CakePHP creará un único UUID de 36 caracteres (*`Cake\Utility\Text::uuid()`*) cada vez que guardes un nuevo registro usando el método `Table::save()` .

Convenciones de Vistas

Los archivos de las plantillas de vistas son nombrados según las funciones de controlador que las muestran empleando "_". La función `viewAll()` de la clase `ArticlesController` mostrará la vista **templates/Articles/view_all.php**.

El patrón base es **templates/Controller/nombre_funcion.php**.

Nombrando los elementos de tu aplicación empleando las convenciones de CakePHP ganarás funcionalidad sin los fastidios y ataduras de mantenimiento de la configuración.

Un último ejemplo que enlaza todas las convenciones:

- Tabla de base de datos: «articles»
- Clase Tabla: `ArticlesTable`, ubicada en **src/Model/Table/ArticlesTable.php**
- Clase Entidad: `Article`, ubicada en **src/Model/Entity/Article.php**
- Clase Controlador: `ArticlesController`, ubicada en **src/Controller/ArticlesController.php**
- Plantilla vista, ubicada en **templates/Articles/index.php**

Usando estas convenciones CakePHP redirige una petición a http://example.com/articles/ a una llamada a la función `index()` de la clase ArticlesController, donde el modelo `Article` está disponible automáticamente (y enlazada, automáticamente también, a la tabla `articles` en la base de datos) y renderiza un archivo. Ninguna de estas relaciones han sido configuradas de ningún modo salvo creando clases y archivos que has tenido que crear de todas formas.

Ahora que te has introducido en los fundamentos de CakePHP. puedes tratar de realizar el tutorial *Tutorial Bookmarker (Favoritos)* para ver como las cosas encajan juntas.

CakePHP Folder Structure

Después de haber descargado y extraido la aplicación CakePHP, estos son los archivos y directorios que podrás ver:

- bin
- config
- logs
- plugins
- src
- tests
- tmp
- vendor
- webroot
- .htaccess
- composer.json
- index.php
- README.md

Notarás unos cuantos directorios de primer nivel:

- La carpeta *bin* contiene los ejecutables por consola de Cake.
- La carpeta *config* contiene los documentos de *Configuration* que utiliza CakePHP. Detalles de la conexión a la Base de Datos, bootstrapping, arhivos de configuración del core y otros, serán almacenados aquí.

- La carpeta *plugins* es donde se almacenan los *Plugins* que utiliza tu aplicación.
- La carpeta de *logs* contiene normalmente tus archivos de log, dependiendo de tu configuración de log.
- La carpeta *src* será donde tu crearás tu mágia: es donde se almacenarán los archivos de tu aplicación.
- La carpeta *tests* será donde pondrás los test para tu aplicación.
- La carpeta *tmp* es donde CakePHP almacenará temporalmente la información. La información actual que almacenará dependerá de cómo se configure CakePHP, pero esta carpeta es normalmente utilizada para almacenar descripciones de modelos y a veces información de sesión.
- La carpeta *vendor* es donde CakePHP y otras dependencias de la aplicación serán instaladas. Comprométete a **no** editar los archivos de esta carpeta. No podremos ayudarte si modificas el core.
- El directorio *webroot* es la raíz de los documentos públicos de tu aplicación. Contiene todos los archivos que quieres que sean accesibles públicamente.

Asegúrate de que las carpetas *tmp* y *logs* existen y permiten escritura, en caso contrario el rendimiento de tu aplicación se verá gravemente perjudicado. En modo debug, CakePHP te avisará si este no es el caso.

La carpeta src

La carpeta *src* de CakePHP es donde tú harás la mayor parte del desarrollo de tu aplicación. Observemos más detenidamente dentro de la carpeta *src*.

Console
: Contiene los comandos de consola y las tareas de consola de tu aplicación. Para más información mirar *Shells, Tasks & Console Tools*.

Controller
: Contiene los controladores de tu aplicación y sus componentes.

Locale
: Almacena los ficheros de string para la internacionalización.

Model
: Contiene las tablas, entidades y funcionamiento de tu aplicación.

View
: Las clases de presentación se ubican aquí: cells, helpers y templates.

Template
: Los archivos de presentación se almacenan aquí: elementos, páginas de error, layouts, y templates.

CAPÍTULO 2

Guía de inicio rápido

La mejor forma de experimentar y aprender CakePHP es sentarse y construir algo.

Para empezar crearemos una sencilla aplicación para guardar favoritos.

Tutorial Bookmarker (Favoritos)

Este tutorial te guiará en la creación de una aplicación sencilla para el guardado de favoritos (Bookmaker).

Para comenzar instalaremos CakePHP creando nuestra base de datos y utilizaremos las herramientas que CakePHP provee para realizar nuestra aplicación rápidamente.

Esto es lo que necesitarás:

1. Un servidor de base de datos. Nosotros utilizaremos MySQL en este tutorial. Necesitarás tener los conocimientos suficientes de SQL para crear una base de datos; CakePHP tomará las riendas desde ahí. Al utilizar MySQL asegúrate de que tienes habilitado `pdo_mysql` en PHP.
2. Conocimientos básicos de PHP.

Antes de empezar deberías de asegurarte de que tienes actualizada la versión de PHP:

```
php -v
```

Deberías tener instalado PHP 7.4 (CLI) o superior. La versión PHP de tu servidor web deberá ser 7.4 o superior y lo ideal es que coincida con la versión de la interfaz de línea de comandos (CLI) de PHP. Si quieres ver la aplicación ya finalizada puedes consultar cakephp/bookmarker[19].

Empecemos!

[19] https://github.com/cakephp/bookmarker-tutorial

Instalar CakePHP

La forma más sencilla de instalar CakePHP es utilizando Composer, una manera sencilla de instalar CakePHP desde tu terminal o prompt de línea de comandos.

Primero necesitarás descargar e instalar Composer si aún no lo tienes. Si ya tienes instalado cURL es tan sencillo como ejecutar:

```
curl -s https://getcomposer.org/installer | php
```

O puedes descargar `composer.phar` desde la Página web de Composer[20].

Después sencillamente escribe la siguiente línea en tu terminal desde tu directorio de instalación para instalar el esqueleto de la aplicación CakePHP en el directorio **bookmarker**:

```
php composer.phar create-project --prefer-dist cakephp/app:4.* bookmarker
```

Si descargaste y ejecutaste el Instalador Windows de Composer[21], entonces escribe la siguiente línea en tu terminal desde tu directorio de instalación (ie. C:\wamp\www\dev\cakephp3):

```
composer self-update && composer create-project --prefer-dist cakephp/app:4.* bookmarker
```

La ventaja de utilizar Composer es que automáticamente realizará algunas tareas importantes como configurar correctamente el archivo de permisos y crear tu archivo **config/app.php**.

Hay otras formas de instalar CakePHP. Si no puedes o no quieres utilizar Composer comprueba la sección *Instalación*.

Sin importar como hayas descargado e instalado CakePHP, una vez hayas finalizado, tu directorio de instalación debería ser algo como:

```
/bookmarker
    /bin
    /config
    /logs
    /plugins
    /src
    /tests
    /tmp
    /vendor
    /webroot
    .editorconfig
    .gitignore
    .htaccess
    .travis.yml
    composer.json
    index.php
    phpunit.xml.dist
    README.md
```

Ahora podría ser un buen momento para que aprendas un poco sobre como funciona la estructura de directorios de CakePHP: *CakePHP Folder Structure*.

[20] https://getcomposer.org/download/
[21] https://getcomposer.org/Composer-Setup.exe

Comprobar la instalación

Podemos comprobar rápidamente que nuestra instalación ha sido correcta accediendo a la página principal que se crea por defecto.

Pero antes necesitarás inicializar el servidor de desarrollo:

```
bin/cake server
```

Nota: Para Windows introduce el comando `bin\cake server` (fíjate en la \ ).

Esto arrancará el servidor integrado en el puerto 8765. Accede a **http://localhost:8765** a través de tu navegador para ver la página de bienvenida. Todos los items deberán estar marcados como correctos para que CakePHP pueda conectarse a tu base de datos. Si no, puede que necesites instalar extensiones adicionales de PHP, o dar permisos de directorio.

Crear la base de datos

Continuamos, creemos ahora la base de datos para nuestra aplicación de favoritos.

Si aún no lo has hecho, crea una base de datos vacía para usar en este tutorial con el nombre que tu quieras, e.g. `cake_bookmarks`.

Puedes ejecutar la siguiente sentencia SQL para crear las tablas necesarias:

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    created DATETIME,
    modified DATETIME
);

CREATE TABLE bookmarks (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    title VARCHAR(50),
    description TEXT,
    url TEXT,
    created DATETIME,
    modified DATETIME,
    FOREIGN KEY user_key (user_id) REFERENCES users(id)
);

CREATE TABLE tags (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255),
    created DATETIME,
    modified DATETIME,
    UNIQUE KEY (title)
);

CREATE TABLE bookmarks_tags (
    bookmark_id INT NOT NULL,
```

(continué en la próxima página)

(proviene de la página anterior)

```
    tag_id INT NOT NULL,
    PRIMARY KEY (bookmark_id, tag_id),
    FOREIGN KEY tag_key(tag_id) REFERENCES tags(id),
    FOREIGN KEY bookmark_key(bookmark_id) REFERENCES bookmarks(id)
);
```

Puedes ver que la tabla `bookmarks_tags` utiliza una clave primaria compuesta. CakePHP soporta claves primarias compuestas en casi cualquier lado, haciendo más fácil construir aplicaciones multi-anidadas.

Los nombres de las tablas y columnas que hemos utilizado no son aleatorios. Utilizando las *convenciones de nombres* podemos hacer mejor uso de CakePHP y evitar tener que configurar el framework.

CakePHP es lo suficientemente flexible para acomodarse incluso a esquemas inconsistentes de bases de datos heredados, pero siguiendo las convenciones ahorrarás tiempo.

Configuración de la base de datos

Siguiente, indiquémosle a CakePHP donde está nuestra base de datos y como conectarse a ella. Para la mayoría de las veces esta será la primera y última vez que necesitarás configurar algo.

La configuración debería ser bastante sencilla: sólo cambia los valores del array `Datasources.default` en el archivo **config/app.php** por aquellos que apliquen a tu instalación. Un ejemplo de array de configuración completado puede lucir así:

```
return [
    // More configuration above.
    'Datasources' => [
        'default' => [
            'className' => 'Cake\Database\Connection',
            'driver' => 'Cake\Database\Driver\Mysql',
            'persistent' => false,
            'host' => 'localhost',
            'username' => 'cakephp',
            'password' => 'AngelF00dC4k3~',
            'database' => 'cake_bookmarks',
            'encoding' => 'utf8',
            'timezone' => 'UTC',
            'cacheMetadata' => true,
        ],
    ],
    // More configuration below.
];
```

Una vez hayas guardado tu archivo **config/app.php** deberías ver que la sección "CakePHP is able to connect to the database" tiene un chechmark de correcto.

Nota: Puedes encontrar una copia de la configuración por defecto de CakePHP en **config/app.default.php**.

Crear el esqueleto del código

Gracias a que nuestra base de datos sigue las convenciones de CakePHP podemos utilizar la consola de bake de la aplicación para crear rápidamente una aplicación básica.

En tu línea de comandos ejecuta las siguientes instrucciones:

```
// En Windows necesitarás utilizar bin\cake.
bin/cake bake all users
bin/cake bake all bookmarks
bin/cake bake all tags
```

Esto creará los controladores, modelos, vistas, sus correspondientes casos de prueba y accesorios para nuestros recursos de users, bookmarks y tags.

Si detuviste tu servidor reinícialo.

Vete a **http://localhost:8765/bookmarks**, deberías poder ver una básica pero funcional aplicación provista de acceso a las tablas de tu base de datos.

Una vez estés en la lista de bookmarks añade unos cuantos usuarios (users), favoritos (bookmarks) y etiquetas (tags)

Nota: Si ves una página de error Not Found (404) comprueba que el módulo de Apache mod_rewrite está cargado.

Añadir encriptación (hashing) a la contraseña

Cuando creaste tus usuarios (visitando **http://localhost:8765/users**) probablemente te darías cuenta de que las contraseñas (password) se almacenaron en texto plano. Algo muy malo desde un punto de vista de seguridad, así que arreglémoslo.

Éste es también un buen momento para hablar de la capa de modelo en CakePHP.

En CakePHP separamos los métodos que operan con una colección de objetos y los que lo hacen con un único objeto en diferentes clases.

Los métodos que operan con una coleccion de entidades van en la clase `Table`, mientras que los que lo hacen con una sola van en la clase `Entity`.

Por ejemplo: el encriptado de una contraseña se hace en un registro individual, por lo que implementaremos este comportamiento en el objeto Entity.

Ya que lo que queremos es encriptar la contraseña cada vez que la introduzcamos en la base de datos utilizaremos un método mutador/setter.

CakePHP utilizará la convención para métodos setter cada vez que una propiedad se introducida en una de tus entidades.

Añadamos un setter para la contraseña añadiendo el siguiente código en **src/Model/Entity/User.php**:

```
namespace App\Model\Entity;

use Cake\Auth\DefaultPasswordHasher; //include this line
use Cake\ORM\Entity;

class User extends Entity
{

    // Code from bake.
```

(continué en la próxima página)

(proviene de la página anterior)

```
    protected function _setPassword($value)
    {
        $hasher = new DefaultPasswordHasher();

        return $hasher->hash($value);
    }
}
```

Ahora actualiza uno de los usuarios que creaste antes, si cambias su contraseña deberías ver una contraseña encriptada en vez del valor original en la lista de usuarios o en su página de View.

CakePHP encripta contraseñas con bcrypt[22] por defecto. Puedes usar también sha1 o md5 si estás trabajando con bases de datos ya existentes.

Nota: Si la contraseña no se ha encriptado asegúrate de que has usado el mismo estilo de escritura que el del atributo password de la clase cuando nombraste la función setter.

Obtener bookmarks con un tag específico

Ahora que estamos almacenando contraseñas con seguridad podemos añadir alguna funcionalidad interesante a nuestra aplicación.

Cuando acumulas una colección de favoritos es útil poder buscarlos a través de etiquetas.

Implementemos una ruta, una acción de controlador y un método finder para buscar bookmarks mediante etiquetas.

Idealmente tendríamos una URL como **http://localhost:8765/bookmarks/tagged/funny/cat/gifs** que nos permitiría encontrar todos los bookmarks que tienen las etiquetas “funny”, “cat” o “gifs”.

Antes de que podamos implementarlo añadiremos una nueva ruta.

Modifica tu **config/routes.php** para que se vea como ésto:

```
<?php
use Cake\Routing\Route\DashedRoute;
use Cake\Routing\Router;

Router::defaultRouteClass(DashedRoute::class);

// Nueva ruta que añadimos para nuestra acción tagged
// The trailing `*` tells CakePHP that this action has
// passed parameters.
Router::scope(
    '/bookmarks',
    ['controller' => 'Bookmarks'],
    function ($routes) {
        $routes->connect('/tagged/*', ['action' => 'tags']);
    }
);
```

(continué en la próxima página)

[22] https://codahale.com/how-to-safely-store-a-password/

(proviene de la página anterior)

```
Router::scope('/', function ($routes) {
    // Connect the default home and /pages/* routes.
    $routes->connect('/', [
        'controller' => 'Pages',
        'action' => 'display', 'home'
    ]);
    $routes->connect('/pages/*', [
        'controller' => 'Pages',
        'action' => 'display'
    ]);

    // Connect the conventions based default routes.
    $routes->fallbacks();
});
```

Lo cual define una nueva "ruta" que conecta el path **/bookmarks/tagged/** a `BookmarksController::tags()`.

Con la definición de rutas puedes separar como se ven tus URLs de como se implementan. Si visitamos **http://localhost:8765/bookmarks/tagged**, podremos ver una página de error bastante útil de CakePHP informando que no existe la acción del controlador.

Implementemos ahora ese método.

En **src/Controller/BookmarksController.php** añade:

```
public function tags()
{
    // The 'pass' key is provided by CakePHP and contains all
    // the passed URL path segments in the request.
    $tags = $this->request->getParam('pass');

    // Use the BookmarksTable to find tagged bookmarks.
    $bookmarks = $this->Bookmarks->find('tagged', [
        'tags' => $tags
    ]);

    // Pass variables into the view template context.
    $this->set([
        'bookmarks' => $bookmarks,
        'tags' => $tags
    ]);
}
```

Para acceder a otras partes del request consulta *Request*.

Crear el método finder

En CakePHP nos gusta mantener las acciones de los controladores sencillas y poner la mayoría de la lógica de la aplicación en los modelos. Si visitas ahora la URL **/bookmarks/tagged** verás un error de que el método `findTagged()` no ha sido implementado todavía, asi que hagámoslo.

En **src/Model/Table/BookmarksTable.php** añade lo siguiente:

```
// El argumento $query es una instancia de query.
// El array $options contendrá las opciones de 'tags' que pasemos
// para encontrar'tagged') en nuestra acción del controlador.
public function findTagged(Query $query, array $options)
{
    $bookmarks = $this->find()
        ->select(['id', 'url', 'title', 'description']);

    if (empty($options['tags'])) {
        $bookmarks
            ->leftJoinWith('Tags')
            ->where(['Tags.title IS' => null]);
    } else {
        $bookmarks
            ->innerJoinWith('Tags')
            ->where(['Tags.title IN ' => $options['tags']]);
    }

    return $bookmarks->group(['Bookmarks.id']);
}
```

Acabamos de implementar un *método finder personalizado*.

Esto es un concepto muy poderoso en CakePHP que te permite empaquetar queries re-utilizables.

Los métodos finder siempre reciben un objeto *Query Builder* y un array de opciones como parámetros. Estos métodos pueden manipular la query y añadir cualquier condición o criterio requerido; cuando se completan devuelven un objeto query modificado.

En nuestro método finder sacamos provecho de los métodos `distinct()` y `matching()` que nos permiten encontrar distintos ("distincts") bookmarks que tienen un tag coincidente (matching). El método `matching()` acepta una función anónima[23] que recibe un generador de consultas. Dentro del callback usaremos este generador para definir las condiciones que filtrarán bookmarks que tienen las etiquetas (tags) especificadas.

Crear la vista

Ahora si visitas la URL **/bookmarks/tagged**, CakePHP mostrará un error advirtiéndote de que no has creado un archivo de vista.

Siguiente paso, creemos un archivo de vista para nuestro método `tags()`.

En **templates/Bookmarks/tags.php** añade el siguiente código:

```
<h1>
    Bookmarks tagged with
    <?= $this->Text->toList(h($tags)) ?>
```

(continué en la próxima página)

[23] https://php.net/manual/es/functions.anonymous.php

(proviene de la página anterior)

```
</h1>

<section>
<?php foreach ($bookmarks as $bookmark): ?>
    <article>
        <!-- Use the HtmlHelper to create a link -->
        <h4><?= $this->Html->link($bookmark->title, $bookmark->url) ?></h4>
        <small><?= h($bookmark->url) ?></small>

        <!-- Use the TextHelper to format text -->
        <?= $this->Text->autoParagraph(h($bookmark->description)) ?>
    </article>
<?php endforeach; ?>
</section>
```

En el código de arriba utilizamos los helpers *HtmlHelper* y *TextHelper* para que asistan en la generación de nuestra salida de la vista.

También utilizamos la función de atajo `h()` para salidas de código HTML. Deberías acordarte siempre de utilizar `h()` cuando muestres datos del usuario para evitar problemas de inyección HTML.

El archivo **tags.php** que acabamos de crear sigue las convenciones de CakePHP para archivos de vistas. La convención es que el nombre del archivo sea una versión en minúsculas y subrayados del nombre de la acción del controlador.

Puedes observar que hemos podido usar las variables `$tags` y `$bookmarks` en nuestra vista.

Cuando utilizamos el método `set()` en nuestro controlador especificamos variables para enviarlas a la vista. Ésta hará disponibles todas las variables que se le pasen como variables locales.

Ahora deberías poder visitar la URL **/bookmarks/tagged/funny** y ver todos los favoritos etiquetados con “funny”.

Hasta aquí hemos creado una aplicación básica para manejar favoritos (bookmarks), etiquetas (tags) y usuarios (users). Sin embargo todo el mundo puede ver las etiquetas de los demás. En el siguiente capítulo implementaremos autenticación y restringiremos el uso de etiquetas únicamente a aquellas que pertenezcan al usuario actual.

Ahora ve a *Tutorial Bookmarker (Favoritos) - Parte 2* para continuar construyendo tu apliación o sumérgete en la documentación para aprender más sobre que puede hacer CakePHP por ti.

Tutorial Bookmarker (Favoritos) - Parte 2

Tras realizar *la primera parte de este tutorial* deberías tener una aplicación muy básica para guardar favoritos.

En este capítulo añadiremos la autenticación y restringiremos los favoritos (bookmarks) para que cada usuario pueda consultar o modificar solamente los suyos.

Añadir login

En CakePHP, la autenticación se maneja mediante *Componentes*.

Los componentes pueden verse como una forma de crear trozos reutilizables de código de controlador para una finalidad o idea. Además pueden engancharse al evento de ciclo de vida de los controladores e interactuar con tu aplicación de ese modo.

Para empezar añadiremos el componente *AuthComponent* a nuestra aplicación.

Como queremos que todos nuestros métodos requieran de autenticación añadimos *AuthComponent* en *AppController* del siguiente modo:

```
// En src/Controller/AppController.php
namespace App\Controller;

use Cake\Controller\Controller;

class AppController extends Controller
{
    public function initialize()
    {
        $this->loadComponent('Flash');
        $this->loadComponent('Auth', [
            'authenticate' => [
                'Form' => [
                    'fields' => [
                        'username' => 'email',
                        'password' => 'password'
                    ]
                ]
            ],
            'loginAction' => [
                'controller' => 'Users',
                'action' => 'login'
            ],
            'unauthorizedRedirect' => $this->referer() // Si no está autorizado,
                                //el usuario regresa a la página que estaba
        ]);

        // Permite ejecutar la acción display para que nuestros controladores de páginas
        // sigan funcionando.
        $this->Auth->allow(['display']);
    }
}
```

Acabamos de decirle a CakePHP que queremos cargar los compomentes `Flash` y `Auth`. Además hemos personalizado la configuración de *AuthComponent* indicando que utilice como *username* el campo *email* de la tabla *Users* de la base de datos.

Ahora si vas a cualquier *URL* serás enviado a **/users/login**, que mostrará una página de error ya que no hemos escrito el código de la función *login* todavía, así que hagámoslo ahora:

```
// En src/Controller/UsersController.php
public function login()
{
```

(continué en la próxima página)

(proviene de la página anterior)

```
    if ($this->request->is('post')) {
        $user = $this->Auth->identify();
        if ($user) {
            $this->Auth->setUser($user);

            return $this->redirect($this->Auth->redirectUrl());
        }
        $this->Flash->error('Tu usuario o contraseña es incorrecta.');
    }
}
```

Y en **templates/Users/login.php** añade lo siguiente:

```
<h1>Login</h1>
<?= $this->Form->create() ?>
<?= $this->Form->input('email') ?>
<?= $this->Form->input('password') ?>
<?= $this->Form->button('Login') ?>
<?= $this->Form->end() ?>
```

Ahora que tenemos un formulario de *login* sencillo deberíamos poder loguearnos con algún usuario que tenga contraseña encriptada.

Nota:

Si ninguno de tus usuarios tiene contraseña encriptada comenta la línea
`loadComponent('Auth')`, a continuación edita un usuario y modifica la contraseña.

Ahora deberías poder loguearte, si no es así asegúrate de que estás utilizando un usuario con contraseña encriptada.

Añadir *logout*

Ahora que la gente puede loguearse probablemente quieras añadir una forma de desloguearse también.

Otra vez en `UsersController`, añade el siguiente código:

```
public function initialize()
{
    parent::initialize();
    $this->Auth->allow(['logout']);
}

public function logout()
{
    $this->Flash->success('Ahora estás deslogueado.');

    return $this->redirect($this->Auth->logout());
}
```

Este código añade la acción `logout` como una acción pública e implementa la función.

Ahora puedes visitar `/users/logout` para desloguearte, deberías ser enviado a la página de inicio.

Habilitar registros

Si no estás logueado e intentas acceder a **/users/add** eres reenviado a la página de login. Deberíamos arreglar esto si queremos permitir que la gente se pueda registrar en nuestra aplicación.

En el controlador `UsersController` añade lo siguiente:

```
public function initialize()
{
    parent::initialize();
    // Añade logout a la lista de actiones permitidas.
    $this->Auth->allow(['logout', 'add']);
}
```

El código anterior le dice a `AuthComponent` que la acción `add()` no necesita autenticación ni autorización.

Tal vez quieras tomarte un tiempo para limpiar **Users/add.php** y eliminar los enlaces erróneos o continuar con el siguiente apartado. No vamos a crear la edición de usuarios, consulta o listado en este tutorial así que no funcionará el control de `AuthComponent` para el acceso a esas acciones del controlador.

Restringiendo el acceso a favoritos

Ahora que los usuarios pueden loguearse queremos restringir los favoritos que uno puede ver a los que creó. Esto lo haremos usando un adaptador de "authorization".

Ya que nuestro requisito es muy sencillo podremos escribir un código también muy sencillo en nuestro `BookmarksController`.

Pero antes necesitamos decirle al componente *AuthComponent* cómo va a autorizar acciones nuestra aplicación. Para ello añade en `AppController`:

```
public function isAuthorized($user)
{
    return false;
}
```

Además añade la siguiente línea a la configuración de `Auth` en tu `AppController`:

```
'authorize' => 'Controller',
```

Tú método `initialize()` debería verse así:

```
public function initialize()
{
    $this->loadComponent('Flash');
    $this->loadComponent('Auth', [
        'authorize'=> 'Controller',// línea añadida
        'authenticate' => [
            'Form' => [
                'fields' => [
                    'username' => 'email',
                    'password' => 'password'
                ]
            ]
        ],
```

(continué en la próxima página)

(proviene de la página anterior)

```
        'loginAction' => [
            'controller' => 'Users',
            'action' => 'login'
        ],
        'unauthorizedRedirect' => $this->referer()
    ]);

    // Permite ejecutar la acción display para que nuestros controladores
    // de páginas sigan funcionando.
    $this->Auth->allow(['display']);
}
```

Por defecto denegaremos el acceso siempre y concederemos los accesos donde tenga sentido.

Primero añadiremos la lógica de autorización para favoritos.

En tu `BookmarksController` añade lo siguiente:

```
public function isAuthorized($user)
{
    $action = $this->request->getParam('action');

    // Las acciones add e index están siempre permitidas.
    if (in_array($action, ['index', 'add', 'tags'])) {
        return true;
    }
    // El resto de acciones requieren un id.
    if (!$this->request->getParam('pass.0')) {
        return false;
    }

    // Comprueba que el favorito pertenezca al usuario actual.
    $id = $this->request->getParam('pass.0');
    $bookmark = $this->Bookmarks->get($id);
    if ($bookmark->user_id == $user['id']) {
        return true;
    }

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

Ahora si intentas consultar, editar o borrar un favorito que no te pertenece deberías ser redirigido a la página desde la que accediste.

Si no se muestra ningún mensaje de error añade lo siguiente a tu layout:

```
// En templates/layout/default.php
<?= $this->Flash->render() ?>
```

Deberías poder ver ahora los mensajes de error de autorización.

Arreglar lista de consulta y formularios

Mientras que *view* y *delete* están funcionando, *edit*, *add* e *index* presentan un par de problemas:

1. Cuando añades un favorito puedes elegir el usuario.
2. Cuando editas un favorito puedes elegir un usuario.
3. La página con el listado muestra favoritos de otros usuarios.

Abordemos el formulario de añadir favorito primero.

Para empezar elimina `input('user_id')` de **templates/Bookmarks/add.php**.

Con esa parte eliminada actualizaremos la acción `add()` de **src/Controller/BookmarksController.php** para que luzca así:

```
public function add()
{
    $bookmark = $this->Bookmarks->newEntity();
    if ($this->request->is('post')) {
        $bookmark = $this->Bookmarks->patchEntity($bookmark, $this->request->getData());
        $bookmark->user_id = $this->Auth->user('id');
        if ($this->Bookmarks->save($bookmark)) {
            $this->Flash->success('El favorito se ha guardado.');

            return $this->redirect(['action' => 'index']);
        }
        $this->Flash->error('El favorito podría no haberse guardado. Por favor,␣
↪inténtalo de nuevo.');
    }
    $tags = $this->Bookmarks->Tags->find('list');
    $this->set(compact('bookmark', 'tags'));
    $this->set('_serialize', ['bookmark']);
}
```

Completando la propiedad de la entidad con datos de la sesión eliminaremos cualquier posibilidad de que el usuario modifique el usuario al que pertenece el favorito. Haremos lo mismo para el formulario de edición.

Tu acción `edit()` de **src/Controller/BookmarksController.php** debería ser así:

```
public function edit($id = null)
{
    $bookmark = $this->Bookmarks->get($id, [
        'contain' => ['Tags']
    ]);
    if ($this->request->is(['patch', 'post', 'put'])) {
        $bookmark = $this->Bookmarks->patchEntity($bookmark, $this->request->getData());
        $bookmark->user_id = $this->Auth->user('id');
        if ($this->Bookmarks->save($bookmark)) {
            $this->Flash->success('El favorito se ha guardado.');

            return $this->redirect(['action' => 'index']);
        }
        $this->Flash->error('El favorito podría no haberse guardado. Por favor,␣
↪inténtalo de nuevo.');
    }
```

(continué en la próxima página)

(proviene de la página anterior)

```
    $tags = $this->Bookmarks->Tags->find('list');
    $this->set(compact('bookmark', 'tags'));
    $this->set('_serialize', ['bookmark']);
}
```

Listado consulta

Ahora solo necesitamos mostrar los favoritos del usuario actualmente logueado.

Podemos hacer eso actualizando la llamada a `paginate()`. Haz que tu método `index()` de **src/Controller/BookmarksController.php** se vea así:

```
public function index()
{
    $this->paginate = [
        'conditions' => [
            'Bookmarks.user_id' => $this->Auth->user('id'),
        ]
    ];
    $this->set('bookmarks', $this->paginate($this->Bookmarks));
    $this->set('_serialize', ['bookmarks']);
}
```

Deberíamos actualizar también el método `tags()` y el método finder relacionado, pero lo dejaremos como un ejercicio para que lo hagas por tu cuenta.

Mejorar la experiencia de etiquetado

Ahora mismo añadir nuevos tags es un proceso complicado desde que `TagsController` desautorizó todos los accesos.

En vez de permitirlos podemos mejorar la *UI* para la selección de tags utilizando un campo de texto separado por comas. Esto proporcionará una mejor experiencia para nuestros usuarios y usa algunas de las mejores características de *ORM*.

Añadir un campo calculado

Para acceder de forma sencilla a las etiquetas formateadas podemos añadir un campo virtual/calculado a la entidad.

En **src/Model/Entity/Bookmark.php** añade lo siguiente:

```
use Cake\Collection\Collection;

protected function _getTagString()
{
    if (isset($this->_fields['tag_string'])) {
        return $this->_fields['tag_string'];
    }
    if (empty($this->tags)) {
        return '';
    }
    $tags = new Collection($this->tags);
```

(continué en la próxima página)

(proviene de la página anterior)

```
    $str = $tags->reduce(function ($string, $tag) {
        return $string . $tag->title . ', ';
    }, '');

    return trim($str, ', ');
}
```

Esto nos dará acceso a la propiedad calculada `$bookmark->tag_string` que utilizaremos más adelante.

Recuerda añadir la propiedad `tag_string` a la lista `_accessible` en tu entidad para poder "guardarla" más adelante.

En **src/Model/Entity/Bookmark.php** añade `tag_string` a `$_accessible` de este modo:

```
protected $_accessible = [
    'user_id' => true,
    'title' => true,
    'description' => true,
    'url' => true,
    'user' => true,
    'tags' => true,
    'tag_string' => true,
];
```

Actualizar las vistas

Con la entidad actualizada podemos añadir un nuevo campo de entrada para nuestros tags. En **templates/Bookmarks/add.php** y **templates/Bookmarks/edit.php**, cambia el campo `tags._ids` por el siguiente:

```
echo $this->Form->input('tag_string', ['type' => 'text']);
```

Guardar el string de tags

Ahora que podemos ver los tags existentes como un string querremos guardar también esa información.

Al haber marcado `tag_string` como accesible el ORM copiará esa información del request a nuestra entidad. Podemos usar un método de gancho `beforeSave()` para parsear el *string* de etiquetas y encontrar/crear las entidades relacionadas.

Añade el siguiente código a **src/Model/Table/BookmarksTable.php**:

```
public function beforeSave($event, $entity, $options)
{
    if ($entity->tag_string) {
        $entity->tags = $this->_buildTags($entity->tag_string);
    }
}

protected function _buildTags($tagString)
{
    // Hace trim a las etiquetas
    $newTags = array_map('trim', explode(',', $tagString));
    // Elimina las etiquetas vacías
```

(continué en la próxima página)

(proviene de la página anterior)

```
    $newTags = array_filter($newTags);
    // Elimina las etiquetas duplicadas
    $newTags = array_unique($newTags);

    $out = [];
    $query = $this->Tags->find()
        ->where(['Tags.title IN' => $newTags]);

    // Elimina las etiquetas existentes de la lista de nuevas etiquetas.
    foreach ($query->extract('title') as $existing) {
        $index = array_search($existing, $newTags);
        if ($index !== false) {
            unset($newTags[$index]);
        }
    }
    // Añade las etiquetas existentes.
    foreach ($query as $tag) {
        $out[] = $tag;
    }
    // Añade las etiquetas nuevas.
    foreach ($newTags as $tag) {
        $out[] = $this->Tags->newEntity(['title' => $tag]);
    }

    return $out;
}
```

Aunque este código sea algo más complicado de lo que hemos hecho hasta ahora, nos ayudará a ver lo potente que es el *ORM* en CakePHP.

Puedes manipular los resultados de la consulta usando los métodos *Collections* y manejar escenearios en los que estás creando entidades *on the fly* con facilidad.

Para finalizar

Hemos mejorado nuestra aplicación de favoritos para manejar escenarios de autenticación y de autorización/control de acceso básicos.

Además hemos añadido algunas mejoras interesantes de experiencia de usuario sacándole provecho a *FormHelper* y al potencial de *ORM*.

Gracias por tomarte tu tiempo para explorar CakePHP. Ahora puedes realizar el tutorial *Tutorial Blog*, aprender más sobre *Acceso a la base de datos & ORM*, o puedes leer detenidamente los /topics.

CAPÍTULO 3

4.0 Migration Guide

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

Por favor, siéntase libre de enviarnos un pull request en Github[24] 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.

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

CAPÍTULO 4

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.

Tutorial Gestor de Contenidos

Este tutorial lo guiará a través de la creación de un CMS (Sistema de Gestión de Contenidos) simple. Para empezar, instalaremos CakePHP, creando nuestra base de datos y construyendo una gestión simple de artículos.

Esto es lo que se necesitará:

1. Un servidor de base de datos. Vamos a utilizar el servidor MySQL en este tutorial. Necesitará saber lo suficiente sobre SQL para crear una base de datos y ejecutar fragmentos SQL del tutorial. CakePHP se encargará de construir todas las consultas que su aplicación necesita. Como estamos usando MySQL, también asegúrese de tener `pdo_mysql` habilitado en PHP.
2. Conocimientos básicos de PHP.

Antes de comenzar, debe asegurarse de tener una versión de PHP actualizada:

```
php -v
```

Al menos debería haber instalado PHP 7.4 (CLI) o superior. La versión PHP de su servidor web también debe ser de 7.4 o superior, y debería ser la misma versión que su interfaz de línea de comando (CLI) de PHP.

