Rapid Web Development mit dem MVC-Framework

2. Auflage
Aktuell zu CakePHP 1.3

Webentwicklung mit CakePHP

Dirk Ammelburger
& Robert Scherer

O'REILLY®

Inhalt

KAPITEL 3

Schnell zum Erfolg – Ihre erste Webapplikation mit CakePHP

Da wären wir nun: Die Grundlagen für CakePHP sind vorhanden, der Rechner läuft, ein Webserver mit PHP und einer Datenbank ist vorhanden, und der Cursor blinkt erwartungsvoll auf einer leeren Seite im geöffneten Editor. Wie geht es weiter? Wie versprochen: direkt mit ein wenig Praxis und der Magie von CakePHP. Im Folgenden werden wir Ihnen zeigen, wie Sie mit wenigen Zeilen Code eine komplette Applikation erzeugen, die auf dem Standardverhalten von CakePHP und den zugrunde liegenden Konventionen basiert. Sie werden überrascht sein, wie wenig Code Sie benötigen, um ein bemerkenswertes Ergebnis zu erhalten.

In diesem Kapitel werden Sie die Grundlage der Beispielapplikation entwickeln, die in Kapitel 1, *CakePHP kennenlernen*, vorgestellt worden ist. Außer der Definition der Datenbanktabellen werden Sie in diesem Kapitel viel Arbeit dem Framework überlassen und mit wenigen Schritten durch die Automatismen des Frameworks zum Ziel gelangen.

Aber genug der Vorrede, jetzt soll das Beispiel für sich sprechen. Los geht's!

Die Datenbank als Grundlage der Applikation

Die Grundlage für jede Applikation ist eine Datenbank, in der die Daten der Anwendung gespeichert werden. CakePHP unterstützt, wie schon eingangs erwähnt, eine ganze Reihe von verschiedenen Datenbanken. Der Standard ist MySQL, dem wir werden auch in diesem Beispiel treu bleiben werden. Zur leichteren Bedienung empfiehlt es sich, auf *phpMyAdmin* oder eine ähnliche Oberfläche zuzugreifen, mit der die Datenbank bequem bedient werden kann. Die XAMPP-Installation hat *phpMyAdmin* in der aktuellen Version immer standardmäßig installiert.

Die Idee unseres Beispiels ist simpel: Es soll eine einfache Applikation entwickelt werden, in der User auf einer Webseite Jobanzeigen hinterlassen können. Im ersten Schritt soll nicht viel mehr möglich sein als das Anlegen von neuen Einträgen, eine Editierfunktion für bestehende Einträge und die Möglichkeit, nicht mehr benötigte Einträge zu löschen. Das klingt einfach, und genauso sollte es auch sein.

Eine neue Datenbanktabelle

Eine große Planung der Anwendung ist bei dem Umfang nicht nötig, darum können Sie direkt mit der Datenstruktur beginnen. In unserem Fall reicht eine einzelne Tabelle, die alle Daten eines Eintrags – im folgenden *Jobs* genannt – enthält. Neben einer eindeutigen ID sollen folgende Felder definiert werden:

Spalte	Inhalt	Spaltenname
ID	Eindeutige ID, über die der Eintrag identifiziert wird	`id`
Titel	Der Titel des Eintrags	`title`
Autor	Der Name des Autors	`name`
Inhalt	Der Inhalt (Es darf auch ein wenig mehr sein.)	`txt`
Erstellungsdatum und -uhrzeit	Wann wurde der Eintrag erstellt?	`created`
Änderungsdatum und -uhrzeit	Wann wurde der Eintrag zum letzten Mal geändert?	`modified`

In SQL sieht der `Create`-Befehl dann so aus:

```
CREATE TABLE IF NOT EXISTS `jobs` (
   `id` int(11) NOT NULL auto_increment,
   `company` varchar(128) NOT NULL,
   `title` varchar(128) NOT NULL,
   `description` text NOT NULL,
   `created` datetime NOT NULL,
   `modified` datetime NOT NULL,
   PRIMARY KEY  (`id`)
);
```

