CakePHP Cookbook Documentation

Release 2.x

Cake Software Foundation

August 06, 2012

Contents

CHAPTER 1

Getting Started

The CakePHP framework provides a robust base for your application. It can handle every aspect, from the user's initial request all the way to the final rendering of a web page. And since the framework follows the principles of MVC, it allows you to easily customize and extend most aspects of your application.

The framework also provides a basic organizational structure, from filenames to database table names, keeping your entire application consistent and logical. This concept is simple but powerful. Follow the conventions and you'll always know exactly where things are and how they're organized.

The best way to experience and learn CakePHP is to sit down and build something. To start off we'll build a simple blog application.

CHAPTER 2

Blog Tutorial

Welcome to CakePHP. You're probably checking out this tutorial because you want to learn more about how CakePHP works. It's our aim to increase productivity and make coding more enjoyable: we hope you'll see this as you dive into the code.

This tutorial will walk you through the creation of a simple blog application. We'll be getting and installing Cake, creating and configuring a database, and creating enough application logic to list, add, edit, and delete blog posts.

Here's what you'll need:

1. A running web server. We're going to assume you're using Apache, though the instructions for using other servers should be very similar. We might have to play a little with the server configuration, but most folks can get Cake up and running without any configuration at all. Make sure you have PHP 5.2.8 or greater.
2. A database server. We're going to be using MySQL server in this tutorial. You'll need to know enough about SQL in order to create a database: Cake will be taking the reins from there. Since we're using MySQL, also make sure that you have `pdo_mysql` enabled in PHP.
3. Basic PHP knowledge. The more object-oriented programming you've done, the better: but fear not if you're a procedural fan.
4. Finally, you'll need a basic knowledge of the MVC programming pattern. A quick overview can be found in *Understanding Model-View-Controller*. Don't worry, it's only a half a page or so.

Let's get started!

Getting Cake

First, let's get a copy of fresh Cake code.

To get a fresh download, visit the CakePHP project on GitHub: http://github.com/cakephp/cakephp/downloads and download the latest release of 2.0

You can also clone the repository using git (http://git-scm.com/). `git clone git://github.com/cakephp/cakephp.git`

Regardless of how you downloaded it, place the code inside of your DocumentRoot. Once finished, your directory setup should look something like the following:

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

Now might be a good time to learn a bit about how Cake's directory structure works: check out *CakePHP Folder Structure* section.

Creating the Blog Database

Next, lets set up the underlying database for our blog. if you haven't already done so, create an empty database for use in this tutorial, with a name of your choice. Right now, we'll just create a single table to store our posts. We'll also throw in a few posts right now to use for testing purposes. Execute the following SQL statements into your database:

```
/* First, create our posts table: */
CREATE TABLE posts (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(50),
    body TEXT,
    created DATETIME DEFAULT NULL,
    modified DATETIME DEFAULT NULL
);

/* Then insert some posts for testing: */
INSERT INTO posts (title,body,created)
    VALUES ('The title', 'This is the post body.', NOW());
INSERT INTO posts (title,body,created)
    VALUES ('A title once again', 'And the post body follows.', NOW());
INSERT INTO posts (title,body,created)
    VALUES ('Title strikes back', 'This is really exciting! Not.', NOW());
```

The choices on table and column names are not arbitrary. If you follow Cake's database naming conventions, and Cake's class naming conventions (both outlined in *CakePHP Conventions*), you'll be able to take advantage of a lot of free functionality and avoid configuration. Cake is flexible enough to accommodate even the worst legacy database schema, but adhering to convention will save you time.

Check out *CakePHP Conventions* for more information, but suffice it to say that naming our table 'posts' automatically hooks it to our Post model, and having fields called 'modified' and 'created' will be automagically managed by Cake.

Cake Database Configuration

Onward and upward: let's tell Cake where our database is and how to connect to it. For many, this is the first and last time you configure anything.

A copy of CakePHP's database configuration file is found in `/app/Config/database.php.default`. Make a copy of this file in the same directory, but name it `database.php`.

The config file should be pretty straightforward: just replace the values in the `$default` array with those that apply to your setup. A sample completed configuration array might look something like the following:

```php
<?php
public $default = array(
    'datasource' => 'Database/Mysql',
    'persistent' => false,
    'host' => 'localhost',
    'port' => '',
    'login' => 'cakeBlog',
    'password' => 'c4k3-rUl3Z',
    'database' => 'cake_blog_tutorial',
    'schema' => '',
    'prefix' => '',
    'encoding' => ''
);
```

Once you've saved your new `database.php` file, you should be able to open your browser and see the Cake welcome page. It should also tell you that your database connection file was found, and that Cake can successfully connect to the database.

Note: Remember that you'll need to have PDO, and pdo_mysql enabled in your php.ini.

Optional Configuration

There are three other items that can be configured. Most developers complete these laundry-list items, but they're not required for this tutorial. One is defining a custom string (or "salt") for use in security hashes. The second is defining a custom number (or "seed") for use in encryption. The third item is allowing CakePHP write access to its `tmp` folder.

The security salt is used for generating hashes. Change the default salt value by editing `/app/Config/core.php` line 187. It doesn't much matter what the new value is, as long as it's not easily guessed.

```php
<?php
/**
 * A random string used in security hashing methods.
 */
Configure::write('Security.salt', 'pl345e-P45s_7h3*S@l7!');
```

The cipher seed is used for encrypt/decrypt strings. Change the default seed value by editing `/app/Config/core.php` line 192. It doesn't much matter what the new value is, as long as it's not easily guessed.

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

The final task is to make the `app/tmp` directory web-writable. The best way to do this is to find out what user your webserver runs as (`<?php echo `whoami`; ?>`) and change the ownership of the `app/tmp` directory to that user. The final command you run (in *nix) might look something like this:

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

If for some reason CakePHP can't write to that directory, you'll be informed by a warning while not in production mode.

A Note on mod_rewrite

Occasionally a new user will run in to mod_rewrite issues, so I'll mention them marginally here. If the CakePHP welcome page looks a little funny (no images or css styles), it probably means mod_rewrite isn't functioning on your system. Here are some tips to help get you up and running:

1. Make sure that an .htaccess override is allowed: in your httpd.conf, you should have a section that defines a section for each Directory on your server. Make sure the `AllowOverride` is set to `All` for the correct Directory. For security and performance reasons, do *not* set `AllowOverride` to `All` in `<Directory />`. Instead, look for the `<Directory>` block that refers to your actual website directory.

2. Make sure you are editing the correct httpd.conf rather than a user- or site-specific httpd.conf.

3. For some reason or another, you might have obtained a copy of CakePHP without the needed .htaccess files. This sometimes happens because some operating systems treat files that start with '.' as hidden, and don't copy them. Make sure your copy of CakePHP is from the downloads section of the site or our git repository.

4. Make sure Apache is loading up mod_rewrite correctly! You should see something like:

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

   or (for Apache 1.3):

   ```
   AddModule             mod_rewrite.c
   ```

   in your httpd.conf.

If you don't want or can't get mod_rewrite (or some other compatible module) up and running on your server, you'll need to use Cake's built in pretty URLs. In `/app/Config/core.php`, uncomment the line that looks like:

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

Also remove these .htaccess files:

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

This will make your URLs look like www.example.com/index.php/controllername/actionname/param rather than www.example.com/controllername/actionname/param.

If you are installing CakePHP on a webserver besides Apache, you can find instructions for getting URL rewriting working for other servers under the *Advanced Installation* section.

Continue to *Blog Tutorial - Adding a layer* to start building your first CakePHP application.