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

Obteniendo CakePHP

La forma más fácil de instalar CakePHP es usar Composer. Composer es una manera simple de instalar CakePHP desde su terminal o línea de comandos. Primero, necesita descargar e instalar Composer si aún no lo ha hecho. Si tiene cURL instalado, es tan fácil como ejecutar lo siguiente:

```
curl -s https://getcomposer.org/installer | php
```

O, puede descargar `composer.phar` desde el sitio web de Composer[27].

Luego simplemente escriba la siguiente línea en su terminal desde el directorio de instalación para instalar el esqueleto de la aplicación CakePHP en la carpeta **cms** del directorio de trabajo actual:

```
php composer.phar create-project --prefer-dist cakephp/app:4.* cms
```

Si ha descargado y ejecutado el Instalador de Composer de Windows[28], entonces, escriba la siguiente línea en el terminal desde el directorio de instalación (ej. C:\wamp\www\dev):

```
composer self-update && composer create-project --prefer-dist cakephp/app:4.* cms
```

La ventaja de usar Composer es que completará automáticamente algunas tareas de configuración importantes, como establecer los permisos de archivo correctos y crear el archivo **config/app.php** por usted.

Hay otras formas de instalar CakePHP. Si no puede o no quiere usar Composer, consulte la sección *Instalación*.

Independientemente de cómo haya descargado e instalado CakePHP, una vez que la configuración es completada, la disposición de su directorio debería ser similar a la siguiente:

```
/cms
 /bin
 /config
 /logs
 /plugins
 /src
 /tests
 /tmp
 /vendor
 /webroot
 .editorconfig
 .gitignore
 .htaccess
 .travis.yml
 .composer.json
 index.php
 phpunit.xml.dist
 README.md
```

Ahora podría ser un buen momento para aprender un poco sobre cómo funciona la estructura de directorios de CakePHP: consulte la sección *CakePHP Folder Structure*.

Si se pierde durante este tutorial, puede ver el resultado final en GitHub[29].

[27] https://getcomposer.org/download/
[28] https://getcomposer.org/Composer-Setup.exe
[29] https://github.com/cakephp/cms-tutorial

Comprobando nuestra instalación

PPodemos verificar rápidamente que nuestra instalación es correcta, verificando la página de inicio predeterminada. Antes de que pueda hacer eso, deberá iniciar el servidor de desarrollo:

```
cd /path/to/our/app
bin/cake server
```

Nota: Para Windows, el comando debe ser `bin\cake server` (tenga en cuenta la barra invertida).

Esto iniciará el servidor web incorporado de PHP en el puerto 8765. Abra **http://localhost:8765** en su navegador web para ver la página de bienvenida. Todos las viñetas deben ser sombreros de chef verdes indicando que CakePHP puede conectarse a De lo contrario, es posible que deba instalar extensiones adicionales de PHP o establecer permisos de directorio.

A continuación, crearemos nuestra *Base de datos y crearemos nuestro primer modelo*.

Tutorial CMS - Creando la Base de Datos

Ahora que tenemos CakePHP instalado, configuremos la base de datos para nuestro CMS. Si aún no lo ha hecho, cree una base de datos vacía para usar en este tutorial, con un nombre de su elección, p. ej. `cake_cms`. Si está utilizando MySQL/MariaDB, puede ejecutar el siguiente SQL para crear las tablas necesarias:

```
USE cake_cms;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    created DATETIME,
    modified DATETIME
);

CREATE TABLE articles (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    title VARCHAR(255) NOT NULL,
    slug VARCHAR(191) NOT NULL,
    body TEXT,
    published BOOLEAN DEFAULT FALSE,
    created DATETIME,
    modified DATETIME,
    UNIQUE KEY (slug),
    FOREIGN KEY user_key (user_id) REFERENCES users(id)
) CHARSET=utf8mb4;

CREATE TABLE tags (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(191),
    created DATETIME,
    modified DATETIME,
```

(continué en la próxima página)

(proviene de la página anterior)

```
    UNIQUE KEY (title)
) CHARSET=utf8mb4;

CREATE TABLE articles_tags (
    article_id INT NOT NULL,
    tag_id INT NOT NULL,
    PRIMARY KEY (article_id, tag_id),
    FOREIGN KEY tag_key(tag_id) REFERENCES tags(id),
    FOREIGN KEY article_key(article_id) REFERENCES articles(id)
);

INSERT INTO users (email, password, created, modified)
VALUES
('cakephp@example.com', 'secret', NOW(), NOW());

INSERT INTO articles (user_id, title, slug, body, published, created, modified)
VALUES
(1, 'First Post', 'first-post', 'This is the first post.', 1, NOW(), NOW());
```

Si está utilizando PostgreSQL, conéctese a la base de datos `cake_cms` y ejecute el siguiente SQL en su lugar:

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    email VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    created TIMESTAMP,
    modified TIMESTAMP
);

CREATE TABLE articles (
    id SERIAL PRIMARY KEY,
    user_id INT NOT NULL,
    title VARCHAR(255) NOT NULL,
    slug VARCHAR(191) NOT NULL,
    body TEXT,
    published BOOLEAN DEFAULT FALSE,
    created TIMESTAMP,
    modified TIMESTAMP,
    UNIQUE (slug),
    FOREIGN KEY (user_id) REFERENCES users(id)
);

CREATE TABLE tags (
    id SERIAL PRIMARY KEY,
    title VARCHAR(191),
    created TIMESTAMP,
    modified TIMESTAMP,
    UNIQUE (title)
);

CREATE TABLE articles_tags (
    article_id INT NOT NULL,
```

(continué en la próxima página)

(proviene de la página anterior)

```
    tag_id INT NOT NULL,
    PRIMARY KEY (article_id, tag_id),
    FOREIGN KEY (tag_id) REFERENCES tags(id),
    FOREIGN KEY (article_id) REFERENCES articles(id)
);

INSERT INTO users (email, password, created, modified)
VALUES
('cakephp@example.com', 'secret', NOW(), NOW());

INSERT INTO articles (user_id, title, slug, body, published, created, modified)
VALUES
(1, 'First Post', 'first-post', 'This is the first post.', TRUE, NOW(), NOW());
```

Es posible que haya notado que la tabla `articles_tags` utiliza una clave primaria compuesta. CakePHP admite claves primarias compuestas en casi todas partes, lo que le permite tener esquemas más simples que no requieren columnas `id` adicionales.

Los nombres de tabla y columna que usamos no fueron arbitrarios. Al usar las *convenciones de nomenclatura* de CakePHP, podemos aprovechar CakePHP más eficazmente y evitar la necesidad de configurar el framework. Si bien CakePHP es lo suficientemente flexible para adaptarse a casi cualquier esquema de base de datos, adherirse a las convenciones le ahorrará tiempo, ya que puede aprovechar los valores predeterminados basados en convenciones que ofrece CakePHP.

Configuración de la base de datos

A continuación, digamos a CakePHP dónde está nuestra base de datos y cómo conectarse a ella. Reemplace los valores en el arreglo `Datasources.default` en su archivo **config/app.php** con los que aplican a su configuración. Una arreglo de configuración completo de muestra podría tener el siguiente aspecto:

```
<?php
return [
    // Más configuración arriba.
    'Datasources' => [
        'default' => [
            'className' => 'Cake\Database\Connection',
            // Replace Mysql with Postgres if you are using PostgreSQL
            'driver' => 'Cake\Database\Driver\Mysql',
            'persistent' => false,
            'host' => 'localhost',
            'username' => 'cakephp',
            'password' => 'AngelF00dC4k3~',
            'database' => 'cake_cms',
            // Comment out the line below if you are using PostgreSQL
            'encoding' => 'utf8mb4',
            'timezone' => 'UTC',
            'cacheMetadata' => true,
        ],
    ],
    // Más configuración abajo.
];
```

Una vez que haya guardado su archivo **config/app.php**, debería ver que la sección "CakePHP is able to connect to the database" tiene un gorro de cocinero verde.

Nota: Si tiene **config/app_local.php** en la carpeta de su aplicación, este anula la configuración de app.php.

Creando nuestro primer modelo

Los modelos son el corazón de las aplicaciones CakePHP. Nos permiten leer y modificar nuestros datos. Nos permiten construir relaciones entre nuestros datos, validarlos y aplicar reglas de aplicación. Los modelos construyen las bases necesarias para construir nuestras acciones y plantillas del controlador.

Los modelos de CakePHP se componen de objetos `Table` y `Entity`. Los objetos `Table` brindan acceso a la colección de entidades almacenadas en una tabla específica. Se almacenan en **src/Model/Table**. El archivo que crearemos se guardará en **src/Model/Table/ArticlesTable.php**. El archivo completo debería verse así:

```
<?php
// src/Model/Table/ArticlesTable.php
namespace App\Model\Table;

use Cake\ORM\Table;

class ArticlesTable extends Table
{
    public function initialize(array $config): void
    {
        $this->addBehavior('Timestamp');
    }
}
```

Hemos agregado el comportamiento *Timestamp Behavior* que automáticamente llenará las columnas `created` y `modified` de nuestra tabla. Al nombrar nuestro objeto `Table` `ArticlesTable`, CakePHP puede usar convenciones de nomenclatura para saber que nuestro modelo usa la tabla *articles* ` de la base de datos. CakePHP también usa convenciones para saber que la columna `id` es la clave primaria de nuestra tabla.

Nota: CakePHP creará dinámicamente un objeto modelo para usted si no puede encontrar un archivo correspondiente en **src/Model/Table**. Esto también significa que si accidentalmente asigna un nombre incorrecto a su archivo (es decir, articlestable.php o ArticleTable.php), CakePHP no reconocerá ninguna de sus configuraciones y utilizará el modelo generado en su lugar.

También crearemos una clase `Entity` para nuestros artículos. Las `Entity` representan un solo registro en la base de datos y proporcionan un comportamiento a nivel de fila para nuestros datos. Nuestra `Entity` se guardará en **src/Model/Entity/Article.php**. El archivo completo debería verse así:

```
<?php
// src/Model/Entity/Article.php
namespace App\Model\Entity;

use Cake\ORM\Entity;

class Article extends Entity
{
```

(continué en la próxima página)

(proviene de la página anterior)

```
    protected $_accessible = [
        '*' => true,
        'id' => false,
        'slug' => false,
    ];
}
```

Nuestra entidad es bastante delgada en este momento, y solo hemos configurado la propiedad `_accessible` que controla cómo las propiedades pueden ser modificadas por *entities-mass-assignment*.

No podemos hacer mucho con nuestros modelos en este momento, así que a continuación crearemos nuestro primer *Controller y Template </tutorials-and-examples/cms/articles-controller>* para permitirnos interactuar con nuestro modelo.

Tutorial Bookmarker (Favoritos)

Este tutorial te guiará en la creación de una aplicación sencilla para el guardado de favoritos (Bookmaker).

Para comenzar instalaremos CakePHP creando nuestra base de datos y utilizaremos las herramientas que CakePHP provee para realizar nuestra aplicación rápidamente.

Esto es lo que necesitarás:

1. Un servidor de base de datos. Nosotros utilizaremos MySQL en este tutorial. Necesitarás tener los conocimientos suficientes de SQL para crear una base de datos; CakePHP tomará las riendas desde ahí. Al utilizar MySQL asegúrate de que tienes habilitado `pdo_mysql` en PHP.
2. Conocimientos básicos de PHP.

Antes de empezar deberías de asegurarte de que tienes actualizada la versión de PHP:

```
php -v
```

Deberías tener instalado PHP 7.4 (CLI) o superior. La versión PHP de tu servidor web deberá ser 7.4 o superior y lo ideal es que coincida con la versión de la interfaz de línea de comandos (CLI) de PHP. Si quieres ver la aplicación ya finalizada puedes consultar cakephp/bookmarker[30].

Empecemos!

Instalar CakePHP

La forma más sencilla de instalar CakePHP es utilizando Composer, una manera sencilla de instalar CakePHP desde tu terminal o prompt de línea de comandos.

Primero necesitarás descargar e instalar Composer si aún no lo tienes. Si ya tienes instalado cURL es tan sencillo como ejecutar:

```
curl -s https://getcomposer.org/installer | php
```

O puedes descargar `composer.phar` desde la Página web de Composer[31].

Después sencillamente escribe la siguiente línea en tu terminal desde tu directorio de instalación para instalar el esqueleto de la aplicación CakePHP en el directorio **bookmarker**:

[30] https://github.com/cakephp/bookmarker-tutorial
[31] https://getcomposer.org/download/

```
php composer.phar create-project --prefer-dist cakephp/app:4.* bookmarker
```

Si descargaste y ejecutaste el Instalador Windows de Composer[32], entonces escribe la siguiente línea en tu terminal desde tu directorio de instalación (ie. C:\wamp\www\dev\cakephp3):

```
composer self-update && composer create-project --prefer-dist cakephp/app:4.* bookmarker
```

La ventaja de utilizar Composer es que automáticamente realizará algunas tareas importantes como configurar correctamente el archivo de permisos y crear tu archivo **config/app.php**.

Hay otras formas de instalar CakePHP. Si no puedes o no quieres utilizar Composer comprueba la sección *Instalación*.

Sin importar como hayas descargado e instalado CakePHP, una vez hayas finalizado, tu directorio de instalación debería ser algo como:

```
/bookmarker
    /bin
    /config
    /logs
    /plugins
    /src
    /tests
    /tmp
    /vendor
    /webroot
    .editorconfig
    .gitignore
    .htaccess
    .travis.yml
    composer.json
    index.php
    phpunit.xml.dist
    README.md
```

Ahora podría ser un buen momento para que aprendas un poco sobre como funciona la estructura de directorios de CakePHP: *CakePHP Folder Structure*.

Comprobar la instalación

Podemos comprobar rápidamente que nuestra instalación ha sido correcta accediendo a la página principal que se crea por defecto.

Pero antes necesitarás inicializar el servidor de desarrollo:

```
bin/cake server
```

Nota: Para Windows introduce el comando `bin\cake server` (fíjate en la \ ).

Esto arrancará el servidor integrado en el puerto 8765. Accede a **http://localhost:8765** a través de tu navegador para ver la página de bienvenida. Todos los items deberán estar marcados como correctos para que CakePHP pueda conectarse a tu base de datos. Si no, puede que necesites instalar extensiones adicionales de PHP, o dar permisos de directorio.

[32] https://getcomposer.org/Composer-Setup.exe

Crear la base de datos

Continuamos, creemos ahora la base de datos para nuestra aplicación de favoritos.

Si aún no lo has hecho, crea una base de datos vacía para usar en este tutorial con el nombre que tu quieras, e.g. `cake_bookmarks`.

Puedes ejecutar la siguiente sentencia SQL para crear las tablas necesarias:

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    created DATETIME,
    modified DATETIME
);

CREATE TABLE bookmarks (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    title VARCHAR(50),
    description TEXT,
    url TEXT,
    created DATETIME,
    modified DATETIME,
    FOREIGN KEY user_key (user_id) REFERENCES users(id)
);

CREATE TABLE tags (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255),
    created DATETIME,
    modified DATETIME,
    UNIQUE KEY (title)
);

CREATE TABLE bookmarks_tags (
    bookmark_id INT NOT NULL,
    tag_id INT NOT NULL,
    PRIMARY KEY (bookmark_id, tag_id),
    FOREIGN KEY tag_key(tag_id) REFERENCES tags(id),
    FOREIGN KEY bookmark_key(bookmark_id) REFERENCES bookmarks(id)
);
```

Puedes ver que la tabla `bookmarks_tags` utiliza una clave primaria compuesta. CakePHP soporta claves primarias compuestas en casi cualquier lado, haciendo más fácil construir aplicaciones multi-anidadas.

Los nombres de las tablas y columnas que hemos utilizado no son aleatorios. Utilizando las *convenciones de nombres* podemos hacer mejor uso de CakePHP y evitar tener que configurar el framework.

CakePHP es lo suficientemente flexible para acomodarse incluso a esquemas inconsistentes de bases de datos heredados, pero siguiendo las convenciones ahorrarás tiempo.

Configuración de la base de datos

Siguiente, indiquémosle a CakePHP donde está nuestra base de datos y como conectarse a ella. Para la mayoría de las veces esta será la primera y última vez que necesitarás configurar algo.

La configuración debería ser bastante sencilla: sólo cambia los valores del array `Datasources.default` en el archivo **config/app.php** por aquellos que apliquen a tu instalación. Un ejemplo de array de configuración completado puede lucir así:

```
return [
    // More configuration above.
    'Datasources' => [
        'default' => [
            'className' => 'Cake\Database\Connection',
            'driver' => 'Cake\Database\Driver\Mysql',
            'persistent' => false,
            'host' => 'localhost',
            'username' => 'cakephp',
            'password' => 'AngelF00dC4k3~',
            'database' => 'cake_bookmarks',
            'encoding' => 'utf8',
            'timezone' => 'UTC',
            'cacheMetadata' => true,
        ],
    ],
    // More configuration below.
];
```

Una vez hayas guardado tu archivo **config/app.php** deberías ver que la sección "CakePHP is able to connect to the database" tiene un chechmark de correcto.

Nota: Puedes encontrar una copia de la configuración por defecto de CakePHP en **config/app.default.php**.

Crear el esqueleto del código

Gracias a que nuestra base de datos sigue las convenciones de CakePHP podemos utilizar la consola de bake de la aplicación para crear rápidamente una aplicación básica.

En tu línea de comandos ejecuta las siguientes instrucciones:

```
// En Windows necesitarás utilizar bin\cake.
bin/cake bake all users
bin/cake bake all bookmarks
bin/cake bake all tags
```

Esto creará los controladores, modelos, vistas, sus correspondientes casos de prueba y accesorios para nuestros recursos de users, bookmarks y tags.

Si detuviste tu servidor reinícialo.

Vete a **http://localhost:8765/bookmarks**, deberías poder ver una básica pero funcional aplicación provista de acceso a las tablas de tu base de datos.

Una vez estés en la lista de bookmarks añade unos cuantos usuarios (users), favoritos (bookmarks) y etiquetas (tags)

Nota: Si ves una página de error Not Found (404) comprueba que el módulo de Apache mod_rewrite está cargado.

Añadir encriptación (hashing) a la contraseña

Cuando creaste tus usuarios (visitando **http://localhost:8765/users**) probablemente te darías cuenta de que las contraseñas (password) se almacenaron en texto plano. Algo muy malo desde un punto de vista de seguridad, así que arreglémoslo.

Éste es también un buen momento para hablar de la capa de modelo en CakePHP.

En CakePHP separamos los métodos que operan con una colección de objetos y los que lo hacen con un único objeto en diferentes clases.

Los métodos que operan con una coleccion de entidades van en la clase `Table`, mientras que los que lo hacen con una sola van en la clase `Entity`.

Por ejemplo: el encriptado de una contraseña se hace en un registro individual, por lo que implementaremos este comportamiento en el objeto Entity.

Ya que lo que queremos es encriptar la contraseña cada vez que la introduzcamos en la base de datos utilizaremos un método mutador/setter.

CakePHP utilizará la convención para métodos setter cada vez que una propiedad se introducida en una de tus entidades.

Añadamos un setter para la contraseña añadiendo el siguiente código en **src/Model/Entity/User.php**:

```
namespace App\Model\Entity;

use Cake\Auth\DefaultPasswordHasher; //include this line
use Cake\ORM\Entity;

class User extends Entity
{

    // Code from bake.

    protected function _setPassword($value)
    {
        $hasher = new DefaultPasswordHasher();

        return $hasher->hash($value);
    }
}
```

Ahora actualiza uno de los usuarios que creaste antes, si cambias su contraseña deberías ver una contraseña encriptada en vez del valor original en la lista de usuarios o en su página de View.

CakePHP encripta contraseñas con bcrypt[33] por defecto. Puedes usar también sha1 o md5 si estás trabajando con bases de datos ya existentes.

Nota: Si la contraseña no se ha encriptado asegúrate de que has usado el mismo estilo de escritura que el del atributo password de la clase cuando nombraste la función setter.

[33] https://codahale.com/how-to-safely-store-a-password/

Obtener bookmarks con un tag específico

Ahora que estamos almacenando contraseñas con seguridad podemos añadir alguna funcionalidad interesante a nuestra aplicación.

Cuando acumulas una colección de favoritos es útil poder buscarlos a través de etiquetas.

Implementemos una ruta, una acción de controlador y un método finder para buscar bookmarks mediante etiquetas.

Idealmente tendríamos una URL como **http://localhost:8765/bookmarks/tagged/funny/cat/gifs** que nos permitiría encontrar todos los bookmarks que tienen las etiquetas "funny", "cat" o "gifs".

Antes de que podamos implementarlo añadiremos una nueva ruta.

Modifica tu **config/routes.php** para que se vea como ésto:

```
<?php
use Cake\Routing\Route\DashedRoute;
use Cake\Routing\Router;

Router::defaultRouteClass(DashedRoute::class);

// Nueva ruta que añadimos para nuestra acción tagged
// The trailing `*` tells CakePHP that this action has
// passed parameters.
Router::scope(
    '/bookmarks',
    ['controller' => 'Bookmarks'],
    function ($routes) {
        $routes->connect('/tagged/*', ['action' => 'tags']);
    }
);

Router::scope('/', function ($routes) {
    // Connect the default home and /pages/* routes.
    $routes->connect('/', [
        'controller' => 'Pages',
        'action' => 'display', 'home'
    ]);
    $routes->connect('/pages/*', [
        'controller' => 'Pages',
        'action' => 'display'
    ]);

    // Connect the conventions based default routes.
    $routes->fallbacks();
});
```

Lo cual define una nueva "ruta" que conecta el path **/bookmarks/tagged/** a `BookmarksController::tags()`.

Con la definición de rutas puedes separar como se ven tus URLs de como se implementan. Si visitamos **http://localhost:8765/bookmarks/tagged**, podremos ver una página de error bastante útil de CakePHP informando que no existe la acción del controlador.

Implementemos ahora ese método.

En **src/Controller/BookmarksController.php** añade:

```
public function tags()
{
    // The 'pass' key is provided by CakePHP and contains all
    // the passed URL path segments in the request.
    $tags = $this->request->getParam('pass');

    // Use the BookmarksTable to find tagged bookmarks.
    $bookmarks = $this->Bookmarks->find('tagged', [
        'tags' => $tags
    ]);

    // Pass variables into the view template context.
    $this->set([
        'bookmarks' => $bookmarks,
        'tags' => $tags
    ]);
}
```

Para acceder a otras partes del request consulta *Request*.

Crear el método finder

En CakePHP nos gusta mantener las acciones de los controladores sencillas y poner la mayoría de la lógica de la aplicación en los modelos. Si visitas ahora la URL **/bookmarks/tagged** verás un error de que el método `findTagged()` no ha sido implementado todavía, asi que hagámoslo.

En **src/Model/Table/BookmarksTable.php** añade lo siguiente:

```
// El argumento $query es una instancia de query.
// El array $options contendrá las opciones de 'tags' que pasemos
// para encontrar'tagged') en nuestra acción del controlador.
public function findTagged(Query $query, array $options)
{
    $bookmarks = $this->find()
        ->select(['id', 'url', 'title', 'description']);

    if (empty($options['tags'])) {
        $bookmarks
            ->leftJoinWith('Tags')
            ->where(['Tags.title IS' => null]);
    } else {
        $bookmarks
            ->innerJoinWith('Tags')
            ->where(['Tags.title IN ' => $options['tags']]);
    }

    return $bookmarks->group(['Bookmarks.id']);
}
```

Acabamos de implementar un *método finder personalizado*.

Esto es un concepto muy poderoso en CakePHP que te permite empaquetar queries re-utilizables.

Los métodos finder siempre reciben un objeto *Query Builder* y un array de opciones como parámetros. Estos métodos

pueden manipular la query y añadir cualquier condición o criterio requerido; cuando se completan devuelven un objeto query modificado.

En nuestro método finder sacamos provecho de los métodos `distinct()` y `matching()` que nos permiten encontrar distintos ("distincts") bookmarks que tienen un tag coincidente (matching). El método `matching()` acepta una función anónima[34] que recibe un generador de consultas. Dentro del callback usaremos este generador para definir las condiciones que filtrarán bookmarks que tienen las etiquetas (tags) especificadas.

Crear la vista

Ahora si visitas la URL **/bookmarks/tagged**, CakePHP mostrará un error advirtiéndote de que no has creado un archivo de vista.

Siguiente paso, creemos un archivo de vista para nuestro método `tags()`.

En **templates/Bookmarks/tags.php** añade el siguiente código:

```
<h1>
    Bookmarks tagged with
    <?= $this->Text->toList(h($tags)) ?>
</h1>

<section>
<?php foreach ($bookmarks as $bookmark): ?>
    <article>
        <!-- Use the HtmlHelper to create a link -->
        <h4><?= $this->Html->link($bookmark->title, $bookmark->url) ?></h4>
        <small><?= h($bookmark->url) ?></small>

        <!-- Use the TextHelper to format text -->
        <?= $this->Text->autoParagraph(h($bookmark->description)) ?>
    </article>
<?php endforeach; ?>
</section>
```

En el código de arriba utilizamos los helpers *HtmlHelper* y *TextHelper* para que asistan en la generación de nuestra salida de la vista.

También utilizamos la función de atajo `h()` para salidas de código HTML. Deberías acordarte siempre de utilizar `h()` cuando muestres datos del usuario para evitar problemas de inyección HTML.

El archivo **tags.php** que acabamos de crear sigue las convenciones de CakePHP para archivos de vistas. La convención es que el nombre del archivo sea una versión en minúsculas y subrayados del nombre de la acción del controlador.

Puedes observar que hemos podido usar las variables `$tags` y `$bookmarks` en nuestra vista.

Cuando utilizamos el método `set()` en nuestro controlador especificamos variables para enviarlas a la vista. Ésta hará disponibles todas las variables que se le pasen como variables locales.

Ahora deberías poder visitar la URL **/bookmarks/tagged/funny** y ver todos los favoritos etiquetados con "funny".

Hasta aquí hemos creado una aplicación básica para manejar favoritos (bookmarks), etiquetas (tags) y usuarios (users). Sin embargo todo el mundo puede ver las etiquetas de los demás. En el siguiente capítulo implementaremos autenticación y restringiremos el uso de etiquetas únicamente a aquellas que pertenezcan al usuario actual.

Ahora ve a *Tutorial Bookmarker (Favoritos) - Parte 2* para continuar construyendo tu apliación o sumérgete en la documentación para aprender más sobre que puede hacer CakePHP por ti.

[34] https://php.net/manual/es/functions.anonymous.php

Tutorial Bookmarker (Favoritos) - Parte 2

Tras realizar *la primera parte de este tutorial* deberías tener una aplicación muy básica para guardar favoritos.

En este capítulo añadiremos la autenticación y restringiremos los favoritos (bookmarks) para que cada usuario pueda consultar o modificar solamente los suyos.

Añadir login

En CakePHP, la autenticación se maneja mediante *Componentes*.

Los componentes pueden verse como una forma de crear trozos reutilizables de código de controlador para una finalidad o idea. Además pueden engancharse al evento de ciclo de vida de los controladores e interactuar con tu aplicación de ese modo.

Para empezar añadiremos el componente *AuthComponent* a nuestra aplicación.

Como queremos que todos nuestros métodos requieran de autenticación añadimos *AuthComponent* en *AppController* del siguiente modo:

```
// En src/Controller/AppController.php
namespace App\Controller;

use Cake\Controller\Controller;

class AppController extends Controller
{
    public function initialize()
    {
        $this->loadComponent('Flash');
        $this->loadComponent('Auth', [
            'authenticate' => [
                'Form' => [
                    'fields' => [
                        'username' => 'email',
                        'password' => 'password'
                    ]
                ]
            ],
            'loginAction' => [
                'controller' => 'Users',
                'action' => 'login'
            ],
            'unauthorizedRedirect' => $this->referer() // Si no está autorizado,
                                //el usuario regresa a la página que estaba
        ]);

        // Permite ejecutar la acción display para que nuestros controladores de páginas
        // sigan funcionando.
        $this->Auth->allow(['display']);
    }
}
```

Acabamos de decirle a CakePHP que queremos cargar los compomentes `Flash` y `Auth`. Además hemos personalizado la configuración de *AuthComponent* indicando que utilice como *username* el campo *email* de la tabla *Users* de la base

de datos.

Ahora si vas a cualquier *URL* serás enviado a **/users/login**, que mostrará una página de error ya que no hemos escrito el código de la función *login* todavía, así que hagámoslo ahora:

```
// En src/Controller/UsersController.php
public function login()
{
    if ($this->request->is('post')) {
        $user = $this->Auth->identify();
        if ($user) {
            $this->Auth->setUser($user);

            return $this->redirect($this->Auth->redirectUrl());
        }
        $this->Flash->error('Tu usuario o contraseña es incorrecta.');
    }
}
```

Y en **templates/Users/login.php** añade lo siguiente:

```
<h1>Login</h1>
<?= $this->Form->create() ?>
<?= $this->Form->input('email') ?>
<?= $this->Form->input('password') ?>
<?= $this->Form->button('Login') ?>
<?= $this->Form->end() ?>
```

Ahora que tenemos un formulario de *login* sencillo deberíamos poder loguearnos con algún usuario que tenga contraseña encriptada.

Nota:

Si ninguno de tus usuarios tiene contraseña encriptada comenta la línea
`loadComponent('Auth')`, a continuación edita un usuario y modifica la contraseña.

Ahora deberías poder loguearte, si no es así asegúrate de que estás utilizando un usuario con contraseña encriptada.

Añadir *logout*

Ahora que la gente puede loguearse probablemente quieras añadir una forma de desloguearse también.

Otra vez en `UsersController`, añade el siguiente código:

```
public function initialize()
{
    parent::initialize();
    $this->Auth->allow(['logout']);
}

public function logout()
{
    $this->Flash->success('Ahora estás deslogueado.');
```

(continué en la próxima página)

(proviene de la página anterior)

```
    return $this->redirect($this->Auth->logout());
}
```

Este código añade la acción `logout` como una acción pública e implementa la función.

Ahora puedes visitar `/users/logout` para desloguearte, deberías ser enviado a la página de inicio.

Habilitar registros

Si no estás logueado e intentas acceder a **/users/add** eres reenviado a la página de login. Deberíamos arreglar esto si queremos permitir que la gente se pueda registrar en nuestra aplicación.

En el controlador `UsersController` añade lo siguiente:

```
public function initialize()
{
    parent::initialize();
    // Añade logout a la lista de actiones permitidas.
    $this->Auth->allow(['logout', 'add']);
}
```

El código anterior le dice a `AuthComponent` que la acción `add()` no necesita autenticación ni autorización.

Tal vez quieras tomarte un tiempo para limpiar **Users/add.php** y eliminar los enlaces erróneos o continuar con el siguiente apartado. No vamos a crear la edición de usuarios, consulta o listado en este tutorial así que no funcionará el control de `AuthComponent` para el acceso a esas acciones del controlador.

Restringiendo el acceso a favoritos

Ahora que los usuarios pueden loguearse queremos restringir los favoritos que uno puede ver a los que creó. Esto lo haremos usando un adaptador de "authorization".

Ya que nuestro requisito es muy sencillo podremos escribir un código también muy sencillo en nuestro `BookmarksController`.

Pero antes necesitamos decirle al componente *AuthComponent* cómo va a autorizar acciones nuestra aplicación. Para ello añade en `AppController`:

```
public function isAuthorized($user)
{
    return false;
}
```

Además añade la siguiente línea a la configuración de `Auth` en tu `AppController`:

```
'authorize' => 'Controller',
```

Tú método `initialize()` debería verse así:

```
public function initialize()
{
    $this->loadComponent('Flash');
```

(continué en la próxima página)

(proviene de la página anterior)

```
    $this->loadComponent('Auth', [
        'authorize'=> 'Controller',// línea añadida
        'authenticate' => [
            'Form' => [
                'fields' => [
                    'username' => 'email',
                    'password' => 'password'
                ]
            ]
        ],
        'loginAction' => [
            'controller' => 'Users',
            'action' => 'login'
        ],
        'unauthorizedRedirect' => $this->referer()
    ]);

    // Permite ejecutar la acción display para que nuestros controladores
    // de páginas sigan funcionando.
    $this->Auth->allow(['display']);
}
```

Por defecto denegaremos el acceso siempre y concederemos los accesos donde tenga sentido.

Primero añadiremos la lógica de autorización para favoritos.