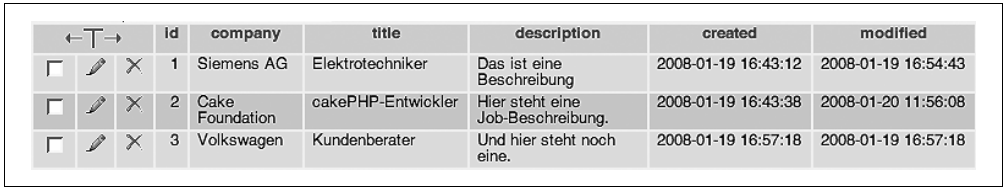
Wenn Sie diesen Befehl an die Datenbank schicken, wird eine Tabelle mit dem Namen *jobs* angelegt, in der Sie die einzelnen *Jobs* der Benutzer abspeichern. Um das Datengerüst noch mit etwas Inhalt zu füllen, schreiben Sie ein paar Testdatensätze:

```
INSERT INTO `jobs`
   (`id`, `company`, `title`, `description`, `created`, `modified`)
VALUES
   (1, 'Siemens AG', 'Elektrotechniker', 'Das ist eine Beschreibung',
       NOW(), NOW()),
   (2, 'Cake Foundation', 'cakePHP-Entwickler', 'Hier steht eine Job-Beschreibung.',
       NOW(), NOW()),
   (3, 'Volkswagen', 'Kundenberater', 'Und hier steht noch eine.',
         NOW(), NOW());
```

phpMyAdmin honoriert Ihre Bemühungen mit dem Anblick, den Sie in Abbildung 3-1 sehen.

Das war es im Prinzip schon zum Thema Datenbank. Wesentlich an dieser Stelle sind nur zwei Punkte: zum einen die eindeutige ID, die Sie mit dem Bezeichner *id* in der Tabelle definiert haben. Damit haben Sie schon eine Konvention von CakePHP eingehalten, da das Framework davon ausgeht, dass in einer Tabelle der Primary Key immer *id* genannt

wird. Der zweite Punkt ist der Name der Tabelle, über den automatisch das dazugehörige Model identifiziert wird. Dazu mehr im nächsten Abschnitt.



←T→	id	company	title	description	created	modified
☐ ✎ ✕	1	Siemens AG	Elektrotechniker	Das ist eine Beschreibung	2008-01-19 16:43:12	2008-01-19 16:54:43
☐ ✎ ✕	2	Cake Foundation	cakePHP-Entwickler	Hier steht eine Job-Beschreibung.	2008-01-19 16:43:38	2008-01-20 11:56:08
☐ ✎ ✕	3	Volkswagen	Kundenberater	Und hier steht noch eine.	2008-01-19 16:57:18	2008-01-19 16:57:18

Abbildung 3-1: Die Datenbanktabelle jobs

Das Model für die Datenbanktabelle

Die Basis unserer Applikation ist die Datenbanktabelle, die Sie im vorangegangenen Abschnitt definiert haben. Da CakePHP in der Konfigurationsdatei die Zugangsdaten zur Datenbank übergeben bekommt, ist der Zugriff auf diese Daten über das Framework ohne Weiteres möglich. Sie müssen sich also nur noch um den Umgang mit den Daten in der Applikationslogik kümmern. In der Regel wird für jede Datenbanktabelle ein Model angelegt, das die Daten in der Applikation repräsentiert. Basierend auf dem MVC-Pattern, das gleich noch genauer besprochen wird, wird die Datenbanktabelle in Ihrer Applikation durch eine Klasse dargestellt. Das geschieht ganz einfach, indem eine bestehende Klasse aus dem Framework abgeleitet wird und eine spezialisierte Subklasse für die Datenbanktabelle *jobs* entsteht.

Legen Sie eine Datei mit dem Namen *job.php* im Unterordner *app/models/* Ihrer CakeInstallation im *htdocs*-Verzeichnis des Webservers an. Der Name orientiert sich an dem Tabellenamen *jobs*, den Sie festgelegt haben – mit dem Unterschied, dass Sie jetzt den englischen Singular `job` wählen. Sie werden im Laufe des Buches feststellen, dass sich die Datei- und Klassennamen von CakePHP an der englischen Grammatik orientieren. Um diese Konventionen einhalten zu können, ohne merkwürdig klingende Namen verwenden zu müssen, empfehlen wir, alle Bezeichner ebenfalls auf Englisch zu wählen.