En tu `BookmarksController` añade lo siguiente:

```
public function isAuthorized($user)
{
    $action = $this->request->getParam('action');

    // Las acciones add e index están siempre permitidas.
    if (in_array($action, ['index', 'add', 'tags'])) {
        return true;
    }
    // El resto de acciones requieren un id.
    if (!$this->request->getParam('pass.0')) {
        return false;
    }

    // Comprueba que el favorito pertenezca al usuario actual.
    $id = $this->request->getParam('pass.0');
    $bookmark = $this->Bookmarks->get($id);
    if ($bookmark->user_id == $user['id']) {
        return true;
    }

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

Ahora si intentas consultar, editar o borrar un favorito que no te pertenece deberías ser redirigido a la página desde la que accediste.

Si no se muestra ningún mensaje de error añade lo siguiente a tu layout:

```
// En templates/layout/default.php
<?= $this->Flash->render() ?>
```

Deberías poder ver ahora los mensajes de error de autorización.

Arreglar lista de consulta y formularios

Mientras que *view* y *delete* están funcionando, *edit*, *add* e *index* presentan un par de problemas:

1. Cuando añades un favorito puedes elegir el usuario.
2. Cuando editas un favorito puedes elegir un usuario.
3. La página con el listado muestra favoritos de otros usuarios.

Abordemos el formulario de añadir favorito primero.

Para empezar elimina `input('user_id')` de **templates/Bookmarks/add.php**.

Con esa parte eliminada actualizaremos la acción `add()` de **src/Controller/BookmarksController.php** para que luzca así:

```
public function add()
{
    $bookmark = $this->Bookmarks->newEntity();
    if ($this->request->is('post')) {
        $bookmark = $this->Bookmarks->patchEntity($bookmark, $this->request->getData());
        $bookmark->user_id = $this->Auth->user('id');
        if ($this->Bookmarks->save($bookmark)) {
            $this->Flash->success('El favorito se ha guardado.');

            return $this->redirect(['action' => 'index']);
        }
        $this->Flash->error('El favorito podría no haberse guardado. Por favor,␣
↪inténtalo de nuevo.');
    }
    $tags = $this->Bookmarks->Tags->find('list');
    $this->set(compact('bookmark', 'tags'));
    $this->set('_serialize', ['bookmark']);
}
```

Completando la propiedad de la entidad con datos de la sesión eliminaremos cualquier posibilidad de que el usuario modifique el usuario al que pertenece el favorito. Haremos lo mismo para el formulario de edición.

Tu acción `edit()` de **src/Controller/BookmarksController.php** debería ser así:

```
public function edit($id = null)
{
    $bookmark = $this->Bookmarks->get($id, [
        'contain' => ['Tags']
    ]);
    if ($this->request->is(['patch', 'post', 'put'])) {
        $bookmark = $this->Bookmarks->patchEntity($bookmark, $this->request->getData());
        $bookmark->user_id = $this->Auth->user('id');
```

(continué en la próxima página)

(proviene de la página anterior)

```
        if ($this->Bookmarks->save($bookmark)) {
            $this->Flash->success('El favorito se ha guardado.');

            return $this->redirect(['action' => 'index']);
        }
        $this->Flash->error('El favorito podría no haberse guardado. Por favor,␣
↪inténtalo de nuevo.');
    }
    $tags = $this->Bookmarks->Tags->find('list');
    $this->set(compact('bookmark', 'tags'));
    $this->set('_serialize', ['bookmark']);
}
```

Listado consulta

Ahora solo necesitamos mostrar los favoritos del usuario actualmente logueado.

Podemos hacer eso actualizando la llamada a `paginate()`. Haz que tu método `index()` de **src/Controller/BookmarksController.php** se vea así:

```
public function index()
{
    $this->paginate = [
        'conditions' => [
            'Bookmarks.user_id' => $this->Auth->user('id'),
        ]
    ];
    $this->set('bookmarks', $this->paginate($this->Bookmarks));
    $this->set('_serialize', ['bookmarks']);
}
```

Deberíamos actualizar también el método `tags()` y el método finder relacionado, pero lo dejaremos como un ejercicio para que lo hagas por tu cuenta.

Mejorar la experiencia de etiquetado

Ahora mismo añadir nuevos tags es un proceso complicado desde que `TagsController` desautorizó todos los accesos.

En vez de permitirlos podemos mejorar la *UI* para la selección de tags utilizando un campo de texto separado por comas. Esto proporcionará una mejor experiencia para nuestros usuarios y usa algunas de las mejores características de *ORM*.

Añadir un campo calculado

Para acceder de forma sencilla a las etiquetas formateadas podemos añadir un campo virtual/calculado a la entidad.

En **src/Model/Entity/Bookmark.php** añade lo siguiente:

```
use Cake\Collection\Collection;

protected function _getTagString()
{
    if (isset($this->_fields['tag_string'])) {
        return $this->_fields['tag_string'];
    }
    if (empty($this->tags)) {
        return '';
    }
    $tags = new Collection($this->tags);
    $str = $tags->reduce(function ($string, $tag) {
        return $string . $tag->title . ', ';
    }, '');

    return trim($str, ', ');
}
```

Esto nos dará acceso a la propiedad calculada `$bookmark->tag_string` que utilizaremos más adelante.

Recuerda añadir la propiedad `tag_string` a la lista `_accessible` en tu entidad para poder "guardarla" más adelante.

En **src/Model/Entity/Bookmark.php** añade `tag_string` a `$_accessible` de este modo:

```
protected $_accessible = [
    'user_id' => true,
    'title' => true,
    'description' => true,
    'url' => true,
    'user' => true,
    'tags' => true,
    'tag_string' => true,
];
```

Actualizar las vistas

Con la entidad actualizada podemos añadir un nuevo campo de entrada para nuestros tags. En **templates/Bookmarks/add.php** y **templates/Bookmarks/edit.php**, cambia el campo `tags._ids` por el siguiente:

```
echo $this->Form->input('tag_string', ['type' => 'text']);
```

Guardar el string de tags

Ahora que podemos ver los tags existentes como un string querremos guardar también esa información.

Al haber marcado `tag_string` como accesible el ORM copiará esa información del request a nuestra entidad. Podemos usar un método de gancho `beforeSave()` para parsear el *string* de etiquetas y encontrar/crear las entidades relacionadas.

Añade el siguiente código a **src/Model/Table/BookmarksTable.php**:

```
public function beforeSave($event, $entity, $options)
{
    if ($entity->tag_string) {
        $entity->tags = $this->_buildTags($entity->tag_string);
    }
}

protected function _buildTags($tagString)
{
    // Hace trim a las etiquetas
    $newTags = array_map('trim', explode(',', $tagString));
    // Elimina las etiquetas vacías
    $newTags = array_filter($newTags);
    // Elimina las etiquetas duplicadas
    $newTags = array_unique($newTags);

    $out = [];
    $query = $this->Tags->find()
        ->where(['Tags.title IN' => $newTags]);

    // Elimina las etiquetas existentes de la lista de nuevas etiquetas.
    foreach ($query->extract('title') as $existing) {
        $index = array_search($existing, $newTags);
        if ($index !== false) {
            unset($newTags[$index]);
        }
    }
    // Añade las etiquetas existentes.
    foreach ($query as $tag) {
        $out[] = $tag;
    }
    // Añade las etiquetas nuevas.
    foreach ($newTags as $tag) {
        $out[] = $this->Tags->newEntity(['title' => $tag]);
    }

    return $out;
}
```

Aunque este código sea algo más complicado de lo que hemos hecho hasta ahora, nos ayudará a ver lo potente que es el *ORM* en CakePHP.

Puedes manipular los resultados de la consulta usando los métodos *Collections* y manejar escenearios en los que estás creando entidades *on the fly* con facilidad.

Para finalizar

Hemos mejorado nuestra aplicación de favoritos para manejar escenarios de autenticación y de autorización/control de acceso básicos.

Además hemos añadido algunas mejoras interesantes de experiencia de usuario sacándole provecho a *FormHelper* y al potencial de *ORM*.

Gracias por tomarte tu tiempo para explorar CakePHP. Ahora puedes realizar el tutorial *Tutorial Blog*, aprender más sobre *Acceso a la base de datos & ORM*, o puedes leer detenidamente los /topics.

Tutorial Blog

Bienvenido a CakePHP. Probablemente estás consultando este tutorial porque quieres aprender más sobre 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.

Este tutorial te guiará en la creación de una aplicación sencilla de blog. Obtendremos e instalaremos CakePHP, crearemos y configuraremos la base de datos y añadiremos suficiente lógica como para listar, añadir, editar y eliminar artículos del blog.

Esto es lo que necesitarás:

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 la mayoría pueden poner en marcha CakePHP sin configuración alguna. Asegúrate de tener PHP 7.4 o superior así como tener las extensiones `mbstring`, `intl` y `mcrypt` activadas en PHP.
2. Servidor de base de datos. Usaremos MySQL en este tutorial. Necesitarás saber cómo crear una base de datos nueva. CakePHP se encargará del resto. Dado que utilizamos MySQL, asegúrate también de tener `pdo_mysql` habilitado en PHP.
3. Conocimientos básicos de PHP.

¡Vamos allá!

Obtener CakePHP

La manera más sencilla de ponerse en marcha es utilizando Composer. Composer te permite instalar fácilmente CakePHP desde tu terminal o consola. Primero, debes descargar e instalar Composer si todavía no lo has hecho. Si tienes cURL instalado, es tan fácil como ejecutar lo siguiente:

```
curl -s https://getcomposer.org/installer | php
```

O puedes descargar `composer.phar` desde la página web de Composer[35].

Instalando Composer de manera global evitarás tener que repetir este paso para cada proyecto.

Luego, simplemente escribe la siguiente línea en tu terminal desde tu directorio de instalación para instalar el esqueleto de la aplicación de CakePHP en el directorio [nombre_app].

[35] https://getcomposer.org/download/

```
php composer.phar create-project --prefer-dist cakephp/app:4.* [nombre_app]
```

O si tienes Composer instalado globalmente:

```
composer create-project --prefer-dist cakephp/app:4.* [nombre_app]
```

La ventaja de utilizar Composer es que automáticamente completará algunas tareas de inicialización, como aplicar permisos a ficheros y crear tu fichero config/app.php por ti.

Existen otros modos de instalar CakePHP si no te sientes cómodo con Composer. Para más información revisa la sección *Instalación*.

Dejando de lado cómo has descargado e instalado CakePHP, una vez ha terminado la configuración, tu directorio de instalación debería tener la siguiente estructura:

```
/directorio_raiz
    /config
    /logs
    /src
    /plugins
    /tests
    /tmp
    /vendor
    /webroot
    .gitignore
    .htaccess
    .travis.yml
    README.md
    composer.json
    phpunit.xml.dist
```

Quizás sea buen momento para aprender algo sobre cómo funciona esta estructura de directorios: echa un vistazo a la sección *CakePHP Folder Structure*.

Permisos de directorio en tmp

También necesitarás aplicar los permisos adecuados en el directorio `/tmp` para que el servidor web pueda escribir en él. El mejor modo de hacer esto es encontrar con qué usuario corre tu servidor web (`<?= `whoami`; ?>`) y cambiar la propiedad del directorio `tmp` hacia dicho usuario. El comando final que ejecutarás (en *nix) se parecerá al siguiente:

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

Si por alguna razón CakePHP no puede escribir en ese directorio, serás informado mediante una alerta mientras no estés en modo producción.

A pesar de que no se recomienda, si no eres capaz de aplicar la propiedad del directorio al mismo usuario que el servidor web, puedes simplemente aplicar permisos de escritura al directorio ejecutando un comando tipo:

```
$ chmod -R 777 tmp
```

Creando la base de datos del 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:

```
# Primero, creamos la tabla artículos
CREATE TABLE articles (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(50),
    body TEXT,
    created DATETIME DEFAULT NULL,
    modified DATETIME DEFAULT NULL
);

# Luego insertamos algunos artículos para probar
INSERT INTO articles (title,body,created)
    VALUES ('El título', 'Esto es el cuerpo del artículo.', NOW());
INSERT INTO articles (title,body,created)
    VALUES ('Un título de nuevo', 'Y el cuerpo sigue.', NOW());
INSERT INTO articles (title,body,created)
    VALUES ('El título ataca de nuevo', '¡Esto es realmente emocionante! No.', 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 CakePHP*), aprovecharás la potencia del framework y ahorrarás mucho trabajo de configuración. CakePHP es suficientemente flexible como para acomodarse hasta en el peor esquema de base de datos, pero utilizando las convenciones ahorrarás tiempo.

Echa un vistazo a *las convencionnes* para más información, pero basta decir que nombrando nuestra tabla "articles" automáticamente lo vincula a nuestro modelo Articles y que campos llamados *modified* y *created* serán gestionados automáticamente 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. Seguramente esta sea la primera y última vez que configuras nada.

Una copia del fichero de configuración de CakePHP puede ser hallado en **config/app.default.php**. Copia este fichero en su mismo directorio, pero nómbralo **app.php**.

El fichero de configuración debería de ser bastante sencillo: simplemente reemplaza los valores en la matriz `` Datasources.default`` con los que encajen con tu configuración. Una configuración completa de ejemplo podría parecerse a esto:

```
return [
    // Más configuración arriba
    'Datasources' => [
        'default' => [
            'className' => 'Cake\Database\Connection',
            'driver' => 'Cake\Database\Driver\Mysql',
            'persistent' => false,
            'host' => 'localhost',
            'username' => 'cake_blog',
```

(continué en la próxima página)

(proviene de la página anterior)

```
            'password' => 'AngelF00dC4k3~',
            'database' => 'cake_blog',
            'encoding' => 'utf8',
            'timezone' => 'UTC'
        ],
    ],
    // Más configuración abajo
];
```

En cuanto guardes tu nuevo fichero **app.php** deberías de ser capaz de acceder mediante tu navegador web y ver la página de bienvenida de CakePHP. También debería decirte que se ha encontrado el fichero de configuración así como que ha podido conectarse a la base de datos.

Nota: Recuerda que debes tener PDO y pdo_mysql habilitados en tu php.ini.

Configuración Opcional

Aún hay unas pocas cosas que puedes configurar. La mayoría de desarrolladores acaban estos ítems de la lista de la compra, pero no se necesitan para este tutorial. Uno de ellos es definir un string de seguridad (security salt) para realizar los "hash" de seguridad.

El string de seguridad se utiliza para generar "hashes". Cambia el valor por defecto editando el fichero **config/app.php**. No importa mucho el valor que contenga, cuanto más largo más difícil de averiguar:

```
'Security' => [
    'salt' => 'Algo largo y conteniendo un montón de distintos valores.',
],
```

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 mod_rewrite no está funcionando en tu sistema. Por favor, consulta la sección para tu servidor entre las siguientes acerca de re-escritura de URLs para poder poner en marcha la aplicación:

1. 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.
2. 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 **config/app.php**, quita el comentario a la línea:

```
'App' => [
    // ...
    // 'baseUrl' => env('SCRIPT_NAME'),
]
```

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

```
/.htaccess
/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 url-rewriting

Ahora continúa hacia *Tutorial Blog - Parte 2* para empezar a construir tu primera aplicación en CakePHP.

Tutorial Blog - Parte 2

Nota: The documentation is currently partially supported in es language for this page.

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

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

Crear un modelo Artículo (`Article`)

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 modelos están separados entre los objetos `Tabla` (`Table`) y `Entidad` (`Entity`). Los objetos `Tabla` proporcionan acceso a la coleción de entidades almacenada en una tabla específica y va en **src/Model/Table**. El fichero que crearemos se guardará en **src/Model/Table/ArticlesTable.php**. El fichero completo debería tener este aspecto:

```
namespace App\Model\Table;

use Cake\ORM\Table;

class ArticlesTable extends Table
{
    public function initialize(array $config)
    {
        $this->addBehavior('Timestamp');
    }
}
```

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

Los convenios usados para los nombres son importantes. Llamando a nuestro objeto Tabla `ArticlesTable`, CakePHP deducirá automáticamente que esta Tabla será utilizada en el controlador ArticlesController, y que se vinculará a una tabla en nuestra base de datos llamada `articles`.

Nota: CakePHP creará dinámicamente un objeto para el modelo si no encuentra el fichero correspondiente en **src/Model/Table**. Esto significa que si te equivocas al nombrar el fichero (por ejemplo lo llamas articlestable.php —en minúscula— o ArticleTable.php —en singular) CakePHP no va a reconocer la configuración que escribas en ese fichero y utilizará valores por defecto.

Para más información sobre modelos, como callbacks y validaciones echa un vistazo al capítulo del Manual *Acceso a la base de datos & ORM*.

Crear el Controlador de Artículos (`Articles Controller`)

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 **ArticlesController.php** dentro del directorio **src/Controller**. A continuación puedes ver el aspecto básico que debería tener este controlador:

```
namespace App\Controller;

class ArticlesController extends AppController
{
}
```

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/articles/index (que es lo mismo que www.example.com/articles/) esperan ver un listado de artículos. El código para tal acción sería este:

```
namespace App\Controller;

class ArticlesController extends AppController
{

    public function index()
    {
        $articles = $this->Articles->find('all');
        $this->set(compact('articles'));
    }
}
```

Por el hecho de haber definido el método `index()` en nuestro ArticlesController, los usuarios ahora pueden acceder a su lógica solicitando www.example.com/articles/index. Del mismo modo, si definimos un método llamado `foobar()` los usuarios tendrán acceso a él desde www.example.com/articles/foobar.

Advertencia: Puede que tengas la tentación de llamar tus controladores y acciones de cierto modo para obtener una URL en concreto. Resiste la tentación. Sigue las convenciones de CakePHP (mayúsculas, nombre en plural, etc.) y crea acciones comprensibles, que se dejen leer. Luego podrás asignar URLs a tu código utilizando «rutas», que veremos más adelante.

La única instrucción en la acción utiliza `set()` para pasar datos desde el controlador hacia la vista (que crearemos a

continuación). La línea en cuestión asigna una variable en la vista llamada "articles" igual al valor retornado por el método `find('all')` del objeto de tabla Artículos (`ArticlesTable`).

Para aprender más sobre los controladores, puedes visitar el capítulo *Controladores*.

Crear Vistas de Artículos (`Article Views`)

Ahora que tenemos nuestros datos fluyendo por el modelo, y que la lógica de nuestra aplicación está definida en nuestro controlador, vamos a crear una vista para la acción índex creada en el paso anterior.

Las vistas en CakePHP únicamente son fragmentos de presentación que encajan dentro de la plantilla (`layout`) de nuestra aplicación. Para la mayoría de aplicaciones son HTML mezclados con PHP, pero bien podrían acabar siendo XML, CSV o incluso datos binarios.

Una plantilla es una presentación de código que envuelve una vista. Se pueden definir múltiples plantillas y puedes cambiar entre ellas pero, por ahora, utilizaremos la plantilla por defecto (`default`).

¿Recuerdas cómo en la sección anterior hemos asignado la variable "articles" a la vista utilizando el método `set()`? Esto asignaría el objeto de consulta (`query object`) a la vista para ser invocado por una iteración `foreach`.

Las vistas en CakePHP se almacenan en la ruta `/src/Template` y en un directorio con el mismo nombre que el controlador al que pertenecen (tendremos que crear una carpeta llamada "Articles" en este caso). Para dar formato a los datos de este artículo en una bonita tabla, el código de nuestra vista debería ser algo así:

```
<!-- File: /templates/Articles/index.php -->

<h1>Artículos</h1>
<table>
    <tr>
        <th>Id</th>
        <th>Title</th>
        <th>Created</th>
    </tr>

    <!-- Aquí es donde iteramos nuestro objeto de consulta $articles, mostrando en␣
↪pantalla la información del artículo -->

    <?php foreach ($articles as $article): ?>
    <tr>
        <td><?= $article->id ?></td>
        <td>
            <?= $this->Html->link($article->title,
            ['controller' => 'Articles', 'action' => 'view', $article->id]) ?>
        </td>
        <td><?= $article->created->format(DATE_RFC850) ?></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 la clase *Cake\View\Helper\HtmlHelper* de CakePHP. CakePHP proporciona un conjunto de ayudantes de vistas (`helpers`) para ayudarte a completar acciones habituales, como por ejemplo crear un enlace o un formulario. Puedes aprender más sobre esto en *Helpers*, pero lo que es importante destacar aquí es que el método `link()` generará un enlace 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 Rutas (`Routes`). Si utilizas las rutas en formato array podrás aprovecharte de las potentes funcionalidades de generación de rutas inversa de CakePHP en el futuro. Además puedes especificar rutas relativas a la base de tu aplicación de la forma `/controlador/accion/param1/param2` o incluso utilizar *Using Named Routes*.

Llegados a este punto, deberías ser capaz de acceder con tu navegador a http://www.example.com/articles/index. Deberías ver tu vista, correctamente formatada con el título y la tabla listando los artículos.

Si te ha dado por hacer clic en uno de los enlaces que hemos creado en esta vista (que enlazan el título de un artículo hacia la URL `/articles/view/un\_id`), seguramente habrás sido informado por CakePHP de que la acción no ha sido definida todavía. Si no has sido infromado, o bien algo ha ido mal o bien ya la habías definido, en cuyo caso eres muy astuto. En caso contrario, la crearemos ahora en nuestro controlador de artículos:

```
namespace App\Controller;

class ArticlesController extends AppController
{

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

    public function view($id = null)
    {
        $article = $this->Articles->get($id);
        $this->set(compact('article'));
    }
}
```

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

Verás que nuestra función view toma un parámetro: el ID del artículo que queremos ver. Este parámetro se gestiona automáticamente al llamar a la URL `/articles/view/3`, el valor "3" se pasa a la función view como primer parámetro `$id`.

También hacemos un poco de verificación de errores para asegurarnos de que el usuario realmente accede a dicho registro. Si el usuario solicita `/articles/view` lanzaremos una excepción `NotFoundException` y dejaremos al ErrorHandler tomar el control. Utilizando el método `get()` en la tabla Articles también hacemos una verificación similar para asegurarnos de que el usuario ha accedido a un registro que existe. En caso de que el artículo solicitado no esté presente en la base de datos, el método `get()` lanzará una excepción `NotFoundException`.

Ahora vamos a definir la vista para esta nueva función "view" ubicándola en **templates/Articles/view.php**.

```
<!-- File: /templates/Articles/view.php -->
<h1><?= h($article->title) ?></h1>
<p><?= h($article->body) ?></p>
<p><small>Created: <?= $article->created->format(DATE_RFC850) ?></small></p>
```

Verifica que esto funciona probando los enlaces en `/articles/index` o puedes solicitándolo manualmente accediendo a `/articles/view/1`.

Añadiendo Artículos

Leer de la base de datos y mostrar nuestros artículos es un gran comienzo, pero permitamos también añadir nuevos artículos.

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

```
namespace App\Controller;

class ArticlesController extends AppController
{
    public $components = ['Flash'];

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

    public function view($id)
    {
        $article = $this->Articles->get($id);
        $this->set(compact('article'));
    }

    public function add()
    {
        $article = $this->Articles->newEmptyEntity();
        if ($this->request->is('post')) {
            $article = $this->Articles->patchEntity($article, $this->request->getData());
            if ($this->Articles->save($article)) {
                $this->Flash->success(__('Your article has been saved.'));

                return $this->redirect(['action' => 'index']);
            }
            $this->Flash->error(__('Unable to add your article.'));
        }
        $this->set('article', $article);
    }
}
```

Nota: Necesitas incluir el FlashComponent en cualquier controlador donde vayas a usarlo. Si lo ves necesario, inclúyelo en tu AppController.

Lo que la función add() hace es: si el formulario enviado no está vacío, intenta salvar un nuevo artículo utilizando el modelo Articles. Si no se guarda bien, muestra la vista correspondiente, así podremos mostrar los errores de validación u otras alertas.

Cada petición de CakePHP incluye un objeto `ServerRequest` que es accesible utilizando `$this->request`. El objeto de petición contiene información útil acerca de la petición que se recibe y puede ser utilizado para controlar el flujo de nuestra aplicación. En este caso, utilizamos el método `Cake\Network\ServerRequest::is()` para verificar que la petición es una petición HTTP POST.

Cuando un usuario utiliza un formulario y efectúa un POST a la aplicación, esta información está disponible en `$this->request->getData()`. Puedes usar la función `pr()` o `debug()` para mostrar el contenido de esa variable y

ver la pinta que tiene.

Utilizamos el método mágico `__call` del `FlashComponent` para guardar un mensaje en una variable de sesión que será mostrado en la página después de la redirección. En la plantilla tenemos `<?= $this->Flash->render() ?>` que muestra el mensaje y elimina la correspondiente variable de sesión. El método *Cake\Controller\Controller::redirect* del controlador redirige hacia otra URL. El parámetro `['action' => 'index']` se traduce a la URL /articles (p.e. la acción index del controlador de artículos). Puedes echar un ojo al método `Cake\Routing\Router::url()` en la API[37] para ver los formatos en que puedes especificar una URL para varias funciones de CakePHP.

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 lo hace más rápido y sencillo.

Para aprovechar estas funciones es conveniente que utilices el FormHelper en tus vistas. La clase *Cake\View\Helper\FormHelper* está disponible en tus vistas por defecto a través de `$this->Form`.

He aquí nuestra vista `add`:

```
<!-- File: templates/Articles/add.php -->

<h1>Añadir Artículo</h1>
<?php
    echo $this->Form->create($article);
    echo $this->Form->input('title');
    echo $this->Form->input('body', ['rows' => '3']);
    echo $this->Form->button(__('Guardar artículo'));
    echo $this->Form->end();
?>
```

Hemos usado FormHelper para generar la etiqueta "form". La ejecución de `$this->Form->create()` genera el siguiente código:

```
<form method="post" action="/articles/add">
```

Si `create()` no tiene parámetros al ser llamado, asume que estás creando un formulario que envía vía POST a la acción `add()` (o `edit()` cuando `id` es incluido en los datos de formulario) del controlador actual.

El método `$this->Form->input()` se utiliza para crear elementos de formulario del mismo nombre. El primer parámetro le indica a CakePHP a qué campo corresponde y el segundo parámetro te permite especificar un abanico muy ámplio de opciones - en este caso, el número de filas del textarea que se generará. Hay un poco de introspección y «automagia» aquí: `input()` generará distintos elementos de formulario en función del campo del modelo especificado.

La llamada a `$this->Form->end()` cierra el formulario. También generará campos ocultos si la CSRF/prevención de manipulación de formularios ha sido habilitada.

Volvamos atrás un minuto y actualicemos nuestra vista **templates/Articles/index.php** para añadir un enlace de «Añadir Artículo». Justo antes del tag <table> añade la siguiente línea:

```
<?= $this->Html->link(
    'Añadir artículo',
```

(continué en la próxima página)

[37] https://api.cakephp.org

(proviene de la página anterior)

```
    ['controller' => 'Articles', '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. Volvamos al modelo `Articles` y hagamos algunos ajustes:

```
namespace App\Model\Table;

use Cake\ORM\Table;
use Cake\Validation\Validator;

class ArticlesTable extends Table
{
    public function initialize(array $config)
    {
        $this->addBehavior('Timestamp');
    }

    public function validationDefault(Validator $validator)
    {
        $validator
            ->notEmpty('title')
            ->notEmpty('body');

        return $validator;
    }
}
```

El método `validationDefault()` le dice a CakePHP cómo validar tus datos cuando se invoca el método `save()`. Aquí hemos especificado que ambos campos, el cuerpo y el título, no pueden quedar vacíos. El motor de validaciones de CakePHP es potente y con numerosas reglas ya predefinidas (tarjetas de crédito, direcciones de e-mail, etc.) así como flexibilidad para añadir tus propias reglas de validación. Para más información en tal configuración, echa un vistazo a la documentación *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 `Cake\View\Helper\FormHelper::input()`, los mensajes de error se construyen automáticamente en la vista sin código adicional.

Editando Artículos

Editando artículos: allá vamos. Ya eres un profesional de CakePHP, así que habrás cogido la pauta. Crear una acción, luego la vista. He aquí cómo debería ser la acción `edit()` del controlador `ArticlesController`:

```
public function edit($id = null)
{
    $article = $this->Articles->get($id);
    if ($this->request->is(['post', 'put'])) {
        $this->Articles->patchEntity($article, $this->request->getData());
        if ($this->Articles->save($article)) {
            $this->Flash->success(__('Tu artículo ha sido actualizado.'));

            return $this->redirect(['action' => 'index']);
```

(continué en la próxima página)

(proviene de la página anterior)

```
        }
        $this->Flash->error(__('Tu artículo no se ha podido actualizar.'));
    }

    $this->set('article', $article);
}
```

Lo primero que hace este método es asegurarse de que el usuario ha intentado acceder a un registro existente. Si no han pasado el parámetro `$id` o el artículo no existe lanzaremos una excepción `NotFoundException` para que el `ErrorHandler` se ocupe de ello.

Luego verifica si la petición es POST o PUT. Si lo es, entonces utilizamos los datos recibidos para actualizar nuestra entidad artículo (`article`) utilizando el método "patchEntity". Finalmente utilizamos el objeto tabla para guardar la entidad de nuevo o mostrar errores de validación al usuario en caso de haberlos.

La vista sería algo así:

```
<!-- File: templates/Articles/edit.php -->

<h1>Edit Article</h1>
<?php
    echo $this->Form->create($article);
    echo $this->Form->input('title');
    echo $this->Form->input('body', ['rows' => '3']);
    echo $this->Form->button(__('Guardar artículo'));
    echo $this->Form->end();
?>
```

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

CakePHP utilizará el resultado de `$article->isNew()` para determinar si un `save()` debería insertar un nuevo registro o actualizar uno existente.

Puedes actualizar tu vista índice (`index`) con enlaces para editar artículos específicos:

```
<!-- File: templates/Articles/index.php  (edit links added) -->

<h1>Artículos</h1>
<p><?= $this->Html->link("Añadir artículo", ['action' => 'add']) ?></p>
<table>
    <tr>
        <th>Id</th>
        <th>Title</th>
        <th>Created</th>
        <th>Action</th>
    </tr>

<!-- Aquí es donde iteramos nuestro objeto de consulta $articles, mostrando en pantalla␣
↪la información del artículo -->

<?php foreach ($articles as $article): ?>
    <tr>
        <td><?= $article->id ?></td>
```

(continué en la próxima página)

(proviene de la página anterior)

```
        <td>
            <?= $this->Html->link($article->title, ['action' => 'view', $article->id]) ?>
        </td>
        <td>
            <?= $article->created->format(DATE_RFC850) ?>
        </td>
        <td>
            <?= $this->Html->link('Editar', ['action' => 'edit', $article->id]) ?>
        </td>
    </tr>
<?php endforeach; ?>

</table>
```

Borrando Artículos

Vamos a permitir a los usuarios que borren artículos. Empieza con una acción `delete()` en el controlador `ArticlesController`:

```
public function delete($id)
{
    $this->request->allowMethod(['post', 'delete']);

    $article = $this->Articles->get($id);
    if ($this->Articles->delete($article)) {
        $this->Flash->success(__('El artículo con id: {0} ha sido eliminado.', h($id)));

        return $this->redirect(['action' => 'index']);
    }
}
```

La lógica elimina el artículo especificado por $id y utiliza `$this->Flash->success()` para mostrar al usuario un mensaje de confirmación tras haber sido redirigidos a `/articles`. Si el usuario intenta eliminar utilizando una petición GET, el "allowMethod" devolvería una Excepción. Las excepciones que no se traten serán capturadas por el manejador de excepciones de CakePHP (`exception handler`) y una bonita página de error es mostrada. Hay muchas *Excepciones* que pueden ser utilizadas para indicar los varios errores HTTP que tu aplicación pueda generar.

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.php para incluír el ya habitual enlace:

```
<!-- File: templates/Articles/index.php -->

<h1>Artículos</h1>
<p><?= $this->Html->link("Añadir artículo", ['action' => 'add']) ?></p>
<table>
    <tr>
        <th>Id</th>
        <th>Title</th>
        <th>Created</th>
        <th>Action</th>
    </tr>
```

(continué en la próxima página)

(proviene de la página anterior)

```
<!-- Aquí es donde iteramos nuestro objeto de consulta $articles, mostrando en pantalla
↪la información del artículo -->

<?php foreach ($articles as $article): ?>
    <tr>
        <td><?= $article->id ?></td>
        <td>
            <?= $this->Html->link($article->title, ['action' => 'view', $article->id]) ?>
        </td>
        <td>
            <?= $article->created->format(DATE_RFC850) ?>
        </td>
        <td>
            <?= $this->Form->postLink(
                'Eliminar',
                ['action' => 'delete', $article->id],
                ['confirm' => '¿Estás seguro?'])
            ?>
            <?= $this->Html->link('Editar', ['action' => 'edit', $article->id]) ?>
        </td>
    </tr>
<?php endforeach; ?>

</table>
```

Utilizando `postLink()` crearemos un enlace que utilizará JavaScript para hacer una petición POST que eliminará nuestro artículo. Permitiendo que contenido sea eliminado vía peticiones GET es peligroso, ya que arañas web (`crawlers`) podrían eliminar accidentalmente tu contenido.

Nota: Esta vista utiliza el FormHelper para pedir confirmación vía diálogo de confirmación de JavaScript al usuario antes de borrar un artículo.

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 así como técnicas avanzadas revisa *Connecting Routes*.

Por defecto CakePHP responde a las llamadas a la raíz de tu sitio (por ejemplo http://www.example.com) usando el controlador PagesController, mostrando una vista llamada «home». En lugar de eso, lo reemplazaremos con nuestro controlador `ArticlesController` creando una nueva ruta.

Las reglas de enrutamiento están en **config/routes.php**. Querrás eliminar o comentar la línea que define la raíz por defecto. Dicha ruta se parece a esto:

```
Router::connect('/', ['controller' => 'Pages', 'action' => 'display', 'home']);
```

Esta línea conecta la url "/" con la página por defecto de inicio de CakePHP. Queremos conectarla a nuestro propio controlador, así que reemplaza dicha línea por esta otra:

```
Router::connect('/', ['controller' => 'Articles', 'action' => 'index']);
```

Esto debería, cuando un usuario solicita "/", devolver la acción index() del controlador ArticlesController.

Nota: CakePHP también calcula las rutas a la inversa. Si en tu código pasas el array `['controller' => 'Articles', '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, honor, amor, dinero a carretas e incluso tus fantasías más salvajes. Simple, no te parece? Ten en cuenta que este tutorial es muy básico, CakePHP tiene *muchas* otras cosas que ofrecer 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.

Ahora que ya has creado una aplicación CakePHP básica, estás listo para la vida real. Empieza tu nuevo proyecto y lee el resto del Cookbook así como la API[38].

Si necesitas ayuda, hay muchos modos de encontrar la ayuda que buscas - por favor, míralo en la página *Donde obtener ayuda*. ¡Bienvenido a CakePHP!

Lectura sugerida para continuar desde aquí

Hay varias tareas comunes que la gente que está aprendiendo CakePHP quiere aprender después:

1. *Layouts*: Personaliza la plantilla *layout* de tu aplicación
2. *Elementos* Incluír vistas y reutilizar trozos de código
3. /bake/usage: Generación básica de CRUDs
4. *Tutorial Blog - Autenticación y Autorización*: Tutorial de autenticación y permisos

Tutorial Blog - Parte 3

Crear categorias en Arbol

Vamos a continuar con nuestro blog e imaginar que queremos categorizar nuestros articulos. Queremos que las categorias estén ordenadas, y para esto, vamos a usar *Tree behavior* para ayudarnos a organizar las categorías.

Pero primero necesitamos modificar nuestras tablas.

[38] https://api.cakephp.org

Plugin de migración

Vamos a usar el migrations plugin[39] para crear una tabla en nuestra base de datos. Si tienes una tabla de articulos en tu base de datos, borrala.

Abre tu archivo `composer.json`. Generalmente el plugin de migración ya esta incluido en `require`. Si no es el caso, agrégalo:

```
"require": {
    "cakephp/migrations": "~1.0"
}
```

Luego corre el comando `composer update`. El plugin de migración se alojara en tu carpeta de `plugins`. Agrega también `Plugin::load('Migrations');` en el archivo bootstrap.php de tú aplicación.

Una vez que el plugin sea cargado, corre el siguiente comando para crear el archivo de migración:

```
bin/cake migrations create Initial
```

Un archivo de migración será creado en la carpeta `/config/Migrations`. Puedes abrir tu archivo y agregar las siguientes lineas:

```
<?php

use Phinx\Migration\AbstractMigration;

class Initial extends AbstractMigration
{
    public function change()
    {
        $articles = $this->table('articles');
        $articles->addColumn('title', 'string', ['limit' => 50])
            ->addColumn('body', 'text', ['null' => true, 'default' => null])
            ->addColumn('category_id', 'integer', ['null' => true, 'default' => null])
            ->addColumn('created', 'datetime')
            ->addColumn('modified', 'datetime', ['null' => true, 'default' => null])
            ->save();

        $categories = $this->table('categories');
        $categories->addColumn('parent_id', 'integer', ['null' => true, 'default' =>␣
↪null])
            ->addColumn('lft', 'integer', ['null' => true, 'default' => null])
            ->addColumn('rght', 'integer', ['null' => true, 'default' => null])
            ->addColumn('name', 'string', ['limit' => 255])
            ->addColumn('description', 'string', ['limit' => 255, 'null' => true,
↪'default' => null])
            ->addColumn('created', 'datetime')
            ->addColumn('modified', 'datetime', ['null' => true, 'default' => null])
            ->save();
    }
}
```

Ahora corre el siguiente comando para crear tús tablas:

[39] https://github.com/cakephp/migrations

```
bin/cake migrations migrate
```

Modificando las tablas

Con nuestras tablas creadas, ahora podemos enfocarnos en categorizar los artículos.

Suponemos que ya tienes los archivos (Tables, Controllers y Templates de Articles) de la parte 2 de esta serie de tutoriales, por lo que solamente vamos a agregar referencia a las categorías.

Necesitamos asociar las tablas de Articles y Categories. Abre el archivo **src/Model/Table/ArticlesTable.php** y agrega las siguientes lineas:

```
// src/Model/Table/ArticlesTable.php

namespace App\Model\Table;

use Cake\ORM\Table;

class ArticlesTable extends Table
{
    public function initialize(array $config)
    {
        $this->addBehavior('Timestamp');
        // Just add the belongsTo relation with CategoriesTable
        $this->belongsTo('Categories', [
            'foreignKey' => 'category_id',
        ]);
    }
}
```

Generando el código base para las Categorías

Crea todos los archivos corriendo los siguientes comandos:

```
bin/cake bake model Categories
bin/cake bake controller Categories
bin/cake bake template Categories
```

La herramienta bake ha creado todos los archivos en un instante. Puedes darles una rápida leida si necesitas re-familiarizarte con la forma en la que CakePHP funciona.

Nota: Si estás en Windows recordá usar en lugar de / .

Agregar el TreeBehavior a CategoriesTable

TreeBehavior ayuda a manejar estructuras de árbol jerarquica en una tabla. Utiliza MPTT logic[40] para manejar los datos. Las estructuras en árbol MPTT están optimizadas para lecturas, lo cual las hace ideal para aplicaciones con gran carga de lectura como los blogs.

Si abres el archivo **src/Model/Table/CategoriesTable.php** veras que el TreeBehavior fue agregado a CategoriesTable en el método `initialize()`. Bake agrega este behavior a cualquier tabla que contenga las columnas `lft` y `rght`:

```
$this->addBehavior('Tree');
```

Con el TreeBehavior agregado ahora podras acceder a algunas funcionalidades como reordenar las categorias. Veremos eso en un momento.

Pero por ahora tendrás que removar los siguientes inputs en tus archivos add y edit de Categories:

```
echo $this->Form->input('lft');
echo $this->Form->input('rght');
```

Esos campos son manejados automáticamento por el TreeBehavior cuando una categoría es guardada.

Con tú navegador, agrega alguna nueva categoría usando la acción `/yoursite/categories/add`.

Reordenando categorías con TreeBehavior

En el index de categorias, puedes listar y re-ordenar categorias.

Vamos a modificar el método index en tu `CategoriesController.php`, agregando `move_up()` y `move_down()` para poder reordenar las categorías en ese árbol:

```
class CategoriesController extends AppController
{
    public function index()
    {
        $categories = $this->Categories->find('threaded')
            ->order(['lft' => 'ASC']);
        $this->set(compact('categories'));
    }

    public function move_up($id = null)
    {
        $this->request->allowMethod(['post', 'put']);
        $category = $this->Categories->get($id);
        if ($this->Categories->moveUp($category)) {
            $this->Flash->success('The category has been moved Up.');
        } else {
            $this->Flash->error('The category could not be moved up. Please, try again.
↪');
        }

        return $this->redirect($this->referer(['action' => 'index']));
    }
```

(continué en la próxima página)

[40] https://www.sitepoint.com/hierarchical-data-database-2/

(proviene de la página anterior)

```
    public function move_down($id = null)
    {
        $this->request->allowMethod(['post', 'put']);
        $category = $this->Categories->get($id);
        if ($this->Categories->moveDown($category)) {
            $this->Flash->success('The category has been moved down.');
        } else {
            $this->Flash->error('The category could not be moved down. Please, try again.
↪');
        }

        return $this->redirect($this->referer(['action' => 'index']));
    }
}
```

En **templates/Categories/index.php** reemplazá el contenido existente por el siguiente:

```
<div class="actions columns large-2 medium-3">
    <h3><?= __('Actions') ?></h3>
    <ul class="side-nav">
        <li><?= $this->Html->link(__('New Category'), ['action' => 'add']) ?></li>
    </ul>
</div>
<div class="categories index large-10 medium-9 columns">
    <table cellpadding="0" cellspacing="0">
    <thead>
        <tr>
            <th>id</th>
            <th>Parent Id</th>
            <th>Title</th>
            <th>Lft</th>
            <th>Rght</th>
            <th>Name</th>
            <th>Description</th>
            <th>Created</th>
            <th class="actions"><?= __('Actions') ?></th>
        </tr>
    </thead>
    <tbody>
    <?php foreach ($categories as $category): ?>
        <tr>
            <td><?= $this->Number->format($category->id) ?></td>
            <td><?= $this->Number->format($category->parent_id) ?></td>
            <td><?= $this->Number->format($category->lft) ?></td>
            <td><?= $this->Number->format($category->rght) ?></td>
            <td><?= h($category->name) ?></td>
            <td><?= h($category->description) ?></td>
            <td><?= h($category->created) ?></td>
            <td class="actions">
                <?= $this->Html->link(__('View'), ['action' => 'view', $category->id]) ?>
                <?= $this->Html->link(__('Edit'), ['action' => 'edit', $category->id]) ?>
                <?= $this->Form->postLink(__('Delete'), ['action' => 'delete', $category-
```

(continué en la próxima página)

(proviene de la página anterior)

```
↪>id], ['confirm' => __('Are you sure you want to delete # {0}?', $category->id)]) ?>
                <?= $this->Form->postLink(__('Move down'), ['action' => 'move_down',
↪$category->id], ['confirm' => __('Are you sure you want to move down # {0}?',
↪$category->id)]) ?>
                <?= $this->Form->postLink(__('Move up'), ['action' => 'move_up',
↪$category->id], ['confirm' => __('Are you sure you want to move up # {0}?', $category->
↪id)]) ?>
            </td>
        </tr>
    <?php endforeach; ?>
    </tbody>
    </table>
</div>
```

Modificando el ArticlesController

En tú `ArticlesController`, vamos a obtener el listado de categorías. Esto nos permitirá elegir una categoría para un Article al momento de crearlo o editarlo:

```
// src/Controller/ArticlesController.php

namespace App\Controller;

// Prior to 3.6 use Cake\Network\Exception\NotFoundException
use Cake\Http\Exception\NotFoundException;

class ArticlesController extends AppController
{

    // ...

    public function add()
    {
        $article = $this->Articles->newEmptyEntity();
        if ($this->request->is('post')) {
            $article = $this->Articles->patchEntity($article, $this->request->getData());
            if ($this->Articles->save($article)) {
                $this->Flash->success(__('Your article has been saved.'));

                return $this->redirect(['action' => 'index']);
            }
            $this->Flash->error(__('Unable to add your article.'));
        }
        $this->set('article', $article);

        // Just added the categories list to be able to choose
        // one category for an article
        $categories = $this->Articles->Categories->find('treeList');
        $this->set(compact('categories'));
    }
}
```

Modificando el template de Articles

El template add de Article debería verse similar a esto:

```
.. code-block:: php
```

<!– File: templates/Articles/add.php –>

<h1>Add Article</h1> <?php echo $this->Form->create($article); // just added the categories input echo $this->Form->input("categories"); echo $this->Form->input("title"); echo $this->Form->input("body", ["rows" => "3"]); echo $this->Form->button(__("Save Article")); echo $this->Form->end();

Ingresando a `/yoursite/categories/add` deberías ver una lista de categorías para elegir.

Tutorial Blog - Autenticación y Autorización

Siguiendo con nuestro ejemplo de aplicacion *Tutorial 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,
    email VARCHAR(255),
    password VARCHAR(255),
    role VARCHAR(20),
    created DATETIME DEFAULT NULL,
    modified DATETIME DEFAULT NULL
);
```

Siguimos las convenciones de CakePHP para nombrar tablas pero también estamos aprovechando otra convencion: al usar los campos email y password en nuestra tabla CakePHP configurará automáticamente la mayoria de las cosas al momento de implementar el login.

El siguiente paso es crear Users table, responsable de buscar, guardar y validar los datos de usuario:

```
// src/Model/Table/UsersTable.php
namespace App\Model\Table;

use Cake\ORM\Table;
use Cake\Validation\Validator;

class UsersTable extends Table
{

    public function validationDefault(Validator $validator)
    {
        return $validator
            ->notEmpty('email', 'A email is required')
            ->email('email')
```

(continué en la próxima página)

(proviene de la página anterior)

```
            ->notEmpty('password', 'A password is required')
            ->notEmpty('role', 'A role is required')
            ->add('role', 'inList', [
                'rule' => ['inList', ['admin', 'author']],
                'message' => 'Please enter a valid role'
            ]);
    }

}
```

También vamos a crear UsersController; el siguiente contenido fue generado usando baked UsersController con el generador de código incluído con CakePHP:

```
// src/Controller/UsersController.php

namespace App\Controller;

use App\Controller\AppController;
use Cake\Event\Event;
// Prior to 3.6 use Cake\Network\Exception\NotFoundException
use Cake\Http\Exception\NotFoundException;

class UsersController extends AppController
{

    public function beforeFilter(Event $event)
    {
        parent::beforeFilter($event);
        $this->Auth->allow('add');
    }

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

    public function view($id)
    {
        if (!$id) {
            throw new NotFoundException(__('Invalid user'));
        }

        $user = $this->Users->get($id);
        $this->set(compact('user'));
    }

    public function add()
    {
        $user = $this->Users->newEntity();
        if ($this->request->is('post')) {
            $user = $this->Users->patchEntity($user, $this->request->getData());
            if ($this->Users->save($user)) {
```

(continué en la próxima página)

(proviene de la página anterior)

```
                $this->Flash->success(__('The user has been saved.'));

                return $this->redirect(['action' => 'add']);
            }
            $this->Flash->error(__('Unable to add the user.'));
        }
        $this->set('user', $user);
    }

}
```

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.php:

```
<!-- templates/Users/add.php -->

<div class="users form">
<?= $this->Form->create($user) ?>
    <fieldset>
        <legend><?= __('Add User') ?></legend>
        <?= $this->Form->input('email') ?>
        <?= $this->Form->input('password') ?>
        <?= $this->Form->input('role', [
            'options' => ['admin' => 'Admin', 'author' => 'Author']
        ]) ?>
   </fieldset>
<?= $this->Form->button(__('Submit')); ?>
<?= $this->Form->end() ?>
</div>
```

Autenticación (login y logout)

Ya estamos listos para agregar nuestra autenticación. En CakePHP esto es manejado por `Cake\Controller\Component\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 abre el archivo **src/Controller/AppController.php** y agrega las siguientes lineas:

```
// src/Controller/AppController.php

namespace App\Controller;

use Cake\Controller\Controller;
use Cake\Event\Event;

class AppController extends Controller
{
    //...

    public function initialize()
    {
```

(continué en la próxima página)

(proviene de la página anterior)

```
        $this->loadComponent('Flash');
        $this->loadComponent('Auth', [
            'loginRedirect' => [
                'controller' => 'Articles',
                'action' => 'index'
            ],
            'logoutRedirect' => [
                'controller' => 'Pages',
                'action' => 'display',
                'home'
            ]
        ]);
    }

    public function beforeFilter(Event $event)
    {
        $this->Auth->allow(['index', 'view', 'display']);
    }
    //...
}
```

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 `/articles/` 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:

```
// src/Controller/UsersController.php

public function beforeFilter(Event $event)
{
    parent::beforeFilter($event);
    // Allow users to register and logout.
    // You should not add the "login" action to allow list. Doing so would
    // cause problems with normal functioning of AuthComponent.
    $this->Auth->allow(['add', 'logout']);
}

public function login()
{
    if ($this->request->is('post')) {
        $user = $this->Auth->identify();
        if ($user) {
            $this->Auth->setUser($user);

            return $this->redirect($this->Auth->redirectUrl());
        }
        $this->Flash->error(__('Invalid email or password, try again'));
    }
```

(continué en la próxima página)

(proviene de la página anterior)

```
}

public function logout()
{
    return $this->redirect($this->Auth->logout());
}
```

El hasheo del password aún no está hecho, necesitamos una clase Entity para nuestra clase User para así manejar esta lógica específica. Crea el archivo **src/Model/Entity/User.php** y agrega las siguientes lineas:

```
// src/Model/Entity/User.php
namespace App\Model\Entity;

use Cake\Auth\DefaultPasswordHasher;
use Cake\ORM\Entity;

class User extends Entity
{

    // Make all fields mass assignable for now.
    protected $_accessible = ['*' => true];

    // ...

    protected function _setPassword($password)
    {
        if (strlen($password) > 0) {
            return (new DefaultPasswordHasher)->hash($password);
        }
    }

    // ...
}
```

Ahora cada vez que la propiedad password sea asignada a un usuario, será hasheada usando la clase `DefaultPasswordHasher`. Solamente nos falta un archivo para la vista de la acción login. Abre tu archivo **templates/Users/login.php** y agrega las siguientes lineas:

```
<!-- File: templates/Users/login.php -->

<div class="users form">
<?= $this->Flash->render('auth') ?>
<?= $this->Form->create() ?>
    <fieldset>
        <legend><?= __('Please enter your email and password') ?></legend>
        <?= $this->Form->input('email') ?>
        <?= $this->Form->input('password') ?>
    </fieldset>
<?= $this->Form->button(__('Login')); ?>
<?= $this->Form->end() ?>
</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 `/articles/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->identify()` del AuthComponent, y funciona sin ninguna otra configuración ya que seguimos la convención. Es decir, tener un modelo llamado User con los campos email 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 articles ADD COLUMN user_id INT(11);
```

También, un pequeño cambio en ArticlesController es necesario para guardar el usuario logeado como referencia en los artículos creados:

```
// src/Controller/ArticlesController.php

public function add()
{
    $article = $this->Articles->newEmptyEntity();
    if ($this->request->is('post')) {
        $article = $this->Articles->patchEntity($article, $this->request->getData());
        // Added this line
        $article->user_id = $this->Auth->user('id');
        // You could also do the following
        //$newData = ['user_id' => $this->Auth->user('id')];
        //$article = $this->Articles->patchEntity($article, $newData);
        if ($this->Articles->save($article)) {
            $this->Flash->success(__('Your article has been saved.'));

            return $this->redirect(['action' => 'index']);
        }
        $this->Flash->error(__('Unable to add your article.'));
    }
    $this->set('article', $article);
}
```

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 artículos 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. Abre nuevamente AppController y agregá las siguientes opciones en la configuración del Auth:

```
// src/Controller/AppController.php

public function initialize()
{
    $this->loadComponent('Flash');
    $this->loadComponent('Auth', [
        'authorize' => ['Controller'], // Added this line
        'loginRedirect' => [
            'controller' => 'Articles',
            'action' => 'index'
        ],
        'logoutRedirect' => [
            'controller' => 'Pages',
            'action' => 'display',
            'home'
        ]
    ]);
}

public function isAuthorized($user)
{
    // Admin can access every action
    if (isset($user['role']) && $user['role'] === 'admin') {
        return true;
    }

    // Default deny
    return false;
}
```

Hemos creado 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 ArticlesController deberian permitirle a los autores crear artículos, pero prevenir que editen artículos que no le pertenezcan. Abre el archivo ArticlesController.php y agregá las siguientes lineas:

```
// src/Controller/ArticlesController.php

public function isAuthorized($user)
{
    // All registered users can add articles
    if ($this->request->getParam('action') === 'add') {
        return true;
    }

    // The owner of an article can edit and delete it
    if (in_array($this->request->getParam('action'), ['edit', 'delete'])) {
        $articleId = (int)$this->request->getParam('pass.0');
        if ($this->Articles->isOwnedBy($articleId, $user['id'])) {
            return true;
```

(continué en la próxima página)

(proviene de la página anterior)

```
        }
    }

    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 Articles. Es en general una buena practica mover la mayor parte de la logica posible hacia los modelos:

```
// src/Model/Table/ArticlesTable.php

public function isOwnedBy($articleId, $userId)
{
    return $this->exists(['id' => $articleId, 'user_id' => $userId]);
}
```

Esto concluye nuestro simple tutorial de autenticación y autorización. Para proteger el UsersController se puede seguir la misma técnica utilizada para ArticlesController. 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 *Authentication*, donde encontrarás mas información para configurar el componente y crear clases de autorizacion a tú medida.

Lectura sugerida

1. /bake/usage Generar código CRUD básico
2. *Authentication*: Registro y login de usuarios

CAPÍTULO 5

Contribuir

Existen diversas maneras con las que puedes contribuir a CakePHP:

Documentación

Contribuir con la documentación es fácil. Los archivos están hospedados en https://github.com/cakephp/docs. Siéntete libre de hacer un *fork* del repositorio, añadir tus cambios, mejoras, traducciones y comenzar a ayudar a través de un nuevo *pull request*. También puedes editar los archivos de manera online con GitHub sin la necesidad de descargarlos – el botón *Improve this Doc* que aparece en todas las páginas te llevará al editor online de GitHub de esa página.

La documentación de CakePHP dispone de integración continua[41] y se despliega automáticamente tras realizar el *merge* del *pull request*.

Traducciones

Envía un email al equipo de documentación (docs *arroba* cakephp *punto* org) o utiliza IRC (#cakephp en *freenode*) para hablar de cualquier trabajo de traducción en el que quieras participar.

[41] https://es.wikipedia.org/wiki/Integraci%C3%B3n_continua

Nueva traducción

Nos gustaría poder disponer de traducciones que estén todo lo completas posible. Sin embargo hay ocasiones donde un archivo de traducción no está al día, por lo que debes considerar siempre la versión en inglés como la versión acreditada.

Si tu idioma no está entre los disponibles, por favor, contacta con nosotros a través de Github y estudiaremos la posibilidad de crear la estructura de archivos para ello.

Las siguientes secciones son las primeras que deberías considerar traducir ya que estos archivos no cambian a menudo:

- index.rst
- intro.rst
- quickstart.rst
- installation.rst
- /intro (carpeta)
- /tutorials-and-examples (carpeta)

Recordatorio para administradores de documentación

La estructura de archivos de todos los idiomas deben seguir la estructura de la versión en inglés. Si la estructura cambia en esta versión debemos realizar dichos cambios en los demás idiomas.

Por ejemplo, si se crea un nuevo archivo en inglés en **en/file.rst** tendremos que:

- Añadir el archivo en todos los idiomas: **fr/file.rst**, **zh/file.rst**,...
- Borrar el contenido pero manteniendo el `title`, `meta` información y `toc-tree` que pueda haber. Se añadirá la siguiente nota mientras nadie traduzca el archivo:

```
File Title
##########

.. note::
    The documentation is not currently supported in XX language for this
    page.

    Please feel free to send us a pull request on
    `Github <https://github.com/cakephp/docs>`_ or use the **Improve This Doc**
    button to directly propose your changes.

    You can refer to the English version in the select top menu to have
    information about this page's topic.

// If toc-tree elements are in the English version
.. toctree::
    :maxdepth: 1

    one-toc-file
    other-toc-file

.. meta::
    :title lang=xx: File Title
    :keywords lang=xx: title, description,...
```

Consejos para traductores

- Navega y edita en el idioma al que quieras traducir el contenido - de otra manera no verás lo que ya está traducido.
- Siéntete libre de bucear en la traducción si ya existe en tu idioma.
- Usa la Forma informal[42].
- Traduce el título y el contenido a la vez.
- Compara con la versión en inglés antes de subir una corrección (si corriges algo pero no indicas una referencia tu subida no será aceptada).
- Si necesitas escribir un término en inglés envuélvelo en etiquetas `<em>`. E.g. «asdf asdf *Controller* asdf» o «asdf asdf Kontroller (*Controller*) asfd» como proceda.
- No subas traducciones parciales.
- No edites una sección con cambios pendientes.
- No uses entidades HTML[43] para caracteres acentudados, la documentación utiliza UTF-8.
- No cambies significatibamente el etiquetado (HTML) o añadas nuevo contenido.
- Si falta información en el contenido original sube primero una corrección de ello.

Guía de formato para la documentación

La nueva documentación de CakePHP está escrito con texto en formato ReST[44].

ReST (*Re Structured Text*) es una sintaxis de marcado de texto plano similar a *Markdown* o *Textile*.

Para mantener la consistencia cuando añadas algo a la documentación de CakePHP recomendamos que sigas las siguientes líneas guía sobre como dar formato y estructurar tu texto.

Tamaño de línea

Las líneas de texto deberían medir como máximo 40 caracteres. Las únicas excepciones son URLs largas y fragmentos de código.

Cabeceras y secciones

Las cabeceras de las secciones se crean subrayando el título con caracteres de puntuación. El subrayado deberá ser por lo menos tan largo como el texto.

- # Se utiliza para indicar los títulos de páginas.
- = Se utiliza para los títulos de las secciones de una página.
- - Se utiliza para los títulos de subsecciones.
- ~ Se utiliza para los títulos de sub-subsecciones.
- ^ Se utiliza para los títulos de sub-sub-subsecciones.

Los encabezados no deben anidarse con más de 5 niveles de profundidad y deben estar precedidos y seguidos por una línea en blanco.

[42] https://es.wikipedia.org/wiki/Registro_ling%C3%BC%C3%ADstico
[43] https://es.wikipedia.org/wiki/Anexo:Entidades_de_caracteres_XML_y_HTML
[44] https://es.wikipedia.org/wiki/ReStructuredText

Párrafos

Párrafos son simplemente bloques de texto con todas las líneas al mismo nivel de indexación. Los párrafos deben separarse por al menos una línea vacía.

Marcado en línea

- Un asterisco: *texto* en cursiva. Lo usaremos para enfatizar/destacar de forma general.
  - `*texto*`.
- Dos astericos: **texto** en negrita. Lo usaremos para indicar directorios de trabajo, títulos de listas y nombres de tablas (excluyendo la palabra *table*).
  - `**/config/Migrations**`, `**articulos**`, etc.
- Dos acentos graves (``): `texto` para ejemplos de código. Lo usaramos para nombres de opciones de métodos, columnas de tablas, objetos (excluyendo la palabra «objeto») y para nombres de métodos y funciones (incluídos los paréntesis)
  - ``` ``cascadeCallbacks`` ```, ``` ``true`` ```, ``` ``id`` ```, ``` ``PagesController`` ```, ``` ``config()`` ```, etc.

Si aparecen asteriscos o acentos graves en el texto y pueden ser confundidos con los delimitadores de marcado habrá que escaparlos con \.

Los marcadores en línea tienen algunas restricciones:

- **No pueden** estar anidados.
- El contenido no puede empezar o acabar con espacios en blanco: `* texto*` está mal.
- El contenido debe separarse del resto del texto por caracteres que no sean palabras. Utiliza \ para escapar un espacio y solucionarlo: `onelong\ *bolded*\ word`.

Listas

El etiquetado de listas es muy parecido a *Markdown*. Las listas no ordenadas se indican empezando una línea con un asterisco y un espacio.

Las listas enumeradas pueden crearse con enumeraciones o # para auto enumeración:

- **Esto es una viñeta**
  - Esto también, pero esta línea tiene dos líneas.

1. **Primera línea**
   2. Segunda línea
2. La enumeración automática
3. Te ahorrará algo de tiempo.

También se pueden crear listas anidadas tabulando secciones y separándolas con una línea en blanco:

```
* Primera línea
* Segunda línea

    * Bajando un nivel
    * Yeah!
```

(continué en la próxima página)

(proviene de la página anterior)

```
* Volviendo al primer nivel
```

Pueden crearse listas de definiciones haciendo lo siguiente:

```
Término
    Definición
CakePHP
    Un framework MVC para PHP
```

Los términos no pueden ocupar más de una línea pero las definiciones pueden ocupar más líneas mientras se aniden consistentemente.

Enlaces

Hay diferentes tipos de enlaces, cada uno con sus características.

Enlaces externos

Los enlaces a documentos externos pueden hacerse de la siguiente manera:

```
`Enlace externo a php.net <https://php.net>`_
```

El resultado debería verse así: Enlace externo a php.net[45]

Enlaces a otras páginas

`:doc:`

Puedes crear enlaces a otras páginas de la documentación usando la función `::doc:`. Puedes enlazar a un archivo específico empleando rutas relativas o absolutas omitiendo la extensión `.rst`. Por ejemplo: si apareciese `:doc:`form`` en el documento `core-helpers/html`, el enlace haría referencia a `core-helpers/form`. Si la referencia fuese `:doc:`/core-helpers`` el enlace sería siempre a `/core-helpers` sin importar donde se utilice.

Enlaces a referencias cruzadas

`:ref:`

Puedes hacer referncia cruzada a cualquier título de cualquier documento usando la función `:ref:`. Los enlaces a etiquetas de destino deben ser únicos a lo largo de toda la documentación. Cuando se crean etiquetas para métodos de clase lo mejor es usar `clase-método` como formato para tu etiqueta de destino.

El uso más habitual de etiquetas es encima de un título. Ejemplo:

```
.. _nombre-etiqueta:

Título sección
--------------
```

(continué en la próxima página)

[45] https://php.net

(proviene de la página anterior)

```
Resto del contenido.
```

En otro sitio podrías enlazar a la sección de arriba usando `:ref:`nombre-etiqueta``. El texto del enlace será el título al que precede el enlace pero puedes personalizarlo usando `:ref:`Texto del enlace <nombre-etiqueta>``.

Evitar alertas de Sphinx

Sphinx mostrará avisos si un archivo no es referenciado en un *toc-tree*. Es una buena manera de asegurarse de que todos los archivos tienen un enlace dirigido a ellos. Pero a veces no necesitas introducir un enlace a un archivo, p.ej. para nuestros archivos *epub-contents* y *pdf-contents*. En esos casos puedes añadir `:orphan:` al inicio del archivo para eliminar las alertas de que el archivo no está en el *toc-tree*

Describir clases y sus contenidos

La documentación de CakePHP usa el phpdomain[46] para proveer directivas personalizadas para describir objetos PHP y constructores. El uso de estas directivas y funciones es necesario para una correcta indexación y uso de las herramientas de referenciación cruzada.

Describir clases y constructores

Cada directiva introduce el contenido del índice y/o índice del *namespace*.

`.. php:global:: nombre`
: Esta directiva declara una nueva variable PHP global.

`.. php:function:: nombre(firma)`
: Define una nueva función global fuera de una clase.

`.. php:const:: nombre`
: Esta directiva declara una nueva constante PHP, puedes usarla también anidada dentro de una directiva de clase para crear constantes de clase.

`.. php:exception:: nombre`
: Esta directiva declara una nueva excepción en el *namespace* actual. La firma puede incluir argumentos de constructor.

`.. php:class:: nombre`
: Describe una clase. Métodos, atributos y atributos que pertenezcan a la clase deberán ir dentro del cuerpo de la directiva:

```
.. php:class:: MyClass

    Descripción de la clase

   .. php:method:: method($argument)

   Descripción del método
```

[46] https://pypi.org/project/sphinxcontrib-phpdomain/

Atributos, métodos y constantes no necesitan estar anidados, pueden seguir la siguiente declaración de clase:

```
.. php:class:: MyClass

    Texto sobre la clase

.. php:method:: methodName()

    Texto sobre el método
```

.. php:method:: nombre(firma)

Describe un método de clase, sus argumentos, salida y excepciones:

```
.. php:method:: instanceMethod($one, $two)

    :param string $one: El primer parámetro.
    :param string $two: El segundo parámetro.
    :returns: Un array de cosas
    :throws: InvalidArgumentException

   Esto es una instancia de método.
```

.. php:staticmethod:: ClassName::nombreMetodo(firma)

Describe un método estático, sus argumentos, salida y excepciones, ver *php:method* para opciones.

.. php:attr:: nombre

Describe una propiedad/atributo en una clase.

Evitar avisos de Sphinx

Sphinx mostrará avisos si una función es referenciada en múltiples archivos. Es una buena manera de asegurarse de que no añades una función dos veces, pero algunas veces puedes querer escribir una función en dos o más archivos, p.ej. *"debug object"* es referenciado en *`/development/debugging`* y *`/core-libraries/global-constants-and-functions`*. En este caso tu puedes añadir **:noindex:** debajo de la función *debug* para eliminar los avisos. Mantén únicamente una referencia **sin** **:no-index:** para seguir teniendo la función referenciada:

```
.. php:function:: debug(mixed $var, boolean $showHtml = null, $showFrom = true)
    :noindex:
```

Referencias cruzadas

Los siguientes *roles* hacen referencia a objetos PHP y los enlaces son generados si se encuentra una directiva que coincida:

:php:func:

Referencia a una función PHP.

:php:global:

Referencia a una variable global cuyo nombre tiene prefijo $.

:php:const:

Referencia tanto a una constante global como a una de clase. Las constantes de clase deberán ir precedidas por la clase que las contenga:

```
DateTime tiene una constante :php:const:`DateTime::ATOM`.
```

:php:class:

Referencia una clase por el nombre:

```
:php:class:`ClassName`
```

:php:meth:

Referencia un método de una clase. Este *role* soporta ambos tipos de métodos:

```
:php:meth:`DateTime::setDate`
:php:meth:`Classname::staticMethod`
```

:php:attr:

Referencia una propiedad de un objeto:

```
:php:attr:`ClassName::$propertyName`
```

:php:exc:

Referencia una excepción.

Código fuente

Los bloques de citas de código fuente se crean finalizando un párrafo con `::`. El bloque debe ir anidado y, como todos los párrafos, separados por líneas en blanco:

```
Esto es un párrafo::

    while ($i--) {
        doStuff()
    }

Esto es otra vez texto normal.
```

Los textos citados no son modificados ni formateados salvo el primer nivel de anidamiento, que es eliminado.

Notas y avisos

Hay muchas ocasiones en las que quieres avisar al lector de un consejo importante, una nota especial o un peligro potencial. Las admonestaciones en *Sphinx* se utilizan justo para eso. Hay cinco tipos de admonestaciones:

- `.. tip::` Los consejos (*tips*) se utilizan para documentar o reiterar información interesante o importante. El contenido de la directiva debe escribirse en sentencias completas e incluir todas las puntuaciones apropiadas.
- `.. note::` Las notas (*notes*) se utilizan para documentar una pieza de información importante. El contenido de la directiva debe escribirse en sentencias completas e incluir todas las puntuaciones apropiadas.
- `.. warning::` Avisos (*warnings*) se utilizan para documentar posibles obstáculos o información relativa a seguridad. El contenido de la directiva debe escribirse en sentencias completas e incluir todas las puntuaciones apropiadas.
- `.. versionadded:: X.Y.Z` las admonestaciones *«Version added»* se utilizan para mostrar notas específicas a nuevas funcionalidades añadidas en una versión específica, siendo `X.Y.Z` la versión en la que se añadieron.

- `.. deprecated:: X.Y.Z` es lo opuesto a *versionadded*, se utiliza para avisar de una funcionalidad obsoleta, siendo `X.Y.Z` la versión en la que pasó a ser obsoleta.

Todas las admonestaciones se escriben igual:

```
.. note::

    Anidado y precedido por una línea en blanco.
    Igual que un párafo.

Este texto no es parte de la nota.
```

Ejemplos

Truco: Esto es un consejo útil que probablemente hayas olvidado.

Nota: Deberías prestar atención aquí.

Advertencia: Podría ser peligroso.

Nuevo en la versión 4.0.0: Esta funcionalidad tan genial fue añadida en la versión 4.0.0

Obsoleto desde la versión 4.0.1: Esta antigua funcionalidad pasó a ser obsoleta en la versión 4.0.1

Tickets

Aportar *feedback* y ayudar a la comunidad en la forma de tickets es una parte extremadamente importante en el proceso de desarrollo de CakePHP. Todos los tickets de CakePHP están alojados en GitHub[47].

Reportar errores

Los reportes de errores bien escritos son de mucha ayuda. Para ello hay una serie de pasos que ayudan a crear el mejor reporte de error posible:

- **Correcto**: Por favor, busca tickets[48] similares que ya existan y asegúrate de que nadie haya reportado ya tu problema o que no haya sido arreglado en el repositorio.
- **Correcto**: Por favor, incluye instrucciones detalladas de **cómo reproducir el error**. Esto podría estar escrito en el formato de caso de prueba o con un **snippet** de código que demuestre el problema. No tener una forma de reproducir el error significa menos probabilidades de poder arreglarlo.
- **Correcto**: Por favor, danos todos los detalles posibles de tu entorno: sistema operativo, versión de PHP, versión de CakePHP…

[47] https://github.com/cakephp/cakephp/issues
[48] https://github.com/cakephp/cakephp/search?q=it+is+broken&ref=cmdform&type=Issues

- **Incorrecto**: Por favor, no utilices el sistema de tickets para hacer preguntas de soporte. El canal #cakephp IRC en Freenode[49] tiene muchos desarrolladores disponibles para ayudar a responder tus preguntas. También échale un vistazo a Stack Overflow[50].

Reportar problemas de seguridad

Si has encontrado problemas de seguridad en CakePHP, por favor, utiliza el siguiente procedimiento en vez del sistema de reporte de errores. En vez de utilizar el *tracker* de errores, lista de correos o IRC, por favor, envía un email a **security [at] cakephp.org**. Los emails enviados a esta dirección van al equipo principal de CakePHP en una lista de correo privada.

Por cada reporte primero tratamos de confirmar la vulnerabilidad, una vez confirmada el equipo de CakePHP tomará las siguientes acciones:

- Dar a conocer al reportador que hemos recibido el problema y que estamos trabajando en una solución. Pediremos al reportador que mantenga en secreto el problema hasta que nosotros lo anunciemos.
- Preparar una solución/parche.
- Preparar un *post* describiendo la vulnerabilidad y las posibles consecuencias.
- Publicar nuevas versiones para todas las que estén afectadas.
- Mostrar de manera acentuada el problema en el anuncio de la publicación.

Código

Parches y *pull requests* son una manera genial de contribuir con código a CakePHP. Los *Pull requests* pueden ser creados en Github, preferiblemente a los archivos de parches en los comentarios de tickets.

Configuración inicial

Antes de trabajar en parches para CakePHP es una buena idea configurar tu entorno de trabajo.

Necesitarás los siguientes programas:

- Git
- PHP 7.4 o mayor
- PHPUnit 5.7.0 o mayor

Configura tu información de usuario con tu nombre/alias y correo electrónico de trabajo:

```
git config --global user.name 'Bob Barker'
git config --global user.email 'bob.barker@example.com'
```

Nota: Si eres nuevo en Git, te recomendamos encarecidamente que leas el maravilloso y gratuito libro ProGit[51]

Clona el código fuente de CakePHP desde GitHub:

[49] https://webchat.freenode.net
[50] https://stackoverflow.com/questions/tagged/cakephp
[51] https://git-scm.com/book/

- Si no tienes una cuenta de GitHub[52] créate una.
- Haz un *fork* del repositorio CakePHP[53] haciendo click en el botón **Fork**.

Después de haber hecho el fork, clónalo en tu equipo local:

```
git clone git@github.com:TUNOMBRE/cakephp.git
```

Añade el repositorio original de CakePHP como respositorio remoto, lo usarás más adelante para buscar cambios en el repositorio de CakePHP. Esto te mantendrá actualizado con CakePHP:

```
cd cakephp
git remote add upstream git://github.com/cakephp/cakephp.git
```

Ahora que tienes configurado CakePHP deberías poder definir un `$test` de *conexión de base de datos* y *ejecutar todos los tests*.

Trabajar en un parche

Cada vez que quieras trabajar en un bug, una funcionalidad o en una mejora, crea una rama específica.

Tu rama debería ser creada a partir de la versión que quieras arreglar/mejorar. Por ejemplo, si estás arreglando un error en la versión `3.x` deberías utilizar la rama `master` como rama origen. Si tu cambio es para un error de la serie 2.x deberías usar la rama `2.x`. Esto hará más adelante tus *merges* más sencillos al no permitirte Github editar la rama destino:

```
# arreglando un error en 3.x
git fetch upstream
git checkout -b ticket-1234 upstream/master

# arreglando un error en 2.x
git fetch upstream
git checkout -b ticket-1234 upstream/2.x
```

Truco: Usa un nombre descriptivo para tu rama, referenciar el ticket o nombre de la característica es una buena convención. P.ej. ticket-1234, nueva-funcionalidad

---

Lo anterior creará una rama local basada en la rama *upstream* 2.x (CakePHP)

Trabaja en tu correción y haz tantos *commits* como necesites, pero ten siempre en mente lo siguiente:

- Sigue las *Estándares de codificación*.
- Añade un caso de prueba para mostrar el error arreglado o que la nueva funcionalidad funciona.
- Mantén lógicos tus commits y escribe comentarios de *commit* bien claros y concisos.

[52] https://github.com
[53] https://github.com/cakephp/cakephp

Enviar un *Pull Request*

Una vez estén hechos tus cambios y estés preparado para hacer el *merge* con CakePHP tendrás que actualizar tu rama:

```
# Hacer rebase de la corrección en el top de master
git checkout master
git fetch upstream
git merge upstream/master
git checkout <nombre_rama>
git rebase master
```

Esto buscará y hará *merge* de cualquier cambio que haya sucedido en CakePHP desde que empezaste. Entonces ejecutará *rebase* o replicará tus cambios en el *top* del actual código.