Öffnen Sie die erstellte Datei und geben Sie den folgenden Quelltext für das Model ein:

```
<?php
class Job extends AppModel {
   public $name = 'Job';

}
```

Speichern Sie diese Datei ab. Nun sollten Sie die Datei *app/models/job.php* im CakePHP-Installationsordner des Webservers haben. Diese Klasse repräsentiert das Datenmodell, das Sie in der Datenbank definiert haben. Die Verknüpfung zur Datenbanktabelle erfolgt automatisch durch die CakePHP-Konventionen, die Sie durch den Klassennamen `job` eingehalten haben. In Kapitel 5, *Das Model*, erfahren Sie mehr zu diesem Thema.

Daten mit dem Controller verarbeiten

Jetzt haben Sie ein Datenmodell, mit dem CakePHP arbeiten kann. Im nächsten Schritt müssen Sie einen Controller erstellen, der die Applikationslogik enthält. Die Datei ist notwendig, auch wenn in unserem Beispiel keine Logik implementiert wird. Auf diese Weise kommt das Standardverhalten des Frameworks zum Einsatz, wie das Beispiel zeigen wird.

Erstellen Sie eine Datei mit dem Namen *jobs_controller.php* im Unterordner *app/controller/* Ihrer Cake-Installation im *htdocs*-Verzeichnis des Webservers. Der Name orientiert sich auch in diesem Fall an den Namenskonventionen von CakePHP und ist so eindeutig auf das Model festgelegt. Bitte beachten Sie, dass in diesem Fall der Plural für die `Jobs` verwendet wird.

Öffnen Sie die erstellte Datei und geben Sie folgenden Quelltext für den Controller ein:

```
<?php
class JobsController extends AppController {
   public $name = 'Jobs';
   public $scaffold;
}
```

Schon sind Sie fertig – und zwar nicht nur mit dem Inhalt dieser Datei, sondern auch mit unserer kompletten Anwendung. Mehr ist in der Tat nicht nötig. Der Beweis wird im nächsten Abschnitt angetreten.

Die Anwendung starten und Gerüste bauen

Schon ist die Applikation fertig! Das glauben Sie nicht? Dann geben Sie folgende URL in Ihren Browser ein:

http://localhost/cakejobs/jobs

Wenn Sie die CakePHP-Installation in den *webroot*-Ordner des Webservers eingetragen haben, dann sollten Sie eine Ausgabe ähnlich wie die in Abbildung 3-2 erhalten.

Erstaunlich, nicht wahr? Sie haben nichts weiter gemacht, als einige Daten zu definieren, die den Konventionen von CakePHP entsprechen. Dazu waren nicht mehr als fünf Zeilen Code notwendig, die mit der Klassenstruktur von CakePHP arbeiten. Alles andere passiert an dieser Stelle automatisch (oder auch automagisch) durch das Framework, und Sie erhalten eine komplette Oberfläche zur Verwaltung der Datenbanktabelle.

Sie werden feststellen, dass die Darstellung und die Möglichkeiten des Default-Verhaltens des Frameworks nicht willkürlich sind, sondern der CRUD-Funktionalität entsprechen, die oben schon besprochen wurde. CakePHP hat auf Basis der wenigen Zeilen Code und des Frameworks im Hintergrund eine Oberfläche geschaffen, mit der die Daten in der Tabelle komplett verwaltet werden können. Das Standard-Stylesheet von CakePHP gibt dem Ganzen einen formalen Rahmen, so dass die Ausgabe in den Farben von Cake-

PHP erscheint. Wie und wo Sie diese Stylesheets ändern können, werden Sie in Kapitel 6, *Das View*, erfahren.