Puede que encuentres algún conflicto durante el *rebase*. Si este finaliza precipitadamente puedes ver qué archivos son conflictivos/*un-merged* con `git status`. Resuelve cada conflicto y continúa con el *rebase*:

```
git add <nombre_archivo> # haz esto con cada archivo conflictivo.
git rebase --continue
```

Comprueba que todas tus pruebas continúan pasando. Entonces sube tu rama a tu *fork*:

```
git push origin <nombre-rama>
```

Si has vuelto a hacer *rebase* después de hacer el *push* de tu rama necesitarás forzar el *push*:

```
git push --force origin <nombre-rama>
```

Una vez tu rama esté en GitHub puedes enviar un *pull request* en GitHub.

Seleccionar donde harán el *merge* tus cambios

Cuando hagas *pull requests* deberás asegurarte de seleccionar la rama correcta como base ya que no podrás editarla una vez creada.

- Si tus cambios son un *bugfix* (corrección de error) y no introduce ninguna funcionalidad nueva entonces selecciona **master** como destino del merge.
- Si tu cambio es una *new feature* (nueva funcionalidad) o un añadido al framework entonces deberías seleccionar la rama con el número de la siguiente versión. Por ejemplo si la versión estable actualmente es la `3.2.10`, la rama que estará aceptando nuevas funcionalidades será la `3.next`.
- Si tu cambio cesa una funcionalidad existente o de la *API* entonces tendrás que escojer la versión mayor siguiente. Por ejemplo, si la actual versión estable es la `3.2.2` entonces la siguiente versión en la que se puede cesar es la `4.x` por lo que deberás seleccionar esa rama.

Nota: Recuerda que todo código que contribuyas a CakePHP será licenciado bajo la Licencia MIT, y la Cake Software Foundation[54] será la propietaria de cualquier código contribuido. Los contribuidores deberán seguir las Guías de la comunidad CakePHP[55].

Todos los *merge* de corrección de errores que se hagan a una rama de mantenimiento se harán también periódicamente sobre futuros lanzamientos por el equipo central.

[54] https://cakefoundation.org/old
[55] https://cakephp.org/get-involved

Estándares de codificación

Los desarrolladores de CakePHP deberán utilizar la Guia de estilo de codificación PSR-12[56] además de las siguientes normas como estándares de codificación.

Es recomendable que otos *CakeIngredients* que se desarrollen sigan los mismos estándares.

Puedes utilizar el CakePHP Code Sniffer[57] para comprobar que tu código siga los estándares requeridos.

Añadir nuevas funcionalidades

Las nuevas funcionalidades no se deberán añadir sin sus propias pruebas, las cuales deberán ser superadas antes de hacer el *commit* en el repositorio.

Configuración del *IDE*

Asegúrate de que tu *IDE* haga *trim* por la derecha para que no haya espacios al final de las líneas.

La mayoría de los *IDEs* modernos soportan archivos `.editorconfig`. El esqueleto de aplicación de CakePHP viene con él por defecto y contiene las mejores prácticas de forma predeterminada.

Tabulación

Se utilizará cuatro espacios para la tabulación.

Por lo que debería everse asi:

```
// nivel base
    // nivel 1
        // nivel 2
    // nivel 1
// nivel base
```

O también:

```
$booleanVariable = true;
$stringVariable = 'moose';
if ($booleanVariable) {
    echo 'Boolean value is true';
    if ($stringVariable === 'moose') {
        echo 'We have encountered a moose';
    }
}
```

En los casos donde utilices llamadas de funciones que ocupen más de un línea usa las siguientes guías:

- El paréntesis de abertura de la llamada de la función deberá ser lo último que contenga la línea.
- Sólo se permite un argumento por línea.
- Los paréntesis de cierre deben estar solos y en una línea por separado.

Por ejemplo, en vez de utilizar el siguiente formato:

[56] https://www.php-fig.org/psr/psr-12/
[57] https://github.com/cakephp/cakephp-codesniffer

```
$matches = array_intersect_key($this->_listeners,
               array_flip(preg_grep($matchPattern,
                   array_keys($this->_listeners), 0)));
```

Utiliza éste en su lugar:

```
$matches = array_intersect_key(
    $this->_listeners,
    array_flip(
        preg_grep($matchPattern, array_keys($this->_listeners), 0)
    )
);
```

Tamaño de línea

Es recomendable mantener un tamaño de 100 caracteres por línea para una mejor lectura del código y tratar de no pasarse de los 120.

En resumen:

- 100 caracteres es el límite recomendado.
- 120 caracteres es el límite máximo.

Estructuras de control

Las estructuras de control son por ejemplo «`if`», «`for`», «`foreach`», «`while`», «`switch`» etc. A continuación un ejemplo con «`if`»:

```
if ((expr_1) || (expr_2)) {
    // accion_1;
} elseif (!(expr_3) && (expr_4)) {
    // accion_2;
} else {
    // accion_por_defecto;
}
```

- En las estructuras de control deberá haber un espacio antes del primer paréntesis y otro entre el último y la llave de apertura.
- Utiliza siempre las llaves en las estructuras de control incluso si no son necesarias. Aumentan la legibilidad del código y te proporcionan menos errores lógicos.
- Las llaves de apertura deberán estar en la misma línea que la estructura de control, las de cierre en líneas nuevas y el código dentro de las dos llaves en un nuevo nivel de tabulación.
- No deberán usarse las asignaciones *inline* en las estructras de control.

```
// Incorrecto: sin llaves y declaración mal posicionada
if (expr) statement;

// Incorrecto: sin llaves
if (expr)
    statement;
```

(continué en la próxima página)

(proviene de la página anterior)

```
// Correcto
if (expr) {
    statement;
}

// Incorrecto = asignación inline
if ($variable = Class::function()) {
    statement;
}

// Correcto
$variable = Class::function();
if ($variable) {
    statement;
}
```

Operador ternario

Los operadores ternarios están permitidos cuando toda su declaración cabe en una sola línea. Operadores más largos deberán ir dentro dentro de una declaración `if else`. Los operadores ternarios no deberían ir nunca anidados y opcionalmente pueden utilizarse paréntesis entorno a las condiciones para dar claridad:

```
// Correcto, sencillo y legible
$variable = isset($options['variable']) ? $options['variable'] : true;

// Incorrecto, operadores anidados
$variable = isset($options['variable']) ? isset($options['othervar']) ? true : false :␣
↪false;
```

Archivos de plantilla

En los archivos de plantilla (archivos .php) los desarrolladores deben utilizar estructuras de control `keyword` al ser más fáciles de leer en archivos complejos. Las estructuras de control pueden estar dentro de bloques de PHP o en etiquetas PHP separadas:

```
<?php
if ($esAdmin):
    echo '<p>Eres el usuario admin.</p>';
endif;
?>
<p>Lo siguiente es aceptado también:</p>
<?php if ($esAdmin): ?>
    <p>Eres el usuario admin.</p>
<?php endif; ?>
```

Comparación

Intenta ser siempre lo más estricto posible. Si una comparación no es estricta de forma deliberada, puede ser inteligente añadir un comentario para evitar confundirla con un error.

Para comprobar si una variables es `null` se recomienda utilizar comprobación estricta:

```
if ($value === null) {
    // ...
}
```

El valor contra el que se va a realizar la comparación deberá ir en el lado derecho de esta:

```
// no recomendado
if (null === $this->foo()) {
    // ...
}

// recomendado
if ($this->foo() === null) {
    // ...
}
```

Llamadas de funciones

Las llamadas a funciones deben realizarse sin espacios entre el nombre de la función y el parentesis de apertura y entre cada parámetro de la llamada deberá haber un espacio:

```
$var = foo($bar, $bar2, $bar3);
```

Como puedes ver arriba también deberá haber un espacio a ambos lados de los signos de igual.

Definición de métodos

Ejemplo de definición de un método:

```
public function someFunction($arg1, $arg2 = '')
{
    if (expr) {
        statement;
    }

    return $var;
}
```

Parámetros con un valor por defecto deberán ir al final de las definiciones. Trata que tus funciones devuelvan siempre un resultado, al menos `true` o `false`, para que se pueda determinar cuando la llamada a la función ha sido correcta:

```
public function connection($dns, $persistent = false)
{
    if (is_array($dns)) {
        $dnsInfo = $dns;
```

(continué en la próxima página)

(proviene de la página anterior)

```
    } else {
        $dnsInfo = BD::parseDNS($dns);
    }

    if (!($dnsInfo) || !($dnsInfo['phpType'])) {
        return $this->addError();
    }

    return true;
}
```

Como puedes ver hay un espacio a ambos lados del signo de igual.

Declaración de tipo

Los argumentos que esperan objetos, arrays o `callbacks` pueden ser tipificados. Solo tipificamos métodos públicos, aunque la tipificación no está libre de costes:

```
/**
 * Alguna descripción del método
 *
 * @param \Cake\ORM\Table $table La clase table a utilizar.
 * @param array $array Algún valor array.
 * @param callable $callback Algún callback.
 * @param bool $boolean Algún valor boolean.
 */
public function foo(Table $table, array $array, callable $callback, $boolean)
{
}
```

Aquí `$table` debe ser una instancia de `\Cake\ORM\Table`, `$array` debe ser un `array` y `$callback` debe ser de tipo `callable` (un `callback` válido).

Fíjate en que si quieres permitir que `$array` sea también una instancia de `\ArrayObject` no deberías tipificarlo ya que `array` acepta únicamente el tipo primitivo:

```
/**
 * Alguna descripción del método.
 *
 * @param array|\ArrayObject $array Algún valor array.
 */
public function foo($array)
{
}
```

Funciones anónimas (`Closures`)

Para definir funciones anónimas sigue la guía de estilo de código PSR-12[58] , donde se declaran con un espacio después de la palabra `function` y antes y después de la palabra `use`:

```
$closure = function ($arg1, $arg2) use ($var1, $var2) {
    // código
};
```

Encadenación de métodos

Las encadenaciones de métodos deberán distribuir estos en líneas separadas y tabulados con cuatro espacios:

```
$email->from('foo@example.com')
    ->to('bar@example.com')
    ->subject('A great message')
    ->send();
```

Comentar el código

Todos los comentarios deberán ir escritos en inglés y describir de un modo claro el bloque de código comentado.

Los comentarios pueden incluir las siguientes etiquetas de phpDocumentor[59]:

- @deprecated[60] Usando el formato `@version <vector> <description>`, donde `version` y `description` son obligatorios.
- @example[61]
- @ignore[62]
- @internal[63]
- @link[64]
- @see[65]
- @since[66]
- @version[67]

Las etiquetas PhpDoc son muy similares a las etiquetas JavaDoc en Java. Las etiquetas solo son procesadas si son el primer elemento en una línea DocBlock, por ejemplo:

```
/**
 * Ejemplo de etiqueta.
 *
```

(continué en la próxima página)

[58] https://www.php-fig.org/psr/psr-12/
[59] https://phpdoc.org
[60] https://docs.phpdoc.org/latest/guide/references/phpdoc/tags/deprecated.html
[61] https://docs.phpdoc.org/latest/guide/references/phpdoc/tags/example.html
[62] https://docs.phpdoc.org/latest/guide/references/phpdoc/tags/ignore.html
[63] https://docs.phpdoc.org/latest/guide/references/phpdoc/tags/internal.html
[64] https://docs.phpdoc.org/latest/guide/references/phpdoc/tags/link.html
[65] https://docs.phpdoc.org/latest/guide/references/phpdoc/tags/see.html
[66] https://docs.phpdoc.org/latest/guide/references/phpdoc/tags/since.html
[67] https://docs.phpdoc.org/latest/guide/references/phpdoc/tags/version.html

(proviene de la página anterior)

```
 * @author esta etiqueta es parseada, pero esta @version es ignorada
 * @version 1.0 esta etiqueta es parseada también
 */
```

```
/**
 * Ejemplo de etiquetas phpDoc inline.
 *
 * Esta función trabaja duramente con foo() para manejar el mundo.
 *
 * @return void
 */
function bar()
{
}

/**
 * Función foo.
 *
 * @return void
 */
function foo()
{
}
```

Los bloques de comentarios, con la excepción del primer bloque en un archivo, deberán ir siempre precedidos por un salto de línea.

Tipos de variables

Tipos de variables para utilizar en DocBlocks:

Tipo
: Descripción

mixed
: Una variable de tipo indefinido o múltiples tipos.

int
: Variable de tipo integer (números enteros).

float
: Tipo float (número de coma flotante).

bool
: Tipo booleano (true o false).

string
: Tipo string (cualquier valor entre « « o “ “).

null
: Tipo null. Normalmente usado conjuntamente con otro tipo.

array
: Tipo array.

object
: Tipo object. Debe usarse un nombre de clase específico si es posible.

resource
: Tipo resource (devuelto por ejemplo por mysql_connect()). Recuerda que cuando especificas el tipo como mixed deberás indicar si es desconocido o cuales son los tipos posibles.

callable
: Función Callable.

Puedes combinar tipos usando el caracter `|`:

```
int|bool
```

Para más de dos tipos normalmente lo mejor es utilizar `mixed`.

Cuando se devuelva el propio objeto, p.ej. para encadenar, deberás utilizar `$this` en su lugar:

```
/**
 * Función foo.
 *
 * @return $this
 */
public function foo()
{
    return $this;
}
```

Incluir archivos

`include`, `require`, `include_once` y `require_once` no tienen paréntesis:

```
// mal = paréntesis
require_once('ClassFileName.php');
require_once ($class);

// bien = sin paréntesis
require_once 'ClassFileName.php';
require_once $class;
```

Cuando se incluyan archivos con clases o librerías usa siempre y únicamente la función require_once[68].

Etiquetas PHP

Utiliza siempre las etiquetas <?php y ?> en lugar de <? y ?>.

La sintaxis abreviada de `echo` deberá usarse en los archivos de plantilla (**.php**) donde proceda.

[68] https://php.net/require_once

Sintaxis abreviada de echo

La sintaxis abreviada de `echo` (<?=) deberá usarse en los archivos de plantillas en lugar de `<?php echo`. Deberá ir seguido inmediatamente por un espacio, la variable o valor de la función a imprimir, un espacio y la etiqueta php de cierre:

```
// mal = con punto y coma y sin espacios
<td><?=$name;?></td>

// bien = con espacios y sin punto y coma
<td><?= $name ?></td>
```

A partir de la versión 5.4 de PHP la etiqueta (<?=) no es considerada un `short tag` y está siempre disponible sin importar la directiva `ini` de `short_open_tag`.

Convenciones de nomenclatura

Funciones

Escribe todas las funciones en `camelBack`:

```
function nombreFuncionLargo()
{
}
```

Clases

Los nombres de las clases deberán escribirse en `CamelCase`, por ejemplo:

```
class ClaseEjemplo
{
}
```

Variables

Los nombres de variables deberán ser todo lo descriptibles que puedan pero también lo más corto posible. Se escribirán en minúscula salvo que estén compuestos por múltiples palabras, en cuyo caso irán en `camelBack`. Los nombres de las variables que referencien objetos deberán ir asociados de algún modo a la clase de la cual es objeto. Ejemplo:

```
$usuario = 'John';
$usuarios = ['John', 'Hans', 'Arne'];

$dispatcher = new Dispatcher();
```

Visibilidad de miembros

Usa las palabras clave `public`, `protected` y `private` de PHP para métodos y variables.

Direcciones de ejemplos

Para los ejemplos de URL y correos electrónicos usa «example.com», «example.org» y «example.net», por ejemplo:

- Email: someone@example.com
- WWW: http://www.example.com
- FTP: ftp://ftp.example.com

El nombre de dominio «example.com» está reservado para ello (ver **RFC 2606**[69]) y está recomendado para usar en documentaciones o como ejemplos.

Archivos

Los nombres de archivos que no contengan clases deberán ir en minúsculas y con guiones bajos, por ejemplo:

```
nombre_de_archivo_largo.php
```

Hacer `casts`

Para hacer `casts` usamos:

Tipo
: Descripción

(bool)
: Cast a boolean.

(int)
: Cast a integer.

(float)
: Cast a float.

(string)
: Cast a string.

(array)
: Cast a array.

(object)
: Cast a object.

Por favor utiliza `(int)$var` en lugar de `intval($var)` y `(float)$var` en lugar de `floatval($var)` cuando aplique.

[69] https://datatracker.ietf.org/doc/html/rfc2606.html

Constantes

Los nombres de constantes deberán ir en mayúsculas:

```
define('CONSTANTE', 1);
```

Si el nombre de una constante se compone de varias palabras deberán ir separadas por guiones bajos, por ejemplo:

```
define('NOMBRE_DE_CONSTANTE_LARGO', 2);
```

Cuidado al usar empty()/isset()

Aunque `empty()` es una función sencilla de utilizar, puede enmascarar errores y causar efectos accidentales cuando se usa con `'0'` y `0`. Cuando las variables o propiedades están ya definidas el uso de `empty()` no es recomendable. Al trabajar con variables es mejor utilizar la conversión a tipo booleano en lugar de `empty()`:

```
function manipulate($var)
{
    // No recomendado, $var está definido en el ámbito
    if (empty($var)) {
        // ...
    }

    // Utiliza la conversión a booleano
    if (!$var) {
        // ...
    }
    if ($var) {
        // ...
    }
}
```

Cuando trates con propiedades definidas deberías favorecer las comprobaciones sobre `null` en lugar de `empty()/isset()`:

```
class Thing
{
    private $property; // Definido

    public function readProperty()
    {
        // No recomendado al estar definida la propiedad en la clase
        if (!isset($this->property)) {
            // ...
        }
        // Recomendado
        if ($this->property === null) {

        }
    }
}
```

Cuando se trabaja con arrays, es mejor hacer `merge` de valores por defecto en vez de hacer comprobaciones con `empty()`. Haciendo `merge` de valores por defecto puedes asegurarte de que las claves necesarias están definidas:

```php
function doWork(array $array)
{
    // Hacer merge de valor por defecto para eliminar la necesidad
    // de comprobaciones empty
    $array += [
        'key' => null,
    ];

    // No recomendado, la clave ya está seteada
    if (isset($array['key'])) {
        // ...
    }

    // Recomendado
    if ($array['key'] !== null) {
        // ...
    }
}
```

Guía de compatibilidad hacia atrás

Asegurar que puedas actualizar tus aplicaciones fácilmente es importante para nosotros. Por ello sólo rompemos la compatibilidad en las liberaciones de versiones `major`. Puedes familiarizarte con el versionado semántico[70], el cual utilizamos en todos los proyectos de CakePHP. Pero resumiendo, el versionado semántico significa que sólo las liberaciones de versiones `major` (tales como 2.0, 3.0, 4.0) pueden romper la compatibilidad hacia atrás. Las liberaciones `minor` (tales como 2.1, 3.1, 3.2) pueden introducir nuevas funcionalidades pero no pueden romper la compatibilidad. Los lanzamientos de correcciones de errores (tales como 3.0.1) no añaden nuevas funcionaliades, sólo correcciones de errores o mejoras de rendimiento.

Nota: CakePHP empezó a seguir el versionado semántico a partir de la 2.0.0. Estas reglas no se aplican en las versiones 1.x.

Para aclarar que cambios puedes esperar de cada nivel de lanzamiento tenemos más información detallada para desarrolladores que utilizan CakePHP y que trabajan en él que ayudan a aclarar que puede hacerse en liberaciones `minor`. Las liberaciones `major` pueden tener tantas rupturas de compatibilidad como sean necesarias.

Guías de migración

Para cada liberación `major` y `minor` el equipo de CakePHP facilitará una guía de migración. Estas guías explican las nuevas funcionaliades y cualquier ruptura de compatibilidad que haya en cada lanzamiento. Pueden encontrarse en la sección *Apéndices* del `cookbook`.

[70] https://semver.org/lang/es/

Usar CakePHP

Si estás desarrollando tu aplicación con CakePHP las siguientes pautas explican la estabilidad que puedes esperar.

Interfaces

Con excepción de las liberaciones `major`, las interfaces que provee CakePHP **no** tendrán ningún cambio en los métodos existentes. Podrán añadirse nuevos métodos pero no habrá cambios en los ya existentes.

Clases

Las clases que proporciona CakePHP pueden estar construidas y tener sus métodos y propiedades públicos usados por el código de la aplicación y, a excepción de las liberaciones `major`, la compatibilidad hacia atrás está garantizada.

Nota: Algunas clases en CakePHP están marcadas con la etiqueta API doc `@internal`. Estas clases **no** son estables y no garantizan la compatibilidad hacia atrás.

En liberaciones `minor` pueden añadirse nuevos métodos a las clases y a los ya existentes nuevos argumentos. Cualquier argumento nuevo tendrá un valor por defecto, pero si sobreescribes métodos con una firma diferente puedes encontrar `fatal errors`. Los métodos con nuevos argumentos estarán documentados en las guías de migración..

La siguiente tabla esboza varios casos de uso y que compatibilidad puedes esperar de CakePHP:

Si tu…	¿Compatible hacia atrás?
Tipificas contra la clase	Si
Creas una nueva instancia	Si
Extiendes la clase	Si
Accedes a una propiedad pública	Si
Llamas un método público	Si
Extiendes una clase y…	
Sobrescribes una propiedad pública	Si
Accedes a una propiedad protegida	No[1]
Sobreescribes una propiedad protegida	No[1]
Sobreescribes un método protegido	No[1]
Llamas a un método protegido	No[1]
Añades una propiedad pública	No
Añades un método público	No
Añades un argumento a un método sobreescrito	No[1]
Añades un valor por defecto a un argumento de método existente	Si

[1] Tu código *puede* romperse en lanzamientos `minor`. Comprueba la guía de migración para más detalles.

Trabajando en CakePHP

Si estás ayudando a que CakePHP sea aún mejor, por favor, ten en mente las siguientes pautas cuando añadas/cambies funcionalidades:

En una liberación `minor` puedes:

En una liberación `minor` puedes...	
Clases	
Eliminar una clase	No
Eliminar una interfaz	No
Eliminar un `trait`	No
Hacer final	No
Hacer abstract	No
Cambiar el nombre	Si[2]
Propiedades	
Añadir una propiedad pública	Si
Eliminar una propiedad pública	No
Añadir una propiedad protegida	Si
Eliminar una propiedad protegida	Si[3]
Métodos	
Añadir un método público	Si
Eliminar un método público	No
Añadir un método protegido	Si
Mover a la clase padre	Si
Eliminar un método protegido	Si[3]
Reducir visibilidad	No
Cambiar nombre del método	Si[2]
Añadir un argumento nuevo con valor por defecto	Si
Añadir un nuevo argumento obligatorio a un método existente	No
Eliminar un valor por defecto de un argumento existente	No

[2] Puedes cambiar el nombre de una clase/método siempre y cuando el antiguo nombre se mantenga disponible. Esto es evitado generalmente a menos que el cambio de nombre sea significativamente beneficioso.

[3] Evitarlo cuando sea posible. Cualquier borrado tendrá que ser documentado en la guía de migración.

CAPÍTULO 6

Instalación

CakePHP se instala rápida y fácilmente. Los requisitos mínimos son un servidor web y una copia de CakePHP, y ya! Aunque este manual se enfoca principalmente en configurar Apache (ya que es el más utilizado), puedes configurar CakePHP para que corra con una variedad de servidores web como nginx, LightHTHPD o Microsoft IIS.

Requisitos

- Servidor HTTP. Por ejemplo: Apache. mod_rewrite es recomendado, pero no requerido.
- PHP 7.4 o mayor.
- extensión mbstring.
- extensión intl.

Técnicamente una base de datos no es necesaria, pero imaginamos que la mayoría de aplicaciones utiliza alguna. CakePHP soporta una gran variedad de sistemas de bases de datos:

- MySQL (5.1.10 o mayor).
- PostgreSQL.
- Microsoft SQL Server (2008 o mayor).
- SQLite 3.

Nota: Todos los drivers nativos necesitan PDO. Debes asegurarte de tener las extensiones de PDO correctas.

Licencia

CakePHP está licenciado bajo la Licencia MIT[71]. Esto significa que eres libre para modificar, distribuir y republicar el código fuente con la condición de que las notas de copyright queden intactas. También eres libre para incorporar CakePHP en cualquier aplicación comercial o de código cerrado.

Instalando CakePHP

CakePHP utiliza Composer[72], una herramienta de manejo de dependencias para PHP 5.3+, como el método de instalación oficialmente soportado.

Primero, necesitas descargar e instalar Composer, si no lo has hecho ya. Si tienes instalado cURL, es tan fácil como correr esto en un terminal:

```
curl -s https://getcomposer.org/installer | php
```

O, puedes descargar `composer.phar` desde el sitio web de Composer[73].

Para sistemas Windows, puedes descargar el Instalador de Composer para Windows aquí[74]. Para más instrucciones acerca de esto, puedes leer el README del instalador de Windows aquí[75].

Ya que has descargado e instalado Composer puedes generar una aplicación CakePHP ejecutando:

```
php composer.phar create-project --prefer-dist cakephp/app:4.* [app_name]
```

O si tienes Composer definido globalmente:

```
composer create-project --prefer-dist cakephp/app:4.* [app_name]
```

Una vez que Composer termine de descargar el esqueleto y la librería core de CakePHP, deberías tener una aplicación funcional de CakePHP instalada vía Composer. Asegúrate de que los ficheros composer.json y composer.lock se mantengan junto con el resto de tu código fuente.

Ahora puedes visitar el destino donde instalaste la aplicación y ver los diferentes avisos tipo semáforo de los ajustes.

Mantente al día con los últimos cambios de CakePHP

Si quieres mantenerte al corriente de los últimos cambios en CakePHP puedes añadir las siguientes líneas al `composer.json` de tu aplicación:

```
"require": {
    "cakephp/cakephp": "dev-master"
}
```

Donde <branch> es el nombre del branch que quieres seguir. Cada vez que ejecutes `php composer.phar update` recibirás las últimas actualizaciones del branch seleccionado.

[71] https://www.opensource.org/licenses/mit-license.php
[72] https://getcomposer.org
[73] https://getcomposer.org/download/
[74] https://github.com/composer/windows-setup/releases/
[75] https://github.com/composer/windows-setup

Permisos

CakePHP utiliza el directorio **tmp** para varias operaciones. Descripciones de Modelos, el caché de las vistas y la información de la sesión son algunos ejemplos de lo anterior. El directorio **logs** es utilizado para para escribir ficheros de log por el motor de `FileLog` por defecto.

Asegúrate de que los directorios **logs**, **tmp** y todos sus subdirectorios tengan permisos de escritura por el usuario del Servidor Web. La instalación de CakePHP a través de Composer se encarga de este proceso haciendo que dichos directorios tengan los permisos abiertos globalmente con el fin de que puedas tener ajustado todo de manera más rápida. Obviamente es recomendable que revises, y modifiques si es necesario, los permisos tras la instalación vía Composer para mayor seguridad.

Un problema común es que **logs**, **tmp** y sus subdirectorios deben poder ser modificados tanto por el usuario del Servidor Web como por el usuario de la línea de comandos. En un sistema UNIX, si los usuarios mencionados difieren, puedes ejecutar los siguientes comandos desde el directorio de tu aplicación para asegurarte de que todo esté configurado correctamente:

```
HTTPDUSER=`ps aux | grep -E '[a]pache|[h]ttpd|[_]www|[w]ww-data|[n]ginx' | grep -v root␣
↪| head -1 | cut -d\  -f1`
setfacl -R -m u:${HTTPDUSER}:rwx tmp
setfacl -R -d -m u:${HTTPDUSER}:rwx tmp
setfacl -R -m u:${HTTPDUSER}:rwx logs
setfacl -R -d -m u:${HTTPDUSER}:rwx logs
```

Configuración

Configurar una aplicación de CakePHP puede ser tan simple como colocarla en el directorio raíz de tu Servidor Web, o tan complejo y flexible como lo desees. Esta sección cubrirá los dos tipos principales de instalación de CakePHP: Desarrollo y Producción.

- Desarrollo: fácil de arrancar, las URLs de la aplicación incluyen el nombre del directorio de la aplicación de CakePHP y es menos segura.
- Producción: Requiere tener la habilidad de configurar el directorio raíz del Servidor Web, cuenta con URLs limpias y es bastante segura.

Desarrollo

Este es el método más rápido para configurar CakePHP. En este ejemplo utilizaremos la consola de CakePHP para ejecutar el servidor web nativo de PHP para hacer que tu aplicación esté disponible en **http://host:port**. Para ello ejecuta desde el directorio de la aplicación:

```
bin/cake server
```

Por defecto, sin ningún argumento, esto colocará tu aplicación en **http://localhost:8765/**.

Si tienes algún conflicto con **localhost** o el puerto **8765**, puedes indicarle a la consola de CakePHP que corra el servidor de manera más específica utilizando los siguientes argumentos:

```
bin/cake server -H 192.168.13.37 -p 5673
```

Esto colocará tu aplicación en **http://192.168.13.37:5673/**.

¡Eso es todo! Tu aplicación de CakePHP está corriendo perfectamente sin tener que haber configurado el servidor web manualmente.

Nota: Prueba `bin/cake server -H 0.0.0.0` si el servidor no es accesible desde otra máquina.

Advertencia: El servidor de desarrollo *nunca* debe ser utilizado en un ambiente de producción. Se supone que esto es un servidor básico de desarrollo y nada más.

Si prefieres usar un servidor web «real», Debes poder mover todos tus archivos de la instalación de CakePHP (incluyendo los archivos ocultos) dentro la carpeta raíz de tu servidor web. Debes entonces ser capaz de apuntar tu navegador al directorio donde moviste los archivos y ver tu aplicación en acción.

Producción

Una instalación de producción es una manera más flexible de montar una aplicación de CakePHP. Utilizando este método, podrás tener un dominio entero actuando como una sola aplicación de CakePHP. Este ejemplo te ayudará a instalar CakePHP donde quieras en tu sistema de ficheros y tenerlo disponible en `http://www.example.com`. Toma en cuenta que esta instalación requiere que tengas los derechos de cambiar el directorio raíz (`DocumentRoot`) del servidor web Apache.

Después de instalar tu aplicación utilizando cualquiera de los métodos mencionados en el directorio elegido - asumiremos que has escogido /cake_install - tu estructura de ficheros debe ser la siguiente:

```
/cake_install/
    bin/
    config/
    logs/
    plugins/
    src/
    tests/
    tmp/
    vendor/
    webroot/ (este directorio es ajutado como el DocumentRoot)
    .gitignore
    .htaccess
    .travis.yml
    composer.json
    index.php
    phpunit.xml.dist
    README.md
```

Si utilizas Apache debes configurar la directiva `DocumentRoot` del dominio a:

```
DocumentRoot /cake_install/webroot
```

Si tu configuración del Servidor Web es correcta debes tener tu aplicación disponible ahora en http://www.example.com.

A rodar!

Muy bien, ahora veamos a CakePHP en acción. Dependiendo de los ajustes que hayas utilizado, deberías dirigirte en tu navegador a http://example.com/ o http://localhost:8765/. En este punto, encontrarás la página principal de CakePHP y un mensaje que te dice el estado actual de tu conexión a la base de datos.

¡Felicidades! Estás listo para *Crear tu primera aplicación de CakePHP*.

URL Rewriting

Apache

Mientras que CakePHP está diseñado para trabajar con mod_rewrite recién sacado del horno, usualmente hemos notado que algunos usuarios tienen dificultades para lograr que todo funcione bien en sus sistemas.

Aquí hay algunas cosas que puedes tratar de conseguir para que funcione correctamente. La primera mirada debe ir a httpd.conf. (Asegura de que estás editando el httpd.conf del sistema en lugar del httpd.conf de un usuario o sitio específico)

Hay archivos que pueden variar entre diferentes distribuciones y versiones de Apache. Debes también mirar en https://cwiki.apache.org/confluence/display/httpd/DistrosDefaultLayout para obtener información.

1. Asegura de que un archivo .htaccess de sobreescritura esté permitido y que *AllowOverride* esté ajustado en *All* para el correcto *DocumentRoot*. Debes ver algo similar a:

   ```
   # Cada directorio al que Apache puede acceder puede ser configurado
   # con sus respectivos permitidos/denegados servicios y características
   # en ese directorios (y subdirectorios).
   #
   # Primero, configuramos el por defecto para ser muy restrictivo con sus
   # ajustes de características.
   <Directory />
       Options FollowSymLinks
       AllowOverride All
   #    Order deny,allow
   #    Deny from all
   </Directory>
   ```

2. Asegura que tu estás cargando mod_rewrite correctamente. Debes ver algo similar a esto:

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

   En muchos sistemas esto estará comentado por defecto, así que solo debes remover el símbolo # al comienzo de la línea.

   Después de hacer los cambios, reinicia Apache para asegurarte que los ajustes estén activados.

   Verifica que tus archivos .htaccess está actualmente en directorio correcto. Algunos sistemas operativo tratan los archivos que empiezan con "." como oculto y por lo tanto no podrás copiarlos.

3. Asegúrate que tu copia de CakePHP provenga desde la sección descargas del sitio o de nuestro repositorio de Git, y han sido desempacados correctamente, revisando los archivos .htaccess.

   El directorio app de CakePHP (Será copiado en la raíz de tu aplicación por bake):

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

El directorio webroot de CakePHP (Será copiado a la raíz de tu aplicación web por bake):

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^ index.php [L]
</IfModule>
```

Si tu sitio aún tiene problemas con mod_rewrite, querrás probar modificar los ajustes para el Servidor Virtual. En Ubuntu, edita el archivo **/etc/apache2/sites-available/default** (la ubicación depende de la distribución). En este archivo, debe estar `AllowOverride None` cambiado a``AllowOverride All``, así tendrás:

```
<Directory />
    Options FollowSymLinks
    AllowOverride All
</Directory>
<Directory /var/www>
    Options Indexes FollowSymLinks MultiViews
    AllowOverride All
    Order Allow,Deny
    Allow from all
</Directory>
```

En macOS, otra solución es usar la herramienta virtualhostx[76] para crear servidores virtuales y apuntarlos a tu carpeta.

Para muchos servicios de alojamiento (GoDaddy, 1and1), tu servidor web estará actualmente sirviendo desde un directorio de usuario que actualmente usa mod_rewrite. Si tu estás instalando CakePHP en la carpeta de usuario (http://example.com/~username/cakephp/), o alguna otra estructura de URL que ya utilice mod_rewrite, necesitarás agregar una declaración a los archivos .htaccess que CakePHP usa (.htaccess, webroot/.htaccess).

Esto puede ser agregado a la misma sección con la directiva RewriteEngine, entonces por ejemplo, tu .htaccess en el webroot debería verse algo así:

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteBase /path/to/app
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^ index.php [L]
</IfModule>
```

Los detalles de estos cambios dependerán de tu configuración, y puede incluir algunas líneas adicionales que no están relacionadas con CakePHP. Por favor dirígete a la documentación en línea de Apache para más información.

4. (Opcional) Para mejorar la configuración de producción, debes prevenir archivos adicionales inválidos que sean tomados por CakePHP. Modificando tu .htaccess del webroot a algo cómo esto:

[76] https://clickontyler.com/virtualhostx/

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteBase /path/to/app/
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{REQUEST_URI} !^/(webroot/)?(img|css|js)/(.*)$
    RewriteRule ^ index.php [L]
</IfModule>
```

Lo anterior simplemente previene que archivos adicionales incorrectos sean enviados a index.php en su lugar muestre la página 404 de tu servidor web.

Adicionalmente puedes crear una página 404 que concuerde, o usar la página 404 incluida en CakePHP agregando una directiva `ErrorDocument`:

```
ErrorDocument 404 /404-not-found
```

nginx

nginx no hace uso de un archivo .htaccess como Apache, por esto es necesario crear la reescritura de URL en la configuraciones de *site-available*. Esto usualmente se encuentra en `/etc/nginx/sites-available/your_virtual_host_conf_file`. Dependiendo de la configuración, tu necesitarás modificar esto, pero por lo menos, necesitas PHP corriendo como una instancia FastCGI:

```
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/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 $uri/ /index.php?$args;
    }

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

En algunos servidores (Como Ubuntu 14.04) la configuración anterior no funcionará recién instalado, y de todas formas la documentación de nginx recomienda una forma diferente de abordar esto (https://nginx.org/en/docs/http/converting_rewrite_rules.html). Puedes intentar lo siguiente (Notarás que esto es un bloque de servidor {}, en vez de dos, pese a que si quieres que example.com resuelva a tu aplicación CakePHP en adición a www.example.com consulta el enlace de nginx anterior):

```
server {
    listen   80;
    server_name www.example.com;
    rewrite 301 http://www.example.com$request_uri permanent;

    # root directive should be global
    root   /var/www/example.com/public/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 $uri /index.php?$args;
    }

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

IIS7 (Windows)

IIS7 no soporta de manera nativa los archivos .htaccess. Mientras hayan *add-ons* que puedan agregar soporte a estos archivos, puedes también importar las reglas htaccess en IIS para usar las redirecciones nativas de CakePHP. Para hacer esto, sigue los siguientes pasos:

1. Usa el Intalador de plataforma Web de Microsoft[77] para instalar el Modulo de Redirreción 2.0[78] de URLs o descarga directamente (32-bit[79] / 64-bit[80]).
2. Crear un nuevo archivo llamado web.config en tu directorio de raíz de CakePHP.
3. Usando Notepad o cualquier editor de XML, copia el siguiente código en tu nuevo archivo web.config:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <system.webServer>
        <rewrite>
            <rules>
                <rule name="Exclude direct access to webroot/*"
```

(continué en la próxima página)

[77] https://www.microsoft.com/web/downloads/platform.aspx
[78] https://www.iis.net/downloads/microsoft/url-rewrite
[79] https://download.microsoft.com/download/D/8/1/D81E5DD6-1ABB-46B0-9B4B-21894E18B77F/rewrite_x86_en-US.msi
[80] https://download.microsoft.com/download/1/2/8/128E2E22-C1B9-44A4-BE2A-5859ED1D4592/rewrite_amd64_en-US.msi

(proviene de la página anterior)

```
                stopProcessing="true">
                  <match url="^webroot/(.*)$" ignoreCase="false" />
                  <action type="None" />
                </rule>
                <rule name="Rewrite routed access to assets(img, css, files, js, favicon)
↪"
                  stopProcessing="true">
                  <match url="^(img|css|files|js|favicon.ico)(.*)$" />
                  <action type="Rewrite" url="webroot/{R:1}{R:2}"
                    appendQueryString="false" />
                </rule>
                <rule name="Rewrite requested file/folder to index.php"
                  stopProcessing="true">
                  <match url="^(.*)$" ignoreCase="false" />
                  <action type="Rewrite" url="index.php"
                    appendQueryString="true" />
                </rule>
            </rules>
        </rewrite>
    </system.webServer>
</configuration>
```

Una vez el archivo web.config es creado con las reglas de redirección amigables de IIS, los enlaces, CSS, JavaScript y redirecciones de CakePHP deberían funcionar correctamente.

No puedo usar Redireccionamientos de URL

Si no quieres o no puedes obtener mod_rewirte (o algun otro modulo compatible) en el servidor a correr, necesitarás usar el decorador de URL incorporado en CakePHP. En **config/app.php**, descomentar la línea para que se vea así:

```
'App' => [
    // ...
    // 'baseUrl' => env('SCRIPT_NAME'),
]
```

También remover estos archivos .htaccess:

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

Esto hará tus URL verse así www.example.com/index.php/controllername/actionname/param antes que www.example.com/controllername/actionname/param.

CAPÍTULO 7

Configuration

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

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

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

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

CAPÍTULO 8

Routing

class Cake\Routing**Router**

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

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

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

Connecting Routes

Using Named Routes

Dispatcher Filters

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

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

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

[82] https://github.com/cakephp/docs
[83] https://github.com/cakephp/docs

CAPÍTULO 9

Request & Response Objects

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

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

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

Request

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

CAPÍTULO 10

Controladores

class Cake\Controller**Controller**

Los controladores son la "C" en MVC. Después de aplicar el enrutamiento y que el controlador ha sido encontrado, la acción de tu controlador es llamado. Tu controlador debe manejar la interpretación de los datos de la solicitud, asegurándose de que se llamen a los modelos correctos y se muestre la respuesta o vista correcta. Los controladores se pueden considerar como una capa intermedia entre el Modelo y la Vista. Quieres mantener tus controladores delgados, y tus modelos gruesos. Esto te ayudará a reutilizar tu código y lo hará mas fácil de probar.

Comúnmente, un controlador se usa para administrar la lógica en torno a un solo modelo. Por ejemplo, si estuvieras construyendo un sitio online para una panadería, podrías tener un RecipesController que gestiona tus recetas y un IngredientsController que gestiona tus ingredientes. Sin embargo, es posible hacer que los controladores trabajen con más de un modelo. En CakePHP, un controlador es nombrado a raíz del modelo que maneja.

Los controladores de tu aplicación extienden de la clase `AppController`, que a su vez extiende de la clase principal *`Controller`*. La clase `AppController` puede ser definida en **src/Controller/AppController.php** y debería contener los métodos que se comparten entre todos los controladores de tu aplicación.

Los controladores proveen una serie de métodos que manejan las peticiones. Estos son llamadas *acciones*. Por defecto, cada método público en un controlador es una acción, y es accesible mediante una URL. Una acción es responsable de interpretar la petición y crear la respuesta. Por lo general, las respuestas son de la forma de una vista renderizada, pero también, hay otras maneras de crear respuestas.

El App Controller

Como se indicó en la introducción, la clase `AppController` es clase padre de todos los controladores de tu aplicación. `AppController` extiende de la clase *`Cake\Controller\Controller`* que está incluida en CakePHP. `AppController` se define en **src/Controller/AppController.php** como se muestra a continuación:

```
namespace App\Controller;

use Cake\Controller\Controller;

class AppController extends Controller
{
}
```

Los atributos y métodos del controlador creados en tu `AppController` van a estar disponibles en todos los controladores que extiendan de este. Los componentes (que aprenderás más adelante) son mejor usados para código que se encuentra en muchos (pero no necesariamente en todos) los componentes.

Puedes usar tu `AppController` para cargar componentes que van a ser utilizados en cada controlador de tu aplicación. CakePHP proporciona un método `initialize()` que es llamado al final del constructor de un controlador para este tipo de uso:

```
namespace App\Controller;

use Cake\Controller\Controller;

class AppController extends Controller
{
    public function initialize(): void
    {
        // Siempre habilita el componente CSRF.
        $this->loadComponent('Csrf');
    }
}
```

Flujo de solicitud

Cuando se realiza una solicitud a una aplicación CakePHP, las clases CakePHP *`Cake\Routing\Router`* y `Cake\Routing\Dispatcher` usan *Connecting Routes* para encontrar y crear la instancia correcta del controlador. Los datos de la solicitud son encapsulados en un objeto de solicitud. CakePHP pone toda la información importante de la solicitud en la propiedad `$this->request`. Consulta la sección sobre *Request* para obtener más información sobre el objeto de solicitud de CakePHP.

Acciones del controlador

Las acciones del controlador son las responsables de convertir los parámetros de la solicitud en una respuesta para el navegador/usuario que realiza la petición. CakePHP usa convenciones para automatizar este proceso y eliminar algunos códigos repetitivos que de otro modo se necesitaría escribir.

Por convención, CakePHP renderiza una vista con una versión en infinitivo del nombre de la acción. Volviendo a nuestro ejemplo de la panadería online, nuestro RecipesController podría contener las acciones `view()`, `share()`, y `search()`. El controlador sería encontrado en **src/Controller/RecipesController.php** y contiene:

```
// src/Controller/RecipesController.php

class RecipesController extends AppController
{
    public function view($id)
    {
        // La lógica de la acción va aquí.
    }

    public function share($customerId, $recipeId)
    {
        // La lógica de la acción va aquí.
    }

    public function search($query)
    {
        // La lógica de la acción va aquí.
    }
}
```

Las plantillas para estas acciones serían **templates/Recipes/view.php**, **templates/Recipes/share.php**, y **templates/Recipes/search.php**. El nombre convencional para un archivo de vista es con minúsculas y con el nombre de la acción entre guiones bajos.

Las acciones de los controladores por lo general usan `Controller::set()` para crear un contexto que `View` usa para renderizar la capa de vista. Debido a las convenciones que CakePHP usa, no necesitas crear y renderizar la vista manualmente. En su lugar, una vez que se ha completado la acción del controlador, CakePHP se encargará de renderizar y entregar la vista.

Si por algún motivo deseas omitir el comportamiento predeterminado, puedes retornar un objeto `Cake\Http\Response` de la acción con la respuesta creada.

Para que puedas usar un controlador de manera efectiva en tu aplicación, cubriremos algunos de los atributos y métodos principales proporcionados por los controladores de CakePHP.

Interactuando con vistas

Los controladores interactúan con las vistas de muchas maneras. Primero, los controladores son capaces de pasar información a las vistas, usando `Controller::set()`. También puedes decidir qué clase de vista usar, y qué archivo de vista debería ser renderizado desde el controlador.

Configuración de variables de vista

`Cake\Controller\Controller::set`(*string $var*, *mixed $value*)

El método `Controller::set()` es la manera principal de mandar información desde el controlador a la vista. Una vez que hayas utilizado `Controller::set()`, la variable puede ser accedida en tu vista:

```
// Primero pasas las información desde el controlador:

$this->set('color', 'rosa');

// Después, en la vista, puede utilizar la información:
?>

Has seleccionado cubierta <?= h($color) ?> para la tarta.
```

El método `Controller::set()` también toma un array asociativo como su primer parámetro. A menudo, esto puede ser una forma rápida de asignar un conjunto de información a la vista:

```
$data = [
    'color' => 'pink',
    'type' => 'sugar',
    'base_price' => 23.95
];

// Hace $color, $type, y $base_price
// disponible para la vista:

$this->set($data);
```

Ten en cuenta que las variables de la vista se comparten entre todas las partes renderizadas por tu vista. Estarán disponibles en todas las partes de la vista: la plantilla y todos los elementos dentro de estas dos.

Configuración de las opciones de la vista

Si deseas personalizar la clase vista, las rutas de diseño/plantillas, ayudantes o el tema que se usarán para renderizar la vista, puede usar el método `viewBuilder()` para obtener un constructor. Este constructor se puede utilizar para definir propiedades de la vista antes de crearlas:

```
$this->viewBuilder()
    ->addHelper('MyCustom')
    ->setTheme('Modern')
    ->setClassName('Modern.Admin');
```

Lo anterior muestra cómo puedes cargar ayudantes personalizados, configurar el tema y usar una clase vista personalizada.

Renderizando una vista

Cake\Controller\Controller::**render**(*string $view*, *string $layout*)

El método `Controller::render()` es llamado automáticamente al final de cada solicitud de la acción del controlador. Este método realiza toda la lógica de la vista (usando la información que has enviado usando el método `Controller::set()`), coloca la vista dentro de su `View::$layout`, y lo devuelve al usuario final.

El archivo de vista por defecto utilizado para el renderizado es definido por convención. Si la acción `search()` de RecipesController es solicitada, el archivo vista en **templates/Recipes/search.php** será renderizado:

```
namespace App\Controller;

class RecipesController extends AppController
{
// ...
    public function search()
    {
        // Renderiza la vista en templates/Recipes/search.php
        return $this->render();
    }
// ...
}
```

Aunque CakePHP va a llamarlo automáticamente después de cada acción de lógica (a menos que llames a `$this->disableAutoRender()`), puedes usarlo para especificar un archivo de vista alternativo especificando el nombre de este como primer argumento del método `Controller::render()`.

Si `$view` empieza con "/", se asume que es una vista o un archivo relacionado con la carpeta **templates**. Esto permite el renderizado directo de elementos, muy útil en llamadas AJAX:

```
// Renderiza el elemento en templates/element/ajaxreturn.php
$this->render('/element/ajaxreturn');
```

El segundo parámetro `$layout` de `Controller::render()` te permita especificar la estructura con la que la vista es renderizada.

Renderizando una plantilla específica

En tu controlador, puede que quieras renderizar una vista diferente a la que es convencional. Puedes hacer esto llamando a `Controller::render()` directamente. Una vez que hayas llamado a `Controller::render()`, CakePHP no tratará de re-renderizar la vista:

```
namespace App\Controller;

class PostsController extends AppController
{
    public function my_action()
    {
        $this->render('custom_file');
    }
}
```

Esto renderizará **templates/Posts/custom_file.php** en vez de **templates/Posts/my_action.php**.

También puedes renderizar vistas dentro de plugins usando la siguiente sintaxis: `$this->render('PluginName.PluginController/custom_file')`. Por ejemplo:

```
namespace App\Controller;

class PostsController extends AppController
{
    public function myAction()
    {
        $this->render('Users.UserDetails/custom_file');
    }
}
```

Esto renderizará **plugins/Users/templates/UserDetails/custom_file.php**

Negociación del tipo de contenido

`Cake\Controller\Controller::viewClasses()`

Los controladores pueden definir una lista de clases de vistas que soportan. Después de que la acción del controlador este completa, CakePHP usará la lista de vista para realizar negociación del tipo de contenido. Esto permite a tu aplicación rehusar la misma acción del controlador para renderizar una vista HTML o renderizar una respuesta JSON o XML. Para definir la lista de clases de vista que soporta un controlador se utiliza el método `viewClasses()`:

```
namespace App\Controller;

use Cake\View\JsonView;
use Cake\View\XmlView;

class PostsController extends AppController
{
    public function viewClasses(): array
    {
        return [JsonView::class, XmlView::class];
    }
}
```

La clase `View` de la aplicación se usa automáticamente como respaldo cuando no se puede seleccionar otra vista en función del encabezado de la petición `Accept` o de la extensión del enrutamiento. Si tu aplicación necesita realizar una lógica diferente para diferente formatos de respuesta puedes usar `$this->request->is()` para construir la lógica condicional requerida.

Nota: Las clases de vista deben implementar el método estático `contentType()` para participar en las negociaciones del tipo de contenido.

Negociación de tipo de contenido alternativos

Si ninguna vista puede coincidir con las preferencias del tipo de contenido de la petición, CakePHP usará la clase base `View`. Si deseas solicitar una negociación del tipo de contenido, puedes usar `NegotiationRequiredView` que setea un código de estatus 406:

```
public function viewClasses(): array
{
    // Requiere aceptar la negociación del encabezado o devuelve una respuesta 406.
    return [JsonView::class, NegotiationRequiredView::class];
}
```

Puede usar el valor del tipo de contenido `TYPE_MATCH_ALL` para crear tu lógica de vista alternativa:

```
namespace App\View;

use Cake\View\View;

class CustomFallbackView extends View
{
    public static function contentType(): string
    {
        return static::TYPE_MATCH_ALL;
    }

}
```

Es importante recordar que las vistas coincidentes se aplican sólo *después* de intentar la negociación del tipo de contenido.

Nuevo en la versión 4.4.0: Anterior a 4.4 debes usar *Request Handling* en vez de `viewClasses()`.

Redirigiendo a otras páginas

`Cake\Controller\Controller::redirect`(*string|array $url*, *integer $status*)

El método `redirect()` agrega un encabezado `Location` y establece un código de estado de una respuesta y la devuelve. Deberías devolver la respuesta creada por `redirect()` para que CakePHP envíe la redirección en vez de completar la acción del controlador y renderizar la vista.

Puedes redigir usando los valores de un array ordenado:

```
return $this->redirect([
    'controller' => 'Orders',
    'action' => 'confirm',
    $order->id,
    '?' => [
        'product' => 'pizza',
        'quantity' => 5
    ],
    '#' => 'top'
]);
```

O usando una URL relativa o absoluta:

```
return $this->redirect('/orders/confirm');

return $this->redirect('http://www.example.com');
```

O la referencia de la página:

```
return $this->redirect($this->referer());
```

Usando el segundo parámetro puede definir un código de estatus para tu redirección:

```
// Haz un 301 (movido permanentemente)
return $this->redirect('/order/confirm', 301);

// Haz un 303 (Ver otro)
return $this->redirect('/order/confirm', 303);
```

Reenviando a un acción en el mismo controlador

Cake\Controller\Controller::**setAction**(*$action*, *$args...*)

Si necesitas reenviar la acción actual a una acción diferente en el *mismo* controlador, puedes usar `Controller::setAction()` para actualizar el objeto de la solicitud, modifica la plantilla de vista que será renderizada y reenvía la ejecución a la nombrada acción:

```
// Desde una acción de eliminación, puedes renderizar a lista de página
// actualizada.
$this->setAction('index');
```

Cargando modelos adicionales

Cake\Controller\Controller::**fetchTable**(*string $alias*, *array $config = []*)

La función `fetchTable()` es útil cuando se necesita usar una tabla que no es la predeterminada por el controlador:

```
// En un método del controlador.
$recentArticles = $this->fetchTable('Articles')->find('all', [
        'limit' => 5,
        'order' => 'Articles.created DESC'
    ])
    ->all();
```

Nuevo en la versión 4.3.0: `Controller::fetchTable()` fue añadido. Antes de 4.3 necesitas usar `Controller::loadModel()`.

Paginación de un modelo

Cake\Controller\Controller::**paginate**()

Este método se utiliza para paginar los resultados obtenidos por tus modelos. Puedes especificar tamaño de páginas, condiciones de búsqueda del modelo y más.

El atributo `$paginate` te da una manera de personalizar cómo `paginate()` se comporta:

```
class ArticlesController extends AppController
{
    public $paginate = [
        'Articles' => [
            'conditions' => ['published' => 1]
        ]
    ];
}
```

Configuración de componentes para cargar

Cake\Controller\Controller::**loadComponent**(*$name*, *$config = []*)

En el método `initialize()` de tu controlador, puedes definir cualquier componente que deseas cargar, y cualquier dato de configuración para ellos:

```
public function initialize(): void
{
    parent::initialize();
    $this->loadComponent('Csrf');
    $this->loadComponent('Comments', Configure::read('Comments'));
}
```

Callbacks del ciclo de vida de la petición

Los controladores de CakePHP activan varios eventos/callbacks que puedes usar para insertar lógica alrededor del ciclo de vida de la solicitud.

Lista de eventos

- `Controller.initialize`
- `Controller.startup`
- `Controller.beforeRedirect`
- `Controller.beforeRender`
- `Controller.shutdown`

Métodos de callback del controlador

Por defecto, los siguientes métodos de callback están conectados a eventos relacionados si los métodos son implementados por tus controladores.

Cake\Controller\Controller::**beforeFilter**(*EventInterface $event*)

Llamado durante el evento `Controller.initialize` que ocurre antes de cada acción en el controlador. Es un lugar útil para comprobar si hay una sesión activa o inspeccionar los permisos del usuario.

Nota: El método beforeFilter() será llamado por acciones faltantes.

Devolver una respuesta del método `beforeFilter` no evitará que otros oyentes del mismo evento sean llamados. Debes explícitamente parar el evento.

Cake\Controller\Controller::**beforeRender**(*EventInterface $event*)

Llamado durante el evento `Controller.beforeRender` que ocurre después de la lógica de acción del controlador, pero antes de que la vista sea renderizada. Este callback no se usa con frecuencia, pero puede ser necesaria si estas llamando *render()* de forma manual antes del final de una acción dada.

Cake\Controller\Controller::**afterFilter**(*EventInterface $event*)

Llamado durante el evento `Controller.shutdown` que se desencadena después de cada acción del controlador, y después de que se complete el renderizado. Este es el último método del controlador para ejecutar.

Además de las devoluciones de llamada del ciclo de vida del controlador, *Componentes* también proporciona un conjunto similar de devoluciones de llamada.

Recuerda llamar a los callbacks de `AppController` dentro de los callbacks del controlador hijo para mejores resultados:

```
//use Cake\Event\EventInterface;
public function beforeFilter(EventInterface $event)
{
    parent::beforeFilter($event);
}
```

Middleware del controlador

Cake\Controller\Controller::**middleware**(*$middleware*, *array $options = []*)

Middleware puede ser definido globalmente, en un ámbito de enrutamiento o dentro de un controlador. Para definir el middleware para un controlador en específico usa el método `middleware()` de tu método `initialize()` del controlador:

```
public function initialize(): void
{
    parent::initialize();

    $this->middleware(function ($request, $handler) {
        // Haz la lógica del middleware.

        // Verifica que devuelves una respuesta o llamas a handle()
        return $handler->handle($request);
```

(continué en la próxima página)

(proviene de la página anterior)

```
    });
}
```

El middleware definido por un controlador será llamado **antes** `beforeFilter()` y se llamarán a los métodos de acción.

Nuevo en la versión 4.3.0: `Controller::middleware()` fue agregado.

Más sobre controladores

El controlador de Páginas

El esqueleto oficial de CakePHP incluye un controlador por defecto **PagesController.php**. Este es un controlador simple y opcional que se usa para servir contenido estático. La página home que ves después de la instalación es generada usando este controlador y el archivo de vista **templates/Pages/home.php**. Si se crea el archivo de vista **templates/Pages/about_us.php** se podrá acceder a este usando la URL **http://example.com/pages/about_us**. Sientete libre de modificar el controlador para que cumpla con tus necesidades.

Cuando se cocina una app usando Composer el controlador es creado en la carpeta **src/Controller/**.

Componentes

Los componentes son paquetes de lógica que se comparten entre los controladores. CakePHP viene un con fantástico conjunto de componentes básicos que puedes usar para ayudar en varias tareas comunes. También puedes crear tus propios componentes. Si te encuentras queriendo copiar y pegar cosas entre componentes, deberías considerar crear tu propio componente que contenga la funcionalidad. Crear componentes mantiene el código del controlador limpio y te permite rehusar código entre los diferentes controladores.

Para más información sobre componentes incluidos en CakePHP, consulte el capítulo de cada componente:

Authentication

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

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

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

---

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

FlashComponent

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

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

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

Seguridad

`class SecurityComponent`(*ComponentCollection $collection*, *array $config = []*)

El componente de seguridad crea una forma de integrar seguridad de forma más estricta en tu aplicación. Proporciona métodos para diversas tareas como:

- Restringir qué métodos HTTP acepta tu aplicación.
- Protección contra manipulación de formularios.
- Requerir que se utilice SSL.
- Limitación de la comunicación entre controladores.

Como todos los componentes, se configura a través de varios parámetros configurables. Todas estas propiedades se pueden establecer directamente o a través de métodos setter del mismo nombre en `beforeFilter()` de tu controlador.

Al usar el componente de seguridad, obtienes automáticamente protección contra la manipulación de formularios. Los campos token ocultos se insertarán automáticamente en los formularios y serán verificados por el componente de seguridad.

Si estas utilizando las funciones de protección de formularios del componente de seguridad y otros componentes que procesan datos de formularios en tus devoluciones de llamada `startup()`, asegúrate de colocar el componente de seguridad antes de esos componentes en tu método `initialize()`.

Nota: Al usar el componente de seguridad, **debes** usar FormHelper para crear tus formularios. Además, **no** debes reescribir ninguno de los «nombres» de los campos. El componente de seguridad busca ciertos indicadores que son creados y manejados por FormHelper (especialmente esos creados en `create()` y `end()`). Es probable que la modificación dinámica de los campos que se envían en una solicitud POST, como deshabilitar, borrar o crear nuevos campos a través de JavaScript, haga que la solicitud se envíe a la devolución de llamada de blackhole.

Siempre debes verificar el método HTTP que se utiliza antes de ejecutarlo para evitar efectos secundarios. Debes verificar el método HTTP o usar `Cake\Http\ServerRequest::allowMethod()` para asegurarte de que se utiliza el método HTTP correcto.

---

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

Manejo de devoluciones de llamada blackhole

`SecurityComponent::blackHole`(*Controller $controller*, *string $error = ''*, *?SecurityException $exception = null*)

Si una acción está restringida por el componente de seguridad, es "black-holed" como una solicitud no válida que dará como resultado un error 400 por defecto. Puedes configurar este comportamiento seteando la opción de configuración `blackHoleCallback` para una función de devolución de llamada en el controlador.

Al configurar un método de devolución de llamada, puedes personalizar como el proceso blackhole funciona:

```
public function beforeFilter(EventInterface $event)
{
    parent::beforeFilter($event);

    $this->Security->setConfig('blackHoleCallback', 'blackhole');
}

public function blackhole($type, SecurityException $exception)
{
    if ($exception->getMessage() === 'Request is not SSL and the action is required to␣
↪be secure') {
        // Reformule el mensaje de excepción con un string traducible.
        $exception->setMessage(__('Please access the requested page through HTTPS'));
    }

    // Vuelve a lanzar la excepción reformulada condicionalmente.
    throw $exception;

    // Alternativamente, maneja el error. Por ejemplo, configura un mensaje flash &
    // redirige a la versión HTTPS de la página solicitada.
}
```

El parámetro `$type` puede tener los siguientes valores:

- "auth" Indica un error de validación de formulario o un error de discrepancia entre controlador y acción.
- "secure" Indica un error de restricción del método SSL.

Prevención de manipulación de formularios

Por defecto, `SecurityComponent` evita que los usuarios alteren los formularios de formas específicas. El `SecurityComponent` evitará las siguientes cosas:

- Los campos desconocidos no podrán ser agregados al formulario.
- Los campos no pueden ser eliminados del formulario.
- Los valores en las entradas ocultas no podrán ser modificadas.

La prevención de este tipo de manipulación se logra trabajando con `FormHelper` y rastreando qué campos hay en un formulario. También se realiza un seguimiento de los valores de los campos ocultos. Todos estos datos se combinan y se convierten en un hash. Cuando un formulario es enviado, `SecurityComponent` usará los datos POST para construir la misma estructura y comparar el hash.

Nota: SecurityComponent **no** evitará que se agreguen/cambien opciones seleccionadas. Tampoco impedirá que se agreguen/cambien opciones de radio.

unlockedFields
: Establecer en una lista de campos de formulario para excluir de la validación POST. Los campos se pueden desbloquear en el componente o con `FormHelper::unlockField()`. Los campos que han sido desbloqueados no están obligados a ser parte del POST y los campos desbloqueados ocultos no tienen su valores verificados.

validatePost
: Establece en `false` para omitir por completo la validación de las solicitudes POST, esencialmente desactivando las validaciones de los formularios.

Uso

La configuración del componente de seguridad generalmente se realiza en las devoluciones de llamada `initialize` o `beforeFilter()` del controlador:

```
namespace App\Controller;

use App\Controller\AppController;
use Cake\Event\EventInterface;

class WidgetsController extends AppController
{
    public function initialize(): void
    {
        parent::initialize();
        $this->loadComponent('Security');
    }

    public function beforeFilter(EventInterface $event)
    {
        parent::beforeFilter($event);

        if ($this->request->getParam('prefix') === 'Admin') {
            $this->Security->setConfig('validatePost', false);
        }
    }
}
```

El ejemplo anterior deshabilitaría la prevención de manipulación de formularios para rutas con prefijo de administrador.

Protección CSRF

CSRF o Cross Site Request Forgery es una vulnerabilidad común en las aplicaciones web. Permite a un atacante capturar y reproducir una solicitud anterior, y a veces, enviar solicitudes de datos utilizando etiquetas de imagen o recursos en otros dominios. Para habilitar las funciones de protección CSRF.

Deshabilitar la manipulación de formularios para acciones específicas

Hay muchos casos en los que querrías deshabilitar la prevención de manipulación de formularios para una acción (por ejemplo, solicitudes AJAX). Puedes «desbloquear» estas acciones enumerándolas en `$this->Security->unlockedActions` en tu `beforeFilter()`:

```
namespace App\Controller;

use App\Controller\AppController;
use Cake\Event\EventInterface;

class WidgetController extends AppController
{
    public function initialize(): void
    {
        parent::initialize();
        $this->loadComponent('Security');
    }

    public function beforeFilter(EventInterface $event)
    {
        parent::beforeFilter($event);

        $this->Security->setConfig('unlockedActions', ['edit']);
    }
}
```

Este ejemplo deshabilitaría todas las comprobaciones de seguridad para las acciones de edición.

Paginación

class Cake\Controller\Component**PaginatorComponent**

Uno de los mayores obstaculos para crear aplicaciones web flexibles y amigables para el usuario es diseñar una interfaz de usuario intuitiva. Muchas aplicaciones tienen a crecer en tamaño y complejidad rapidamente, y los diseñadores y programadores por igual no pueden hacer frente a la visualización de cientos o miles de registros. Refactorizar lleva tiempo, y el rendimiento y la satisfación del usuario pueden verse afectados.

Mostrar un número razonable de registros por página siempre ha sido una parte crítica para cada aplicación y suele causar muchos dolores de cabeza a los desarrolladores. CakePHP alivia la carga del desarrollador al proporcionar una manera rapida y fácil de paginar datos.

La paginación en CakePHP es ofrecida por un componente de un controlador. Puedes utilizar *PaginatorHelper* en la vista de tu plantilla para generar los controles de paginación.

Uso Básico

Para paginar una consulta primero debemos cargar el `PaginatorComponent`:

```
class ArticlesController extends AppController
{
    public function initialize(): void
    {
        parent::initialize();
        $this->loadComponent('Paginator');
    }
}
```

Una vez cargado podemos paginar una tabla de clase ORM o un objeto `Query`:

```
public function index()
{
    // Paginate the ORM table.
    $this->set('articles', $this->paginate($this->Articles));

    // Paginate a partially completed query
    $query = $this->Articles->find('published');
    $this->set('articles', $this->paginate($query));
}
```

Uso Avanzado

El componente `PaginatorComponent` admite casos de uso más complejos mediante la configuración de la propiedad del controlador `$paginate` o como el argumento `$settings` para `paginate()`. Estas condiciones sirven como base para tus consultas de paginación. Son aumentados por los parametros `sort`, `direction`, `limit`, y `page` pasados dentro de la URL:

```
class ArticlesController extends AppController
{
    public $paginate = [
        'limit' => 25,
        'order' => [
            'Articles.title' => 'asc'
        ]
    ];
}
```

Truco: Las opciones predeterminadas de `order` deben definirse como un array.

Si bien puedes incluir cualquiera de las opciones soportadas por `find()` como `fields` en tus ajustes de paginación. Es más limpio y sencillo agrupar tus opciones de paginación dentro de *Custom Finder Methods*. Puedes usar tu buscador en la paginación utilizando la opción `finder`

```
class ArticlesController extends AppController
{
    public $paginate = [
```

(continué en la próxima página)

(proviene de la página anterior)

```
        'finder' => 'published',
    ];
}
```

Si tu metodo de busqueda requiere opciones adicionales, puedes pasarlas como como valores para el buscador:

```
class ArticlesController extends AppController
{
    // find articles by tag
    public function tags()
    {
        $tags = $this->request->getParam('pass');

        $customFinderOptions = [
            'tags' => $tags
        ];
        // Estamos utilizando el argumento $settings para paginate() aqui.
        // Pero la misma estructura puede ser utilizada para $this->paginate
        //
        // Nuestro buscador personalizado se llama findTagged dentro ArticlesTable.php
        // por eso estamos usando `tagged` como clave.
        // Nuestro buscador deberia verse como:
        // public function findTagged(Query $query, array $options) {
        $settings = [
            'finder' => [
                'tagged' => $customFinderOptions
            ]
        ];
        $articles = $this->paginate($this->Articles, $settings);
        $this->set(compact('articles', 'tags'));
    }
}
```

Además de definir valores generales de paginación, puedes definir mas de un conjunto de valores predeterminados para la paginación en el controlador. El nombre de cada modelo puede ser usado como clave en la propiedad `$paginate`:

```
class ArticlesController extends AppController
{
    public $paginate = [
        'Articles' => [],
        'Authors' => [],
    ];
}
```

Los valores de las claves de `Articles` y `Authors` podrían contener todas las propiedades que tendría una matriz básica `$paginate`.

Una vez que hayas utilizado `paginate()` para crear resultados. La solicitud del controlador se actualizará con los parámetros de paginación. Puedes acceder a los metadatos de paginación en `$this->request->getParam('paging')`.

Paginación Simple

Por defecto, la paginación utiliza una consulta `count()` para calcular el tamaño del conjunto de resultados para que puedan ser renderizados los enlaces de número de página. En conjuntos de datos muy grandes, esta consulta de conteo puede ser muy costosa. En situaciones donde solo quieres mostrar los enlaces «Siguiente» y «Anterior» puedes utilizar el paginador "simple" que realiza una consulta de conteo:

```
public function initialize(): void
{
    parent::initialize();

    // Load the paginator component with the simple paginator strategy.
    $this->loadComponent('Paginator', [
        'paginator' => new \Cake\Datasource\SimplePaginator(),
    ]);
}
```

Cuando se utilice el `SimplePaginator` no se podra generar los números de pagina, datos de contador, enlaces a la ultima pagina, o controles de recuento total de registros.

Utilizando Directamente PaginatorComponent

Si necesitas paginar datos de otro componente, puedes utilizar el `PaginatorComponent` directamente. Cuenta con una API similar al método controlador:

```
$articles = $this->Paginator->paginate($articleTable->find(), $config);

// Or
$articles = $this->Paginator->paginate($articleTable, $config);
```

El primer parámetro debe ser el objeto de consulta a encontrar en la tabla de objetos de la que se desea paginar los resultados. Opcionalmente, puedes pasar el tabla de objetos y dejar la consulta se construirá para usted. El segundo parametro deberia ser el array de los ajustes para usar en la paginación. Este array deberia tener la misma estructura que la propiedad `$paginate` en el controlador. Al paginar un objeto `Query`, la opción `finder` sera ignorada. Se da por asumido que se esta pasando la consulta que desas que sea paginada.

Paginando Multiples Consultas

Puedes paginar multiples modelos en una sola accion del controlador, usando la opción `scope` tanto en la propiedad `$paginate` del controlador y en la llamada al metodo `paginate()`:

```
// Propiedad paginado
public $paginate = [
    'Articles' => ['scope' => 'article'],
    'Tags' => ['scope' => 'tag']
];

// En una acción del controlador
$articles = $this->paginate($this->Articles, ['scope' => 'article']);
$tags = $this->paginate($this->Tags, ['scope' => 'tag']);
$this->set(compact('articles', 'tags'));
```

La opción `scope` dará como resultado el aspecto de `PaginatorComponent` en parámetros de cadena de consulta con ámbito. Por ejemplo, el siguiente URL podría ser utilizado para paginar tags y articles al mismo tiempo:

```
/dashboard?article[page]=1&tag[page]=3
```

Consulte la sección *paginator-helper-multiple* para saber como generar elementos HTML con ambito y URLs para paginación.

Paginar el Mismo Modelo Varias Veces

Para paginar el mismo modelo multiples veces dentro de una sola acción del controlador necesitas definir un alias para el modelo. Consulte *table-registry-usage* para detalles adicionales sobre como utilizar la tabla de registros:

```
// En una acción del controlador
$this->paginate = [
    'ArticlesTable' => [
        'scope' => 'published_articles',
        'limit' => 10,
        'order' => [
            'id' => 'desc',
        ],
    ],
    'UnpublishedArticlesTable' => [
        'scope' => 'unpublished_articles',
        'limit' => 10,
        'order' => [
            'id' => 'desc',
        ],
    ],
];

// Registrar una tabla de objetos adicional para permitir la diferenciación en el␣
↪componente de paginación
TableRegistry::getTableLocator()->setConfig('UnpublishedArticles', [
    'className' => 'App\Model\Table\ArticlesTable',
    'table' => 'articles',
    'entityClass' => 'App\Model\Entity\Article',
]);

$publishedArticles = $this->paginate(
    $this->Articles->find('all', [
        'scope' => 'published_articles'
    ])->where(['published' => true])
);

$unpublishedArticles = $this->paginate(
    TableRegistry::getTableLocator()->get('UnpublishedArticles')->find('all', [
        'scope' => 'unpublished_articles'
    ])->where(['published' => false])
);
```

Controlar que Campos se utilizan para Ordenar

Por defecto, el ordenamiento se puede realizar en cualquier columna no virtual que la tabla tenga. Esto es, a veces no deseable ya que permite a los usuarios ordenar por columnas no indexadas que pueden provocar gran trabajo para ser ordenadas. Puedes establecer una lista blanca de campos que se pueden ordenar utilando la opción `sortWhitelist`. Esta opción es necesaria cuando quieres ordenar datos asociados o campos calculados que pueden formar parte de la consulta de paginación:

```
public $paginate = [
    'sortWhitelist' => [
        'id', 'title', 'Users.username', 'created'
    ]
];
```

Cualquier solicitud que intente ordenar campos que no se encuentren en el lista blanca será ignorada.

Limitar el Número Máximo de Filas por Página

El número de resultados que se obtienen por página se expone al usuario como el parametro `limit`. Generalmente no es deseable permitir que los usuarios obtengan todas las filas en un conjunto paginado. La opción `maxLimit` establece que nadie puede configurar este límite demasiado alto desde afuera. Por defecto, CakePHP limita el número maximo de filas que pueden ser obtenidas a 100. Si este limite por defecto no es apropiado para tu aplicación, puedes ajustarlo en las opciones de paginación, por ejemplo, reduciendolo a `10`:

```
public $paginate = [
    // Other keys here.
    'maxLimit' => 10
];
```

Si el parametro de la solictud es mayor a este valor, se reducirá al valor de `maxLimit`.

Uniendo Asociaciones Adicionales

Se pueden cargar asociaciones adicionales en la tabla paginada utilizando el parametro `contain`:

```
public function index()
{
    $this->paginate = [
        'contain' => ['Authors', 'Comments']
    ];

    $this->set('articles', $this->paginate($this->Articles));
}
```

Solicitudes de Página Fuera de Rango

El PaginatorComponent lanzará un `NotFoundException` cuando trate de acceder a una página no existente, es decir, cuando el número de página solicitado sea mayor al número de páginas.

Por lo tanto, puedes dejar que se muestre la página de error normal o utilizar un bloque try catch y tomar las medidas apropiadas cuando se detecta un `NotFoundException`:

```
use Cake\Http\Exception\NotFoundException;

public function index()
{
    try {
        $this->paginate();
    } catch (NotFoundException $e) {
        // Has algo aquí como redirigir a la primera página o a la ultima página.
        // $this->request->getAttribute('paging') te dara la información requerida.
    }
}
```

Paginación en la Vista

Consulte la documentación *PaginatorHelper* para saber como crear enlaces para la navegación de paginación.

Request Handling

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

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

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

FormProtection

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

Por favor, siéntase libre de enviarnos un pull request en Github[88] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

---

[87] https://github.com/cakephp/docs
[88] https://github.com/cakephp/docs

Checking HTTP Cache

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

Por favor, siéntase libre de enviarnos un pull request en Github[89] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

Configurando componentes

Muchos de los componentes principales requieren configuración. Algunos ejemplos de componentes que requieren configuración son *Seguridad* y *FormProtection*. La configuración para estos componentes, y para los componentes en general, es usualmente hecho a través `loadComponent()` en el método `initialize()` del controlador o a través del array `$components`:

```
class PostsController extends AppController
{
    public function initialize(): void
    {
        parent::initialize();
        $this->loadComponent('FormProtection', [
            'unlockedActions' => ['index'],
        ]);
        $this->loadComponent('Csrf');
    }

}
```

También puedes configurar los componentes en tiempo de ejecución usando el método `setConfig()`. A veces, esto es hecho en el método `beforeFilter()` del controlador. Lo anterior podría ser también expresado como:

```
public function beforeFilter(EventInterface $event)
{
    $this->FormProtection->setConfig('unlockedActions', ['index']);
}
```

Al igual que los helpers, componentes implementan los métodos `getConfig()` y `setConfig()` para leer y escribir los datos de configuración:

```
// Lee los datos de configuración.
$this->FormProtection->getConfig('unlockedActions');

// Escribe los datos de configuración
$this->Csrf->setConfig('cookieName', 'token');
```

Al igual que con los helpers, los componentes fusionarán automáticamente su propiedad `$_defaultConfig` con la configuración del controlador para crear la propiedad `$_config` que es accesible con `getConfig()` y `setConfig()`.

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

Componentes de alias

Una configuración común para usar es la opción `className`, que te permite utilizar componentes de alias. Esta característica es útil cuando quieres reemplazar `$this->Auth` u otra referencia común de componente con una implementación personalizada:

```
// src/Controller/PostsController.php
class PostsController extends AppController
{
    public function initialize(): void
    {
        $this->loadComponent('Auth', [
            'className' => 'MyAuth'
        ]);
    }
}

// src/Controller/Component/MyAuthComponent.php
use Cake\Controller\Component\AuthComponent;

class MyAuthComponent extends AuthComponent
{
    // Agrega tu código para sobreescribir el AuthComponent principal
}
```

Lo de arriba haría *alias* `MyAuthComponent` a `$this->Auth` en tus controladores.

Nota: El alias de un componente reemplaza esa instancia en cualquier lugar donde se use ese componente, incluso dentro de otros componentes.

Carga de componentes sobre la marcha

Es posible que no necesites todos tus componentes disponibles en cada acción del controlador. En situaciones como estas, puedes cargar un componente en tiempo de ejecución usando el método `loadComponent()` en tu controlador:

```
// En una acción del controlador
$this->loadComponent('OneTimer');
$time = $this->OneTimer->getTime();
```

Nota: Ten en cuenta que los componentes cargados sobre la marcha no perderán devoluciones de llamadas. Si te basas en que las devoluciones de llamada `beforeFilter` o `startup` serán llamadas, necesitarás llamarlas manualmente dependiendo de cuándo cargas tu componente.

Uso de componentes

Una vez que hayas incluido algunos componentes a tu controlador, usarlos es bastante simple. Cada componente que uses se exponen como una propiedad en tu controlador. Si cargaste el `Cake\Controller\Component\FlashComponent` en tu controlador, puedes acceder a él de esta forma:

```
class PostsController extends AppController
{
    public function initialize(): void
    {
        parent::initialize();
        $this->loadComponent('Flash');
    }

    public function delete()
    {
        if ($this->Post->delete($this->request->getData('Post.id')) {
            $this->Flash->success('Post deleted.');

            return $this->redirect(['action' => 'index']);
        }
    }
}
```

Nota: Dado que tanto los modelos como los componentes se agregan a los controladores como propiedades, comparten el mismo "espacio de nombres". Asegúrate de no dar a un componente y un modelo el mismo nombre.

Creando un componente

Supongamos que nuestra aplicación necesita realizar una operación matemática compleja en muchas partes diferentes de la aplicación. Podríamos crear un componente para albergar esta lógica compartida para su uso en muchos controladores diferentes.

El primer paso es crear un nuevo archivo de componente y clase. Crea el archivo en **src/Controller/Component/MathComponent.php**. La estructura básica para el componente debería verse algo como esto:

```
namespace App\Controller\Component;

use Cake\Controller\Component;

class MathComponent extends Component
{
    public function doComplexOperation($amount1, $amount2)
    {
        return $amount1 + $amount2;
    }
}
```

Nota: Todos los componentes deben extender de `Cake\Controller\Component`. De lo contrario, se disparará una

excepción.

Incluyendo tu componente en tus controladores

Una vez que nuestro componente está terminado, podemos usarlo en los controladores de la aplicación cargándolo durante el método `initialize()` del controlador. Una vez cargado, el controlador recibirá un nuevo atributo con el nombre del componente, a través del cual podemos acceder a una instancia del mismo:

```
// En un controlador
// Haz que el nuevo componente esté disponible en $this->Math,
// así como el estándar $this->Csrf
public function initialize(): void
{
    parent::initialize();
    $this->loadComponent('Math');
    $this->loadComponent('Csrf');
}
```

Al incluir componentes en un controlador, también puedes declarar un conjunto de parámetros que se pasarán al constructor del componente. Estos parámetros pueden ser manejados por el componente:

```
// En tu controlador.
public function initialize(): void
{
    parent::initialize();
    $this->loadComponent('Math', [
        'precision' => 2,
        'randomGenerator' => 'srand'
    ]);
    $this->loadComponent('Csrf');
}
```

Lo anterior pasaría el array que contiene precision y randomGenerator a `MathComponent::initialize()` en el parámetro `$config`.

Usando otros componentes en tu componente

A veces, uno de tus componentes necesita usar otro componente. Puedes cargar otros componentes agregándolos a la propiedad *$components*:

```
// src/Controller/Component/CustomComponent.php
namespace App\Controller\Component;

use Cake\Controller\Component;

class CustomComponent extends Component
{
    // El otro componente que tu componente usa
    protected $components = ['Existing'];

    // Ejecuta cualquier otra configuración adicional para tu componente.
```

(continué en la próxima página)

(proviene de la página anterior)

```
    public function initialize(array $config): void
    {
        $this->Existing->foo();
    }

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

// src/Controller/Component/ExistingComponent.php
namespace App\Controller\Component;

use Cake\Controller\Component;

class ExistingComponent extends Component
{
    public function foo()
    {
        // ...
    }
}
```

Nota: A diferencia de un componente incluido en un controlador, no se activarán devoluciones de llamada en el componente de un componente.

Accediendo al controlador de un componente

Desde dentro de un componente, puedes acceder al controlador actual a través del registro:

```
$controller = $this->getController();
```

Devoluciones de llamadas de componentes

Los componentes también ofrecen algunas devoluciones de llamadas de ciclo de vida de las solicitudes que les permiten aumentar el ciclo de solicitud.

beforeFilter(*EventInterface $event*)

Es llamado antes que el método beforeFilter del controlador, pero *después* del método initialize() del controlador.

startup(*EventInterface $event*)

Es llamado después del método beforeFilter del controlador, pero antes de que el controlador ejecute la acción actual del manejador.

beforeRender(*EventInterface $event*)

Es llamado después de que el controlador ejecute la lógica de la acción solicitada, pero antes de que el controlador renderize las vistas y el diseño.

shutdown(*EventInterface $event*)

Es llamado antes de enviar la salida al navegador.

beforeRedirect(*EventInterface $event*, *$url*, *Response $response*)

Es llamado cuando el método de redirección del controlador es llamado pero antes de cualquier otra acción. Si este método devuelve `false` el controlador no continuará en redirigir la petición. Los parámetros $url y $response permiten modificar e inspeccionar la ubicación o cualquier otro encabezado en la respuesta.

Usando redireccionamiento en eventos de componentes

Para redirigir desde dentro de un método de devolución de llamada de un componente, puedes usar lo siguiente:

```
public function beforeFilter(EventInterface $event)
{
    $event->stopPropagation();

    return $this->getController()->redirect('/');
}
```

Al detener el evento, le haces saber a CakePHP que no quieres ninguna otra devolución de llamada de componente para ejecutar, y que el controlador no debe manejar la acción más lejos. A partir de 4.1.0 puedes generar una `RedirectException` para señalar una redirección:

```
use Cake\Http\Exception\RedirectException;
use Cake\Routing\Router;

public function beforeFilter(EventInterface $event)
{
    throw new RedirectException(Router::url('/'))
}
```

Generar una excepción detendrá todos los demás detectores de eventos y creará una nueva respuesta que no conserva ni hereda ninguno de los encabezados de la respuesta actual. Al generar una `RedirectException` puedes incluir encabezados adicionales:

```
throw new RedirectException(Router::url('/'), 302, [
    'Header-Key' => 'value',
]);
```

Nuevo en la versión 4.1.0.

CAPÍTULO 11

Vistas

class Cake\View**View**

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

Por favor, siéntase libre de enviarnos un pull request en Github[90] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

Plantillas de vistas

Layouts

Elementos

Más acerca de Vistas

View Cells

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

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

Por favor, siéntase libre de enviarnos un pull request en Github[91] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

Themes

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

Por favor, siéntase libre de enviarnos un pull request en Github[92] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

Vistas JSON y XML

JsonView y XmlView le permiten crear respuestas JSON y XML, e integrarse con el `Cake\Controller\Component\RequestHandlerComponent`.

Al habilitar `RequestHandlerComponent` en su aplicación y habilitar la compatibilidad con las extensiones `json` y/o `xml`, puede aprovechar automáticamente las nuevas clases de vista. `JsonView` y `XmlView` se denominarán vistas de datos para el resto de esta página.

Hay dos formas de generar vistas de datos. La primera es mediante el uso de la opción `serialize` y la segunda es mediante la creación de archivos de plantilla normales.

Habilitación de vistas de datos en su aplicación

Antes de poder usar las clases de vista de datos, primero deberá cargar el `Cake\Controller\Component\RequestHandlerComponent` en su controlador:

```
public function initialize(): void
{
    ...
    $this->loadComponent('RequestHandler');
}
```

Esto se puede hacer en su `AppController` y habilitará el cambio automático de clase de vista en los tipos de contenido. También puede configurar el componente con la configuración `viewClassMap`, para asignar tipos a sus clases personalizadas y/o asignar otros tipos de datos.

Opcionalmente, puede habilitar las extensiones json y/o xml con *file-extensions*. Esto le permitirá acceder a `JSON`, `XML` o cualquier otra vista de formato especial utilizando una URL personalizada que termine con el nombre del tipo de respuesta como una extensión de archivo como `http://example.com/articles.json`.

De forma predeterminada, cuando no se habilitan las *file-extensions*, se utiliza la solicitud, seleccionando el encabezado `Accept`, seleccionando qué tipo de formato se debe presentar al usuario. Un ejemplo de formato `Accept` que se utiliza para representar respuestas `JSON` es `application/json`.

[91] https://github.com/cakephp/docs
[92] https://github.com/cakephp/docs

Uso de vistas de datos con la clave Serialize

La opción `serialize` indica qué variable(s) de vista se deben serializar cuando se utiliza una vista de datos. Esto le permite omitir la definición de archivos de plantilla para las acciones del controlador si no necesita realizar ningún formateo personalizado antes de que los datos se conviertan en json/xml.

Si necesita realizar algún formateo o manipulación de las variables de vista antes de generar la respuesta, debe usar archivos de plantilla. El valor de `serialize` puede ser un string o un array de variables de vista para serializar:

```
namespace App\Controller;

class ArticlesController extends AppController
{
    public function initialize(): void
    {
        parent::initialize();
        $this->loadComponent('RequestHandler');
    }

    public function index()
    {
        // Set the view vars that have to be serialized.
        $this->set('articles', $this->paginate());
        // Specify which view vars JsonView should serialize.
        $this->viewBuilder()->setOption('serialize', 'articles');
    }
}
```

También puede definir `serialize` como un array de variables de vista para combinar:

```
namespace App\Controller;

class ArticlesController extends AppController
{
    public function initialize(): void
    {
        parent::initialize();
        $this->loadComponent('RequestHandler');
    }

    public function index()
    {
        // Some code that created $articles and $comments

        // Set the view vars that have to be serialized.
        $this->set(compact('articles', 'comments'));

        // Specify which view vars JsonView should serialize.
        $this->viewBuilder()->setOption('serialize', ['articles', 'comments']);
    }
}
```

La definición de `serialize` como un array ha añadido la ventaja de anexar automáticamente un elemento `<response>` de nivel superior cuando se utiliza *XmlView*. Si utiliza un valor de string para `serialize` y XmlView, asegúrese de que la variable de vista tiene un único elemento de nivel superior. Sin un solo elemento de nivel superior, el Xml no

podrá generarse.

Uso de una vista de datos con archivos de plantilla

Debe usar archivos de plantilla si necesita realizar alguna manipulación del contenido de la vista antes de crear el resultado final. Por ejemplo, si tuviéramos artículos que tuvieran un campo que contuviera HTML generado, probablemente querríamos omitirlo de una respuesta JSON. Esta es una situación en la que un archivo de vista sería útil:

```
// Controller code
class ArticlesController extends AppController
{
    public function index()
    {
        $articles = $this->paginate('Articles');
        $this->set(compact('articles'));
    }
}

// View code - templates/Articles/json/index.php
foreach ($articles as &$article) {
    unset($article->generated_html);
}
echo json_encode(compact('articles'));
```

Puede hacer manipulaciones más complejas o usar ayudantes para formatear también. Las clases de vista de datos no admiten diseños. Asumen que el archivo de vista generará el contenido serializado.

Creación de vistas XML

class XmlView

De forma predeterminada, cuando se utiliza `serialize`, XmlView ajustará las variables de vista serializadas con un nodo <response>. Puede establecer un nombre personalizado para este nodo mediante la opción `rootNode`.

La clase XmlView admite la opción `xmlOptions` que le permite personalizar las opciones utilizadas para generar XML, por ejemplo, `tags` frente `attributes`.

Un ejemplo de uso de `XmlView` sería generar un sitemap.xml[93]. Este tipo de documento requiere que cambie `rootNode` y establezca atributos. Los atributos se definen mediante el prefijo `@`:

```
public function sitemap()
{
    $pages = $this->Pages->find()->all();
    $urls = [];
    foreach ($pages as $page) {
        $urls[] = [
            'loc' => Router::url(['controller' => 'Pages', 'action' => 'view', $page->
↪slug, '_full' => true]),
            'lastmod' => $page->modified->format('Y-m-d'),
            'changefreq' => 'daily',
            'priority' => '0.5'
        ];
```

(continué en la próxima página)

[93] https://www.sitemaps.org/protocol.html

(proviene de la página anterior)

```
    }

    // Define a custom root node in the generated document.
    $this->viewBuilder()
        ->setOption('rootNode', 'urlset')
        ->setOption('serialize', ['@xmlns', 'url']);
    $this->set([
        // Define an attribute on the root node.
        '@xmlns' => 'http://www.sitemaps.org/schemas/sitemap/0.9',
        'url' => $urls
    ]);
}
```

Creación de vistas JSON

class JsonView

La clase JsonView admite la opción `jsonOptions` que permite personalizar la máscara de bits utilizada para generar JSON. Consulte la documentación de json_encode[94] para conocer los valores válidos de esta opción.

Por ejemplo, para serializar la salida de errores de validación de las entidades CakePHP en una forma coherente de JSON:

```
// In your controller's action when saving failed
$this->set('errors', $articles->errors());
$this->viewBuilder()
    ->setOption('serialize', ['errors'])
    ->setOption('jsonOptions', JSON_FORCE_OBJECT);
```

Respuestas JSONP

Al utilizar `JsonView`, puede utilizar la variable de vista especial `_jsonp` para habilitar la devolución de una respuesta JSONP. Si se establece en `true` la clase de vista comprueba si se establece el parámetro de string de consulta denominado «callback» y, de ser así, envuelve la respuesta json en el nombre de función proporcionado. Si desea utilizar un nombre de parámetro de string de consulta personalizado en lugar de «callback», establezca `_jsonp` al nombre requerido en lugar de `true`..

Ejemplo de uso

Si bien el *RequestHandlerComponent* puede establecer automáticamente la vista en función del tipo de contenido o la extensión de la solicitud, también puede controlar las asignaciones de vistas en el controlador:

```
// src/Controller/VideosController.php
namespace App\Controller;

use App\Controller\AppController;
use Cake\Http\Exception\NotFoundException;
```

(continué en la próxima página)

[94] https://php.net/json_encode

(proviene de la página anterior)

```
class VideosController extends AppController
{
    public function export($format = '')
    {
        $format = strtolower($format);

        // Format to view mapping
        $formats = [
          'xml' => 'Xml',
          'json' => 'Json',
        ];

        // Error on unknown type
        if (!isset($formats[$format])) {
            throw new NotFoundException(__('Unknown format.'));
        }

        // Set Out Format View
        $this->viewBuilder()->setClassName($formats[$format]);

        // Get data
        $videos = $this->Videos->find('latest')->all();

        // Set Data View
        $this->set(compact('videos'));
        $this->viewBuilder()->setOption('serialize', ['videos']);

        // Set Force Download
        return $this->response->withDownload('report-' . date('YmdHis') . '.' . $format);
    }
}
```

Helpers

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

Por favor, siéntase libre de enviarnos un pull request en Github[95] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

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

Breadcrumbs

`class` Cake\View\Helper**BreadcrumbsHelper**(*View $view*, *array $config = []*)

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

Por favor, siéntase libre de enviarnos un pull request en Github[96] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

FlashHelper

`class` Cake\View\Helper**FlashHelper**(*View $view*, *array $config = []*)

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

Por favor, siéntase libre de enviarnos un pull request en Github[97] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

FormHelper

`class` Cake\View\Helper**FormHelper**(*View $view*, *array $config = []*)

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

Por favor, siéntase libre de enviarnos un pull request en Github[98] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

HtmlHelper

`class` Cake\View\Helper**HtmlHelper**(*View $view*, *array $config = []*)

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

Por favor, siéntase libre de enviarnos un pull request en Github[99] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

[96] https://github.com/cakephp/docs
[97] https://github.com/cakephp/docs
[98] https://github.com/cakephp/docs
[99] https://github.com/cakephp/docs

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

NumberHelper

`class` Cake\View\Helper**NumberHelper**(*View $view*, *array $config = []*)

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

Por favor, siéntase libre de enviarnos un pull request en Github[100] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

PaginatorHelper

`class` Cake\View\Helper**PaginatorHelper**(*View $view*, *array $config = []*)

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

Por favor, siéntase libre de enviarnos un pull request en Github[101] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

RSS

`class` Cake\View\Helper**RssHelper**(*View $view*, *array $config = []*)

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

Por favor, siéntase libre de enviarnos un pull request en Github[102] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

[100] https://github.com/cakephp/docs
[101] https://github.com/cakephp/docs
[102] https://github.com/cakephp/docs

SessionHelper

class Cake\View\Helper**SessionHelper**(*View $view*, *array $config = []*)

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

Por favor, siéntase libre de enviarnos un pull request en Github[103] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

TextHelper

class Cake\View\Helper**TextHelper**(*View $view*, *array $config = []*)

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

Por favor, siéntase libre de enviarnos un pull request en Github[104] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

TimeHelper

class Cake\View\Helper**TimeHelper**(*View $view*, *array $config = []*)

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

Por favor, siéntase libre de enviarnos un pull request en Github[105] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

UrlHelper

class Cake\View\Helper**UrlHelper**(*View $view*, *array $config = []*)

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

Por favor, siéntase libre de enviarnos un pull request en Github[106] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

[103] https://github.com/cakephp/docs
[104] https://github.com/cakephp/docs
[105] https://github.com/cakephp/docs
[106] https://github.com/cakephp/docs

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

CAPÍTULO 12

Acceso a la base de datos & ORM

En CakePHP el acceso a la base de datos se hace por medio de dos objetos primarios. El primero son **repositories** -repositorios- o **table objects** -objetos de tabla-. Estos objetos proveen acceso a colecciones de datos. Nos permiten guardar nuevos registros, modificar y borrar existentes, definir relaciones y realizar operaciones en masa. El segundo tipo de objeto son **entities** -entidades-. Las Entidades representan registros individuales y permiten definir funcionalidad y comportamiento a nivel de registro/fila.

Estas dos clases son responsables de manejar todo lo que sucede con datos, validez, interacción y evolución en tu área de trabajo.

El ORM incluído en CakePHP se especializa en base de datos relacionales, pero puede ser extendido para soportar alternativas.

El ORM de CakePHP toma ideas y conceptos de los modelos ActiveRecord y Datamapper. Aspira a crear una implementación híbrida que combine aspectos de los dos modelos para crear un ORM rápido y fácil de usar.

Antes de comentar explorando el ORM, asegurate de configurar tu conexion *configure your database connections*.

Ejemplo rápido

Para comenzar no es necesario escribir código. Si has seguido las convenciones de nombres para las tablas puedes comenzar a utilizar el ORM. Por ejemplo si quisieramos leer datos de nuestra tabla `articles`:

```
use Cake\ORM\TableRegistry;

// Prior to 3.6 use TableRegistry::get('Articles')
$articles = TableRegistry::getTableLocator()->get('Articles');
$query = $articles->find();
foreach ($query as $row) {
    echo $row->title;
}
```

Como se ve, no es necesario agregar código extra ni ninguna otra configuración, gracias al uso de las convenciones de CakePHP. Si quisieramos modificar nuestra clase ArticlesTable para agregar asociaciones o definir métodos adicionales deberiamos agregar las siguientes lineas en **src/Model/Table/ArticlesTable.php**

```
namespace App\Model\Table;

use Cake\ORM\Table;

class ArticlesTable extends Table
{

}
```

Las clases Table usan una version en CamelCase del nombre de la tabla, con el sufijo `Table`. Una vez que tú clase fue creada, puedes obtener una referencia a esta usando `TableRegistry` como antes:

```
use Cake\ORM\TableRegistry;

// Now $articles is an instance of our ArticlesTable class.
// Prior to 3.6 use TableRegistry::get('Articles')
$articles = TableRegistry::getTableLocator()->get('Articles');
```

Ahora que tenemos una clase Table concreta, probablemente querramos usar una clase Entity concreta. Las clases Entity permiten definir métodos de acceso y mutación, lógica para registros individuales y mucho mas. Comenzaremos agregando las siguientes lineas en **src/Model/Entity/Article.php**:

```
namespace App\Model\Entity;

use Cake\ORM\Entity;

class Article extends Entity
{

}
```

Las Entity usan la version CamelCase en singular del nombre de la tabla como su nombre. Ahora que hemos creado una clase Entity, cuando carguemos entidades de nuestra base de datos, vamos a obtener instancias de nuestra clase Article:

```
use Cake\ORM\TableRegistry;

// Now an instance of ArticlesTable.
// Prior to 3.6 use TableRegistry::get('Articles')
$articles = TableRegistry::getTableLocator()->get('Articles');
$query = $articles->find();

foreach ($query as $row) {
    // Each row is now an instance of our Article class.
    echo $row->title;
}
```

CakePHP usa convenciones de nombres para asociar las clases Table y Entity. Si necesitas modificar qué entidad utilizada una tabla, puedes usar el método `entityClass()` para especificar el nombre de una clase.

Vea *Table Objects* y *Entities* para mas información sobre como utilizar objetos Table y Entity en su aplicación.

Más información

Database Basics

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

Por favor, siéntase libre de enviarnos un pull request en Github[107] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

Query Builder

`class Cake\ORM\Query`

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

Por favor, siéntase libre de enviarnos un pull request en Github[108] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

Table Objects

`class Cake\ORM\Table`

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

Por favor, siéntase libre de enviarnos un pull request en Github[109] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

[107] https://github.com/cakephp/docs
[108] https://github.com/cakephp/docs
[109] https://github.com/cakephp/docs

Entities

`class Cake\ORM\Entity`

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

Por favor, siéntase libre de enviarnos un pull request en Github[110] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

Retrieving Data & Results Sets

`class Cake\ORM\Table`

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

Por favor, siéntase libre de enviarnos un pull request en Github[111] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

Custom Finder Methods

Validating Data

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

Por favor, siéntase libre de enviarnos un pull request en Github[112] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

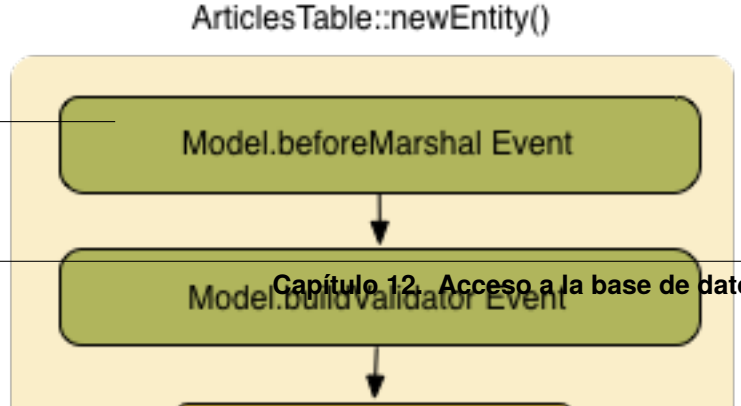
Usted puede hacer referencia a la versión en Inglés en el menú de selección superior para obtener información sobre el tema de esta página.

Saving Data

`class Cake\ORM\Table`

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

Por favor,



[110] https://github.com/cakephp/docs
[111] https://github.com/cakephp/docs
[112] https://github.com/cakephp/docs

siéntase libre de enviarnos un pull request en Github[113] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

Deleting Data

class Cake\ORM**Table**

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

Cake\ORM\Table::del

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

Por favor, siéntase libre de enviarnos un pull request en Github[114] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

Associations - Linking Tables Together

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

Por favor, siéntase libre de enviarnos un pull request en Github[115] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

Behaviors

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

Por favor, siéntase libre de enviarnos un pull request en Github[116] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

Core Behaviors

CounterCache Behavior

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

Por favor, siéntase libre de enviarnos un pull request en Github[117] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

[114] https://github.com/cakephp/docs
[115] https://github.com/cakephp/docs
[116] https://github.com/cakephp/docs
[117] https://github.com/cakephp/docs

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

Timestamp Behavior

class Cake\Model\Behavior**TimestampBehavior**

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

Por favor, siéntase libre de enviarnos un pull request en Github[118] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

Translate

class Cake\Model\Behavior**TranslateBehavior**

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

Por favor, siéntase libre de enviarnos un pull request en Github[119] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

Tree

class Cake\ORM\Behavior**TreeBehavior**

Muchas veces nos encontramos frente a la necesidad de tener que almacenar datos jerarquizados en una base de datos. Podría tomar la forma de categorías sin límite de subcategorías, datos relacionados con un sistema de menú multinivel o una representación literal de la jerarquía como un departamento en una empresa.

Las bases de datos relacionales no son verdaderamente apropiadas para almacenar y recobrar este tipo de datos, pero existen algunas técnicas para hacerlas eficientes y trabajar con una información multinivel.

[118] https://github.com/cakephp/docs
[119] https://github.com/cakephp/docs

El TreeBehavior le ayuda a mantener una estructura de datos jerárquica en la base de datos que puede ser solicitada fácilmente y ayuda a reconstruir los datos bajo una forma de árbol que permite encontrar y visualizar los procesos.

Prerrequisitos

Ese behavior requiere que las siguientes columnas estén presentes en la tabla:

- `parent_id` (nullable) La columna que contiene el ID del registro padre
- `lft` (integer, signed) Utilizado para mantener la estructura en forma de árbol
- `rght` (integer, signed) Utilizado para mantener la estructura en forma de árbol

Usted puede configurar el nombre de esos campos. Encontrará más información sobre la significación de los campos y sobre la manera de utilizarlos en este artículo que describe la MPTT logic[120]

Advertencia

Por el momento, TreeBehavior no soporta las llaves primarias composites.

[120] https://www.sitepoint.com/hierarchical-data-database-2/

Rápido vistazo

Active el Tree behavior agregándolo a la Tabla donde usted desea almacenar los datos jerarquizados en:

```
class␣
↪CategoriesTable␣
↪extends␣
↪Table
{
    public␣
↪function␣
↪initialize(array␣
↪$config)
    {

↪$this->
↪addBehavior(
↪'Tree');
    }
}
```

Tras agregarlas, puede dejar que CakePHP construya la estructura interna si la tabla ya contiene algunos registros:

```
// Prior to␣
↪3.6 use␣
↪TableRegistry::get(
↪'Categories
↪')
$categories␣
↪=␣
↪TableRegistry::getTableL
↪>get(
↪'Categories
↪');
$categories-
↪>
↪recover();
```

Usted puede comprobar que funciona recuperando cualquier registro de la tabla y preguntando cuantos descendientes posee:

```
$node =
↪$categories-
↪>get(1);
echo
↪$categories-
↪>
↪childCount(
↪$node);
```

Obtener una lista plana de los descendientes de un nodo es igual de fácil:

```

↪$descendants␣
```

(continué en la próxima página)

(proviene de la página anterior)

```
↪=
↪$categories-
↪>find(
↪'children
↪', ['for'␣
↪=> 1]);

foreach (
↪$descendants␣
↪as
↪$category)␣
↪{
    echo
↪$category-
↪>name . "\
↪n";
}
```

En cambio, si necesita una lista enlazada donde los hijos de cada nodo están anidados en una jerarquía, usted puede utilizar el finder 'threaded':

```
$children =
↪$categories
    ->find(
↪'children
↪', ['for'␣
↪=> 1])
    ->find(
↪'threaded
↪')
    ->
↪toArray();

foreach (
↪$children␣
↪as
↪$child) {
    echo "{
↪$child->
↪name} has
↪" . count(
↪$child->
↪children)␣
↪. "␣
↪direct␣
↪children";
}
```

Recorrer los resultados encadenados requiere generalmente funciónes recursivas, pero si usted necesita solamente un conjunto de resultados que contenga un campo único a partir de cada nivel para obtener una lista, en un <select> HTML por ejemplo, le será preferible recurrir al finder 'treeList':

```
$list =
↪$categories-
↪>find(
↪'treeList
↪');

// En un␣
↪fichero␣
↪plantilla␣
↪de Cake␣
↪PHP:
echo $this->
↪Form->
↪input(
↪'categories
↪', [
↪'options'␣
↪=>
↪$list]);

//  O puede␣
↪aficharlo␣
↪bajo␣
↪forma de␣
↪texto,␣
↪por␣
↪ejemplo␣
↪en un␣
↪script de␣
↪CLI
foreach (
↪$list as
↪$categoryName)␣
↪{
    echo
↪$categoryName␣
↪. "\n";
}
```

La salida se parecerá a esto:

```
My␣
↪Categories
_Fun
__Sport
___Surfing
___Skating
_Trips
__National
__
↪International
```

El finder `treeList` acepta una serie de opciones:

- `keyPath`: el camino separado por puntos para recuperar el campo que se utilizará en llave de array, o una clausura que devuelve la llave del registro suministrado.
- `valuePath`: el camino separado por puntos para recuperar el campo que se utilizará en llave de array, o una clausura que devuelve la llave del registro suministrado.
- `spacer`: una cadena de caracteres utilizada como prefijo para designar la profundidad del árbol para cada elemento.

Un ejemplo de uso de todas las opciones sería:

```
$query =
↪$categories-
↪>find(
↪'treeList
↪', [
    'keyPath
↪' => 'url
↪',

↪'valuePath
↪' => 'id',
    'spacer
↪' => ' '
]);
```

Une tarea común consiste en encontrar el camino en el árbol a partir de un nodo específico hacia la raíz. Es útil, por ejemplo, para añadir la lista de los hilos de Ariadna para una estructura de menú:

```
$nodeId = 5;
$crumbs =
↪$categories-
↪>find(
↪'path', [
↪'for' =>
↪$nodeId]);

foreach (
↪$crumbs␣
↪as
↪$crumb) {
    echo
↪$crumb->
↪name . ' >
↪ ';
}
```

Los árboles construidos con TreeBehavior no pueden ser clasificados con otras columnas que *lft* `, porque la representación interna del árbol depende de esa clasificación. Afortunadamente se pueden reestructurar los nodos dentro del mismo nivel sin tener que cambiar el elemento padre:

```
$node =
↪$categories-
↪>get(5);

// Desplaza␣
↪el nudo␣
↪para que␣
↪incremente␣
↪de una␣
↪posición␣
↪cuando␣
↪listamos␣
```

(continué en la próxima página)

(proviene de la página anterior)

```
↪los hijos
$categories-
↪>moveUp(
↪$node);

// ␣
↪Desplaza␣
↪el nudo␣
↪hacia lo␣
↪alto de␣
↪la lista␣
↪en el␣
↪mismo␣
↪nivel
$categories-
↪>moveUp(
↪$node,␣
↪true);

// ␣
↪Desplaza␣
↪el nudo␣
↪hacia␣
↪abajo.
$categories-
↪>moveDown(
↪$node,␣
↪true);
```

Configuración

Si los números de columna predeterminados empleados por ese behavior no corresponden a su esquema, usted puede ponerles alias:

```
public␣
↪function␣
↪initialize(array␣
↪$config)
{
    $this->
↪addBehavior(
↪'Tree', [

↪'parent'␣
↪=>
↪'ancestor_
↪id', // ␣
↪Utilice␣
↪esto␣
↪preferencialmente␣
↪en vez de␣
```

(continué en la próxima página)

(proviene de la página anterior)

```
    ↪parent_id

    ↪'left' =>
    ↪'tree_left
    ↪', // ␣
    ↪Utilice␣
    ↪esto en␣
    ↪vez de Ift

    ↪'right' =>
    ↪ 'tree_
    ↪right' //␣
    ↪ Utilice␣
    ↪esto en␣
    ↪vez de␣
    ↪rght
        ]);
}
```

Nivel de Nodos (profundidad)

Conocer la profundidad de una estructura en árbol puede ser útil cuando quiere recuperar los nodos solo hasta cierto nivel, por ejemplo para generar un menú. Puede utilizar la opción `level` para especificar los campos que guardarán el nivel de cada nodo:

```
$this->
↪addBehavior(
↪'Tree', [
    'level'␣
↪=> 'level
↪', //␣
↪null por␣
↪defecto,␣
↪i.e. no␣
↪guarda el␣
↪nivel
]);
```

Si usted no quiere copiar en caché el nivel utilizando un campo de la base de datos, puede utilizar el método `TreeBehavior::getLevel()` para conocer el nivel de un nodo.

Alcance y árboles múltiples

Si usted desea tener más de una estructura de árbol en la misma tabla, puede hacerlo utilizando la configuración 'scope' (alcance). Por ejemplo, si en una tabla locations desea crear un árbol por país:

```
class␣
↪LocationsTable␣
↪extends␣
↪Table
{
    public␣
↪function␣
↪initialize(array␣
↪$config)
    {

↪$this->
↪addBehavior(
↪'Tree', [

↪'scope' =>
↪ [
↪'country_
↪name' =>
↪'Brazil']
        ]);
    }
}
```

En el precedente ejemplo precedentela totalidad de las operaciones realizadas sobre el árbol solo se enfocarán en los registros que tienen la columna `country_name` que vale 'Brazil'. Usted puede cambiar el scope al vuelo utilizando la función 'config':

```
$this->
↪behaviors()-
↪>Tree->
↪config(
↪'scope', [
↪'country_
↪name' =>
↪'France
↪']);
```

Opcionalmente, puede ejercer un control más riguroso pasando una clausura como scope

```
$this->
↪behaviors()-
↪>Tree->
↪config(
↪'scope',␣
↪function (
↪$query) {

↪$country␣
```

(continué en la próxima página)

(proviene de la página anterior)

```
↪= $this->
↪getConfigureContry();
↪ // A␣
↪made-up␣
↪function

    return
↪$query->
↪where([
↪'country_
↪name' =>
↪$country]);
↪
});
```

Recobro con campo de clasificación personalizada

Por defecto, recover() clasifica los elementos por llave primaria. Eso funciona muy bien si se trata de una columna numérica (con incremento automático), pero puede ocasionar resultados raros si usted utiliza los UUIDs. Si necesita una clasificación personalizada para la recuperación de datos, puede agregar una cláusula de orden en la configuración:

```
$this->
↪addBehavior(
↪'Tree', [

↪'recoverOrder
↪' => [
↪'country_
↪name' =>
↪'DESC'],
]);
```

Guardar los datos jerarquizados

Generalmente cuando utiliza el Tree behavior, no tiene que preocuparse por la representación interna de la estructura jerarquizada. Las posiciones donde los nodos están colocados en el árbol se deducen de la columna 'parent_id' en cada una de sus entities:

```
$aCategory␣
↪=
↪$categoriesTable-
↪>get(10);
$aCategory->
↪parent_id␣
↪= 5;

↪$categoriesTable-
↪>save(
↪$aCategory);
↪
```

Proveer ids de padres inexistentes al grabar o intentar crear un bucle en el árbol (hacer un nodo hijo del mismo) provocará una excepción. Puede hacer un nodo a la raíz del árbol asignándolenull a la columna 'parent_id':

```
$aCategory␣
↪=
↪$categoriesTable-
↪>get(10);
$aCategory->
↪parent_id␣
↪= null;

↪$categoriesTable-
↪>save(
↪$aCategory);
↪
```

Los hijos para el nuevo nodo serán preservados.

Suprimir Nodos

Es fácil Suprimir un nodo, así como todo su sub-árbol (todos los hijos que puede tener a todo nivel del árbol):

```
$aCategory␣
↪=
↪$categoriesTable-
↪>get(10);

↪$categoriesTable-
↪>delete(
↪$aCategory);
↪
```

TreeBehavior se ocupará de todas las operaciones internas de supresión. También es posible suprimir solamente un nodo y reasignar todos los hijos al nodo padre inmediatamente superior en el árbol:

```
$aCategory␣
↪=
↪$categoriesTable-
↪>get(10);

↪$categoriesTable-
↪>
↪removeFromTree(
↪$aCategory);
↪

↪$categoriesTable-
↪>delete(
↪$aCategory);
↪
```

Todos los nodos hijos serán conservados y un nuevo padre les será asignado. La supresión de un nodo se basa sobre los valores lft y rght de la entity. Es importante observarlo cuando se ejecuta un bucle sobre los hijos de un nodo para supresiones condicionales:

```
↪$descendants␣
↪= $teams->
↪find(
↪'children
↪', ['for'␣
↪=> 1]);
foreach (
↪$descendants␣
↪as
↪$descendant)␣
↪{
    $team =
↪$teams->
↪get(
↪$descendant-
↪>id); //␣
↪busca el␣
↪objeto␣
↪entity al␣
↪día
    if (
↪$team->
↪expired) {

↪$teams->
↪delete(
↪$team); //
↪ la␣
↪supresión␣
↪reclasifica␣
↪las␣
↪entradas␣
↪lft y␣
↪rght de␣
↪la base␣
↪de datos
    }
}
```

TreeBehavior reclasifica los valores lft y rght de los registros de la tabla cuando se suprime un nodo. Tal como están, los valores lft y rght de las entities dentro de `$descendants` (guardadas antes de la operación de supresión) serán erróneas. Las entities tendrán que estar cargadas, y modificadas al vuelo para evitar incoherencias en la tabla.

Schema System

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

Por favor, siéntase libre de enviarnos un pull request en Github[121] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

ORM Cache Shell

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

Por favor, siéntase libre de enviarnos un pull request en Github[122] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

[121] https://github.com/cakephp/docs
[122] https://github.com/cakephp/docs

CAPÍTULO 13

Consola bake

Esta página se ha movido[123].

[123] https://book.cakephp.org/bake/1.x/es/

CAPÍTULO 14

Caching

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

Por favor, siéntase libre de enviarnos un pull request en Github[124] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

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

CAPÍTULO 15

Shells, Tasks & Console Tools

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

Por favor, siéntase libre de enviarnos un pull request en Github[125] 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.

More Topics

Shell Helpers

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

Por favor, siéntase libre de enviarnos un pull request en Github[126] 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.

[125] https://github.com/cakephp/docs
[126] https://github.com/cakephp/docs

Interactive Console (REPL)

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[127] 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.

Running Shells as Cron Jobs

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[128] 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.

I18N Shell

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

Por favor, siéntase libre de enviarnos un pull request en Github[129] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

Completion Shell

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

Por favor, siéntase libre de enviarnos un pull request en Github[130] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

[127] https://github.com/cakephp/docs
[128] https://github.com/cakephp/docs
[129] https://github.com/cakephp/docs
[130] https://github.com/cakephp/docs

Plugin Shell

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

Por favor, siéntase libre de enviarnos un pull request en Github[131] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

Routes Shell

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

Por favor, siéntase libre de enviarnos un pull request en Github[132] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

Upgrade Shell

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

Por favor, siéntase libre de enviarnos un pull request en Github[133] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

Server Shell

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

Por favor, siéntase libre de enviarnos un pull request en Github[134] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

[131] https://github.com/cakephp/docs
[132] https://github.com/cakephp/docs
[133] https://github.com/cakephp/docs
[134] https://github.com/cakephp/docs

Cache Shell

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

Por favor, siéntase libre de enviarnos un pull request en Github[135] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

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

CAPÍTULO 16

Depuración

La depuración es una parte inevitable y necesaria de cualquier ciclo de desarrollo. Aunque CakePHP no ofrece ninguna herramienta que se conecte directamente con algun IDE o editor, CakePHP proporciona varias herramientas para asistirte en la depuración y exponer lo que se está ejecutando bajo el capó de tu aplicación.

Depuración Básica

debug(*mixed $var*, *boolean $showHtml = null*, *$showFrom = true*)

La función `debug()` es una función que está disponible globalmente y funciona de manera similar a la función `print_r()` de PHP. La función `debug()` te permite mostrar el contenido de una variable de varias maneras. Primero, si deseas que los datos se muestren de una forma amigable con HTML, debes establecer el segundo parámetro en `true`. La función también imprime la línea y el archivo de origen por defecto.

El resultado de esta función solo se mostrará si la variable `$debug` en el archivo core es `true`.

Ver también `dd()`, `pr()` y `pj()`.

stackTrace()

La función `stackTrace()` está disponible globalmente, esta permite mostrar el seguimiento de pila donde sea que se llame.

breakpoint()

Si tienes *Psysh <https://psysh.org/>* _ instalado, puedes usar esta función en entornos CLI para abrir una consola interactiva con el ámbito local actual:

```
// Algún␣
↪código
eval(breakpoint());
↪
```

Abrirá una consola interactiva que puede ser usada para revisar variables locales y ejecutar otro código. Puedes salir del depurador interactivo y reanudar la ejecución original corriendo `quit` o `q` en la sesion interactiva.

Usando La Clase Debugger

class Cake\Error**Debugger**

Para usar el depurador, primero asegúrate de que `Configure::read('debug')` sea `true`.

Imprimiendo Valores

static Cake\Error\Debugger

Dump imprime el contenido de una variable. Imprimirá todas las propiedades y métodos (si existen) de la variable que se le pase:

```
$foo = [1,2,
↪3];

Debugger::dump(
↪$foo);

// Salida
array(
    1,
    2,
    3
)

// Objeto␣
↪simple
$car = new␣
↪Car();

Debugger::dump(
↪$car);
```

(continué en la próxima página)

(proviene de la página anterior)

```
// Salida
object(Car)
↪{
    color =>
↪ 'red'
    make =>
↪'Toyota'
    model =>
↪ 'Camry'
    mileage␣
↪=>␣
↪(int)15000
}
```

Enmascarando Datos

Al volcar datos con `Debugger` o mostrar páginas de error, es posible que desees ocultar claves sensibles como contraseñas o claves API. En tu `config/bootstrap.php` puedes enmascarar claves específicas:

```
Debugger::setOutputMask([

↪'password
↪' =>
↪'xxxxx',
    'awsKey
↪' =>
↪'yyyyy',
]);
```

Registros Con Trazas De Pila

```
static Cake\Error\Debugger
```

Crea un registro de seguimiento de pila detallado al momento de la invocación. El método `log()` imprime datos similar a como lo hace `Debugger::dump()`, pero al debug.log en vez de al buffer de salida. Ten en cuenta que tu directorio **tmp** (y su contenido) debe ser reescribible por el servidor web para que `log()` funcione correctamente.

Generando seguimientos de pila

static Cake\Error\Debugger

Devuelve el seguimiento de pila actual. Cada línea de la pila incluye cual método llama, incluyendo el archivo y la línea en la que se originó la llamada:

```
// En␣
↪PostsController::index()
pr(Debugger::trace());
↪

// Salida
PostsController::index()␣
↪- APP/
↪Controller/
↪DownloadsController.
↪php, line␣
↪48
Dispatcher::_
↪invoke() -
↪ CORE/src/
↪Routing/
↪Dispatcher.
↪php, line␣
↪265
Dispatcher::dispatch()␣
↪- CORE/
↪src/
↪Routing/
↪Dispatcher.
↪php, line␣
↪237
[main] -␣
↪APP/
↪webroot/
↪index.php,
↪ line 84
```

Arriba está el seguimiento de pila generado al llamar `Debugger::trace()` en una acción de un controlador. Leer el seguimiento de pila desde abajo hacia arriba muestra el órden de las funciones (cuadros de pila).

Obtener Un Extracto De Un Archivo

static Cake\Error\Debugger

Saca un extracto de un archivo en $path (el cual es una dirección absoluta), resalta el número de la línea $line con el número $context de líneas alrededor de este.

```
pr(Debugger::excerpt(ROOT␣
↪. DS .␣
↪LIBS .
↪'debugger.
↪php', 321,
↪ 2));

// Mostrará␣
↪lo␣
↪siguiente.
Array
(
    [0] =>
↪<code>
↪<span␣
↪style=
↪"color:
↪#000000">␣
↪* @access␣
↪public</
↪span></
↪code>
    [1] =>
↪<code>
↪<span␣
↪style=
↪"color:
↪#000000">␣
↪*/</span>
↪</code>
    [2] =>
↪<code>
↪<span␣
↪style=
↪"color:
↪#000000">␣
↪  ␣
↪function␣
↪excerpt(
↪$file,
↪$line,
↪$context␣
↪= 2) {</
↪span></
↪code>

    [3] =>
↪<span␣
↪class=
↪"code-
↪highlight
↪"><code>
↪<span␣
```

(continué en la próxima página)

(proviene de la página anterior)

```
↪style=
↪"color:
↪#000000">␣
↪
↪$data =
↪$lines =␣
↪array();</
↪span></
↪code></
↪span>
    [4] =>
↪<code>
↪<span␣
↪style=
↪"color:
↪#000000">␣
↪
↪$data =␣
↪@explode(
↪"\n",␣
↪file_get_
↪contents(
↪$file));</
↪span></
↪code>
)
```

Aunque este método es usado internamente, puede ser útil si estás creando tus propios mensajes de error o entradas de registros para situaciones customizadas.

static Cake\Error\Debugger

Consigue el tipo de una variable. Los objetos devolverán el nombre de su clase.

Usando El Registro Para Depurar

Registrar mensajes es otra buena manera de depurar aplicaciones, puedes usar `Cake\Log\Log` para hacer registros en tu aplicación. Todos los objetos que usen `LogTrait` tienen una instancia del método `log()` que puede ser usado para registrar mensajes:

```
$this->log(
↪'Llegó␣
↪aquí',
↪'debug');
```

Lo anterior escribiría `Llegó aquí` en el registro de depuración. Puedes usar entradas de registro para ayudar a los métodos de depuración que involucran redireccionamientos o búcles complejos. También puedes usar `Cake\Log\Log::write()` para escribir mensajes de registro. Este método puede ser llamado estáticamente en cualquier lugar de tu aplicación que un Log haya sido cargado:

```
// En el␣
↪tope del␣
```

(continué en la próxima página)

(proviene de la página anterior)

```
    ↪archivo␣
    ↪que␣
    ↪quieras␣
    ↪hacer␣
    ↪registros.
use Cake\
↪Log\Log;

// En␣
↪cualquier␣
↪parte que␣
↪Log haya␣
↪sido␣
↪importado.
Log::debug(
↪'Llegó␣
↪aquí');
```

Kit De Depuración

DebugKit es un complemento que proporciona una serie de buenas herramientas de depuración. Principalmente provee una barra de herramientas en el HTML renderizado, que porporciona una gran cantidad de información sobre tu aplicación y la solicitud actual. Ver el capítulo *Debug Kit* para saber cómo instalar y usar DebugKit.

CAPÍTULO 17

ES - Deployment

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

Por favor, siéntase libre de enviarnos un pull request en Github[136] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

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

CAPÍTULO 18

Email

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

Por favor, siéntase libre de enviarnos un pull request en Github[137] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

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

CAPÍTULO 19

Error & Exception Handling

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

Por favor, siéntase libre de enviarnos un pull request en Github[138] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

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

CAPÍTULO 20

Events System

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

Por favor, siéntase libre de enviarnos un pull request en Github[139] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

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

CAPÍTULO 21

Internationalization & Localization

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

Por favor, siéntase libre de enviarnos un pull request en Github[140] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

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

CAPÍTULO 22

Logging

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

Por favor, siéntase libre de enviarnos un pull request en Github[141] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

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

CAPÍTULO 23

Modelless Forms

class Cake\
Form**Form**

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

Por favor, siéntase libre de enviarnos un pull request en Github[142] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

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

CAPÍTULO 24

Plugins

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

Por favor, siéntase libre de enviarnos un pull request en Github[143] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

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

CAPÍTULO 25

REST

Muchos de los nuevos programadores de aplicaciones se estan dando cuenta de la necesidad de abrir el núcleo de la funcionalidad a un mayor publico. Proporcionando acceso fácil y sin restricciones al núcleo de su API puede ayudar a que su plataforma sea aceptada, y permite realizar mashups y fácil integración con otros sistemas.

Si bien existen otras soluciones, REST es una excelente manera de proporcionar un fácil acceso a la lógica que ha creado para su aplicación. Es simple, generalmente basado en XML (estamos hablando de simple XML, nada como un envoltorio de SOAP), y depende de los encabezados HTTP por dirección. Exponer una API utilizando REST en CakePHP es simple.

La Configuración Simple

La forma más rapida para empezar a utilizar REST es agregar unas lineas para configurar la *resource routes* <*resource-routes*> en su archivo **config/routes.php** .

Una vez que la ruta se ha configurado para mapear las solicitudes REST a cierto controlador de acciones, se puede proceder a crear la lógica de nuestro controlador de acciones. Un controlador básico podría visualizarse de la siguiente forma:

```
// src/
↪Controller/
↪RecipesController.
↪php
class␣
↪RecipesController␣
↪extends␣
↪AppController
{
    public␣
↪function␣
```

(continué en la próxima página)

(proviene de la página anterior)

```
↪initialize():␣
↪void
    {
        ␣
↪parent::initialize();
↪

↪$this->
↪loadComponent(
↪'RequestHandler
↪');
    }

    public␣
↪function␣
↪index()
    {

↪$recipes␣
↪= $this->
↪Recipes->
↪find('all
↪');

↪$this->
↪set(
↪'recipes',
↪
↪$recipes);

↪$this->
↪viewBuilder()-
↪>
↪setOption(
↪'serialize
↪', [
↪'recipes
↪']);
    }

    public␣
↪function␣
↪view($id)
    {

↪$recipe =
↪$this->
↪Recipes->
↪get($id);

↪$this->
↪set(
```

(continué en la próxima página)

(proviene de la página anterior)

```
↪'recipe',
↪$recipe);

↪$this->
↪viewBuilder()-
↪>
↪setOption(
↪'serialize
↪', [
↪'recipe
↪']);
    }

    public␣
↪function␣
↪add()
    {

↪$this->
↪request->
↪allowMethod([
↪'post',
↪'put']);

↪$recipe =
↪$this->
↪Recipes->
↪newEntity(
↪$this->
↪request->
↪getData());
↪
        if (
↪$this->
↪Recipes->
↪save(
↪$recipe))
↪{

↪$message␣
↪= 'Saved';
        }␣
↪else {

↪$message␣
↪= 'Error';
        }

↪$this->
↪set([

↪'message'␣
```

(continué en la próxima página)

(proviene de la página anterior)

```
↪=>
↪$message,

↪'recipe'␣
↪=>
↪$recipe,
        ]);

↪$this->
↪viewBuilder()-
↪>
↪setOption(
↪'serialize
↪', [
↪'recipe',
↪'message
↪']);
    }

    public␣
↪function␣
↪edit($id)
    {

↪$this->
↪request->
↪allowMethod([
↪'patch',
↪'post',
↪'put']);

↪$recipe =
↪$this->
↪Recipes->
↪get($id);

↪$recipe =
↪$this->
↪Recipes->
↪patchEntity(
↪$recipe,
↪$this->
↪request->
↪getData());
↪
        if (
↪$this->
↪Recipes->
↪save(
↪$recipe))
↪{
```

(continué en la próxima página)

(proviene de la página anterior)

```
↪$message␣
↪= 'Saved';
        }␣
↪else {

↪$message␣
↪= 'Error';
        }

↪$this->
↪set([

↪'message'␣
↪=>
↪$message,

↪'recipe'␣
↪=>
↪$recipe,
        ]);

↪$this->
↪viewBuilder()-
↪>
↪setOption(
↪'serialize
↪', [
↪'recipe',
↪'message
↪']);
    }

    public␣
↪function␣
↪delete(
↪$id)
    {

↪$this->
↪request->
↪allowMethod([
↪'delete
↪']);

↪$recipe =
↪$this->
↪Recipes->
↪get($id);

↪$message␣
↪= 'Deleted
↪';
```

(continué en la próxima página)

(proviene de la página anterior)

```
        if␣
↪(!$this->
↪Recipes->
↪delete(
↪$recipe))
↪{

↪$message␣
↪= 'Error';
        }

↪$this->
↪set(
↪'message',
↪
↪$message);

↪$this->
↪viewBuilder()-
↪>
↪setOption(
↪'serialize
↪', [
↪'message
↪']);
    }
}
```

Los controladores RESTful a menudo usan extensiones parseadas para mostrar diferentes vistas basado en diferentes tipos de solicitudes. Como estamos tratando con solicitudes REST, estaremos haciendo vistas XML. Puedes realizar vistas en JSON usando el CakePHP *Vistas JSON y XML*. Mediante el uso de `XmlView` se puede definir una opción de `serialize`. Esta opción se usa para definir qué variables de vistas `` XmlView`` deben serializarse en XML.

Si se quiere modificar los datos antes de convertirlos en XML, no se debería definir la opción `serialize`, y en lugar de eso, se debería usar archivos plantilla. Colocaremos las vistas REST de nuestro RecipesController dentro de **templates/Recipes/xml**. también podemos utilizar el `Xml` para una salida XML rápida y fácil en esas vistas. De esta forma, así podría verse nuestra vista de índice:

```
//␣
↪templates/
↪Recipes/
↪xml/index.
↪php
// Realizar␣
↪un␣
↪formateo␣
↪y␣
↪manipulacion␣
↪en
// $recipes␣
↪array.
$xml =␣
↪Xml::fromArray([
```

(continué en la próxima página)

(proviene de la página anterior)

```
↪'response
↪' =>
↪$recipes]);
↪
echo $xml->
↪asXML();
```

Al entregar un tipo de contenido específico usando `Cake\Routing\Router::extensions()`, CakePHP busca automáticamente un asistente de vista que coincida con el tipo. Como estamos utilizando XML como tipo de contenido, no hay un asistente incorporado, sin embargo, si creara uno, se cargaría automáticamente para nuestro uso en esas vistas.

El XML procesado terminará pareciéndose a esto:

```
<recipes>
    <recipe>
        <id>
↪234</id>

↪<created>
↪2008-06-13
↪</created>

↪<modified>
↪2008-06-14
↪</
↪modified>

↪<author>

↪<id>23423
↪</id>

↪<first_
↪name>Billy
↪</first_
↪name>

↪<last_
↪name>Bob</
↪last_name>
        </
↪author>

↪<comment>

↪<id>245</
↪id>

↪<body>
↪Yummy␣
↪yummmy</
↪body>
        </
```

(continué en la próxima página)

(proviene de la página anterior)

```
↪comment>
    </
↪recipe>
    ...
</recipes>
```

Crear la lógica para la acción de edición es un poco más complicado, pero no mucho. Ya que se está proporcionando una API que genera XML como salida, es una opción natural recibir XML como entrada. No te preocupes, las clases `Cake\Controller\Component\RequestHandler` y *`Cake\Routing\Router`* hacen las cosas mucho mas fáciles. Si un POST o una solicitud PUT tiene un tipo de contenido XML, entonces la entrada se ejecuta a través de la clase de CakePHP `Xml`, y la representación del arreglo de los datos se asigna a `$this->request->getData()`. Debido a esta característica, el manejo de datos XML y POST se hace en continuamente en paralelo: no se requieren cambios en el controlador o el código del modelo. Todo lo que necesita debe terminar en `$this->request->getData()`.

Aceptando Entradas en otros formatos

Por lo general, las aplicaciones REST no solo generan contenido en formatos de datos alternativos, sino que también acepta datos en diferentes formatos. En CakePHP, el `RequestHandlerComponent` ayuda a facilitar esto. Por defecto, decodificará cualquier entrada de datos en JSON / XML para solicitudes POST / PUT y proporcionar una versión del arreglo de esos datos en `$this->request->getData()`. También puedes conectar deserializadores adicionales para formatos alternativos si los necesitas, usando: `RequestHandler::addInputType()`.

Enrutamiento RESTful

El enrutador de CakePHP fácilita la conexión de rutas de recursos RESTful. Ver la sección *resource-routes* para más información.

CAPÍTULO 26

Security

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

Por favor, siéntase libre de enviarnos un pull request en Github[144] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

Security

```
class Cake\
Utility\
Security
```

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

Por favor, siéntase libre de enviarnos un pull request en Github[145] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

[144] https://github.com/cakephp/docs
[145] https://github.com/cakephp/docs

Cross Site Request Forgery

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

Por favor, siéntase libre de enviarnos un pull request en Github[146] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

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

CAPÍTULO 27

Sessions

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

Por favor, siéntase libre de enviarnos un pull request en Github[147] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

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

CAPÍTULO 28

Testing

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

Por favor, siéntase libre de enviarnos un pull request en Github[148] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

Running Tests

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

CAPÍTULO 29

Validation

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

Por favor, siéntase libre de enviarnos un pull request en Github[149] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

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

CAPÍTULO 30

La clase App

class Cake\Core**App**

La clase App se encarga de la localización de recursos y de la administración de rutas.

Búsqueda de clases

static Cake\Core\App::**class**

Éste método se utiliza para resolver el nombre completo de una clase en todo Cakephp. Como parámetros del método entran los nombre cortos que usa CakePHP y devuelve el nombre completo (La ruta relativa al espacio de trabajo):

```
// Resuelve␣
↪el nombre␣
↪de clase␣
↪corto␣
↪utilizando␣
↪el nombre␣
↪y el␣
↪sufijo.
```

(continué en la próxima página)

(proviene de la página anterior)

```
App::classname(
↪'Auth',
↪'Controller/
↪Component
↪',
↪'Component
↪');
// Salida:␣
↪Cake\
↪Controller\
↪Component\
↪AuthComponent

// Resuelve␣
↪el nombre␣
↪de plugin.
App::classname(
↪'DebugKit.
↪Toolbar',
↪'Controller/
↪Component
↪',
↪'Component
↪');
// Salida:␣
↪DebugKit\
↪Controller\
↪Component\
↪ToolbarComponent

// Nombres␣
↪con '\' se␣
↪devuelven␣
↪inalterados.
↪
App::classname(
↪'App\
↪Cache\
↪ComboCache
↪');
// Salida:␣
↪App\Cache\
↪ComboCache
```

A la hora de resolver clases, primero se prueba con el espacio de nombres de `App`, si no existe, se prueba con el espacio de nombres de `Cake` . Si no existe ninguno, devuelve `false`.

Búsqueda de rutas al espacio de nombres

static Cake\Core\App::**path**

Se usa para la búsqueda de rutas basada en convenio de nombres de CakePHP:

```
// Buscar
↪la ruta
↪de
↪Controller/
↪ en tu
↪aplicación
App::path(
↪'Controller
↪');
```

Se puede utilizar para todos los espacios de nombres de tu aplicacón. Además puedes extraer rutas de plugins:

```
// Devuelve
↪la ruta
↪del
↪'Component
↪' en
↪DebugKit
App::path(
↪'Component
↪',
↪'DebugKit
↪');
```

`App::path()` sólo devuelve la ruta por defecto,no mostrará ningún tipo de información sobre las rutas adicionales configuadas en autoloader.

static Cake\Core\App::**core**

Se usa para buscar rutas de paquetes dentro del core de Cakephp:

```
// Devuelve
↪la ruta
↪de engine
↪de cake.
App::core(
```

(continué en la próxima página)

(proviene de la página anterior)

```
↪'Cache/
↪Engine');
```

Búsqueda de plugins

```
static Cake\Core\Plugin::pa
```

Los plugins se localizan con el método Plugin. Por ejemplo, `Plugin::path('DebugKit');` devuelve la ruta completa al plugin DebugKit:

```
$path =␣
↪Plugin::path(
↪'DebugKit
↪');
```

Localización de temas (nota:"themes")

Dado que los temas (nota:"themes") son también plugins, se localizan con el método anterior, «Plugin». (nota:"Aquí se refiere a los themes que se pueden crear para modificar el comportamiento del bake, generador de código.")

Cargar archivos externos (nota: "vendor")

Lo ideal es que los archivos externos ("vendor") se carguen automáticamente usando `Composer`, si necesita archivos externos que no se pueden cargar automáticamente o no se pueden instalar con el Composer, entonces hay que usar `require` para cargarlos.

Si no puede instalar alguna librería con el Composer, debería instalar cada librería en el directorio apropiado, siguiendo el convenio del Composer: `vendor/$author/$package`. Si tiene una librería de autor "Acme" que se llama "AcmeLib", la tiene que instalar en: `vendor/Acme/AcmeLib`. Asumiendo que la librería no usa nombres de clase compatibles con "PSR-0", puede cargar las clases definiéndolas en el `classmap`, dentro del archivo: `composer.json` en su aplicación:

```
"autoload":
↪{
    "psr-4
↪": {

↪"App\\":
↪"App",

↪"App\\
↪Test\\":
↪"Test",
        "":
↪"./Plugin"
```

(continué en la próxima página)

(proviene de la página anterior)

```
    },

↪"classmap
↪": [

↪"vendor/
↪Acme/
↪AcmeLib"
    ]
}
```

Si la librería no usa clases y sólo proporciona métodos,puede configurar el Composer para que cargue esos archivos al inicio de cada petición("request"), usando la estrategia de carga automática de ficheros `files`, como sigue:

```
"autoload":
↪{
    "psr-4
↪": {

↪"App\\":
↪"App",

↪"App\\
↪Test\\":
↪"Test",
        "":
↪"./Plugin"
    },
    "files
↪": [

↪"vendor/
↪Acme/
↪AcmeLib/
↪functions.
↪php"
    ]
}
```

Después de la configuración de las librerías externas, tiene que regenerar el autoloader de su aplicación usando:

```
$ php␣
↪composer.
↪phar dump-
↪autoload
```

Si no usa Composer en su aplicación, tendrá que cargar manualmente cada librería en su aplicación.

CAPÍTULO 31

Collections

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

Por favor, siéntase libre de enviarnos un pull request en Github[150] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

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

CAPÍTULO 32

Folder & File

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

Por favor, siéntase libre de enviarnos un pull request en Github[151] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

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

CAPÍTULO 33

Hash

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

Por favor, siéntase libre de enviarnos un pull request en Github[152] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

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

CAPÍTULO 34

Http Client

```
class Cake\Network\Http\Cli
```

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

Por favor, siéntase libre de enviarnos un pull request en Github[153] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

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

CAPÍTULO 35

Inflector

class Cake\
Utility\
Inflector

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

Por favor, siéntase libre de enviarnos un pull request en Github[154] o utilizar el botón **Improve this Doc** para proponer directamente los cambios.

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

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

CAPÍTULO 36

Number

```
class Cake\
I18n\Number
```

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[155] 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.

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

CAPÍTULO 37

Registry Objects

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

Por favor, siéntase libre de enviarnos un pull request en Github[156] 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.

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

CAPÍTULO 38

Text

```
class Cake\
Utility\Text
```

```
static Cake\
Utility\
Text::uuid
```

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[157] 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.

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

CAPÍTULO 39

Time

```
class Cake\
Utility\Time
```

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[158] 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.

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

CAPÍTULO 40

Xml

```
class Cake\
Utility\Xml
```

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

Por favor, siéntase libre de enviarnos un pull request en Github[159] 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.

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

CAPÍTULO 41

Constants & Functions

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

Por favor, siéntase libre de enviarnos un pull request en Github[160] 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.

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

CAPÍTULO 42

Debug Kit

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[161] 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.

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

CAPÍTULO 43

Migrations

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

Por favor, siéntase libre de enviarnos un pull request en Github[162] 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.

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

CAPÍTULO 44

Apéndices

En los apéndices encontrarás información relacionada a las nuevas características introducidas en cada versión, así como también las guías de migración entre versiones.

Guía de Migración a 4.x

Información General

CakePHP Development Process

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

Por favor, siéntase libre de enviarnos un pull request en Github[163] 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.

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

Glosario

array de rutas
: Un array de atributos que son pasados a `Router::url()`. Típicamente se ve algo así:

```
[
↪'controller
↪' =>
↪'Posts
↪',
↪'action
↪' =>
↪'view
↪', 5]
```

Atributos HTML
: Un array con claves => valores que son colocados en los atributos HTML. Por ejemplo:

```
// Dado
['class
↪' =>
↪'mi-
↪clase
↪',
↪'target
↪' =>
↪'_
↪blank
↪']

//␣
↪Generará
class=
↪"mi-
```

(continué en la próxima página)

(proviene de la página anterior)

```
↪clase
↪"␣
↪target=
↪"_
↪blank
↪"
```

Si una opción puede usar su nombre como valor, entonces puede ser usado `true`:

```
// Dado
[
↪'checked
↪' =>␣
↪true]

//␣
↪Generará
checked=
↪"checked
↪"
```

Sintaxis de plugin

La sintáxis de plugin se refiere a el punto que separa los nombres de clases indicando que la clase es parte de un plugin:

```
// El␣
↪plugin␣
↪es
↪"DebugKit
↪", y␣
↪el␣
```

(continué en la próxima página)

(proviene de la página anterior)

```
    ↪nombre␣
    ↪de␣
    ↪la␣
    ↪clase␣
    ↪es
    ↪"Toolbar
    ↪".

    ↪'DebugKit.
    ↪Toolbar
    ↪'

// El␣
    ↪plugin␣
    ↪es
    ↪"AcmeCorp/
    ↪Tools
    ↪", y␣
    ↪el␣
    ↪nombre␣
    ↪de␣
    ↪clase␣
    ↪es
    ↪"Toolbar
    ↪".

    ↪'AcmeCorp/
    ↪Tools.
    ↪Toolbar
    ↪'
```

Notación de punto

La notación de punto define un array de rutas, separando los niveles anidados con `.` Por ejemplo:

```
Cache.
    ↪default.
    ↪engine
```

Apuntará al siguiente

valor:

```
[
    'Cache' => [
        'default' => [
            'engine' => 'File'
        ]
    ]
]
```

CSRF
: *Cross Site Request Forgery*. Previene los ataques de replay o playback, peticiones duplicadas y peticiones falsificadas desde otros dominios.

CDN
: *Content Delivery Network*. Le puedes pagar a un proveedor para que ayude a distribuir el contenido a centros de datos

alrededor del mundo. Esto ayuda a poner elementos estáticos más cerca de tus usuarios geográficamente.

routes.php
: Un archivo en el directorio `config` que contiene las configuraciones de enrutamiento. Este archivo se incluye antes de que cada petición sea procesada. Se deben conectar todas las rutas que necesita tu aplicación para que cada petición sea enrutada correctamente al controlador + acción.

DRY
: *Don't repeat yourself*. Es un principio

de desarrollo de software orientado a reducir la repetición de la información de todo tipo. En CakePHP, DRY se utiliza para que se pueda escribir las cosas una vez y reutilizarlos a través de su aplicación.

PaaS
: *Platform as a Service*. La plataforma como servicio proporcionará recursos de hosting, bases de datos y almacenamiento en caché basado en la nube. Algunos proveedores populares incluyen Heroku, EngineYard y PagodaBox.

DSN
: *Data Source Name*. Una cadena de conexión formateada para que sea como una URI. CakePHP soporta conexiones DSN para Caché, Base de datos, Registro y de E-mail.

PHP Namespace Index

C

Índice

Símbolos

A

B

C

D

E

F

G

H

I

J

L

M

N

P

Q

R

S

T

U

V

X