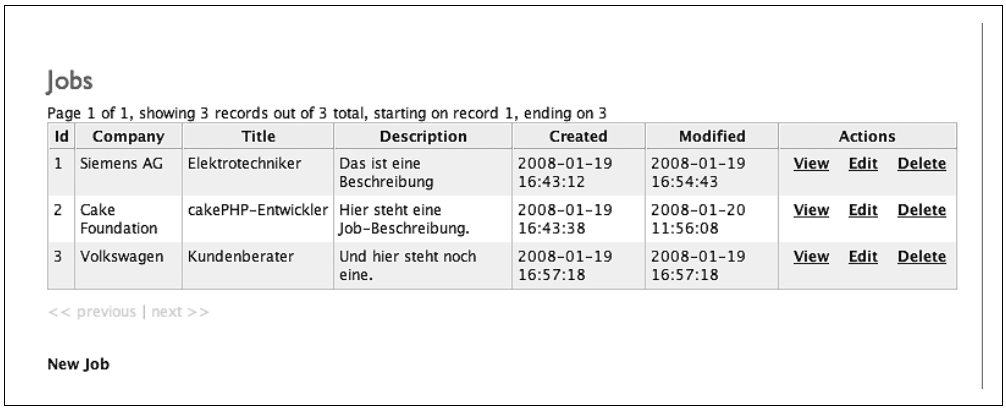


Abbildung 3-2: Das Ergebnis im Browser

Die erste Ansicht, die Sie vom Framework generiert bekommen, ist eine Übersicht über den kompletten Inhalt der Tabelle. Die ersten Spalten werden in einer Art Vorschau angezeigt, mit einer Reihe von möglichen Aktionen daneben. Die Datensätze können in einer Detailansicht aufgerufen werden, und sie können editiert und auch gelöscht werden. Sämtliche HTML-Ansichten für die einzelnen Aktionen sind im Framework hinterlegt und werden dynamisch geladen. Klicken Sie nun auf den *Edit*-Link eines beliebigen Datensatzes. Im Browser wird sich eine Ansicht öffnen, die der in Abbildung 3-3 ähnelt.

Abbildung 3-3: Die Editieransicht

Sie haben nun die Möglichkeit, alle Felder der Tabelle zu editieren und die bestehenden Werte zu ändern. Spielen Sie ein wenig mit den Möglichkeiten herum und ändern Sie die Beispieldatensätze, die hinterlegt worden sind. An dieser Stelle fällt auf, dass zwei Spalten fehlen, nämlich *created* und *modified*, die offensichtlich vom Framework nicht als editierbare Felder angeboten werden. Wenn Sie diese Ansicht mit der Detailseite (klicken Sie auf den *View*-Link) der *Posts*-Tabelle vergleichen (siehe Abbildung 3-4), dann wird allerdings klar, dass die Felder nicht einfach verschwunden sind, sondern an dieser Stelle erscheinen.

Abbildung 3-4: Die Detailansicht

Wenn Sie in Abbildung 3-4 genau hinschauen, fällt noch etwas auf: Das Framework zeigt die Felder zwar nicht an, setzt sie aber automatisch, wenn ein Datensatz über die Editieroberfläche geändert wird. Das Gleiche gilt für das Neuanlegen von Datensätzen: Das Feld *created* wird genau wie das Feld *modified* auf den aktuellen Zeitpunkt gesetzt. Editieren Sie den Datensatz erneut, wird nur das Feld *modified* auf die aktuelle Uhrzeit geändert.

Sie ahnen hier schon den Hintergrund: Die Feldnamen *created* und *modified* sind wieder entsprechend den CakePHP-Konventionen benannt worden, so dass an dieser Stelle ein Default-Verhalten des Frameworks greift. Allgemein gesprochen bedeutet das, dass CakePHP, sobald es in einer Datenbanktabelle auf Felder mit diesen Namen stößt, die entsprechenden Werte automatisch setzt und das auch in der Darstellung automatisch generierter Oberflächen berücksichtigt. Es ist möglich, dieses Verhalten zu überschreiben oder selbst ein derartiges Standardverhalten zu definieren. Auch dazu werden Sie im Laufe des Buches noch kommen.

Zuletzt ist an dieser Stelle noch erwähnenswert, dass CakePHP neben dem PHP-Code auch ein wenig JavaScript mit ins Spiel gebracht hat. Wenn Sie versuchen, einen Datensatz zu löschen, erscheint eine Sicherheitsabfrage, die Sie zuerst bestätigen müssen (siehe Abbildung 3-5).

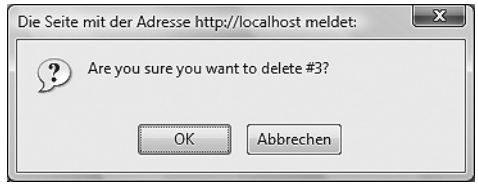


Abbildung 3-5: Eine JavaScript-Sicherheitsabfrage

Das CakePHP-Framework stellt Ihnen eine ganze Reihe von sogenannten HTML- bzw. Form-Helpern zur Verfügung, die es erlauben, im Template auf Funktionalitäten wie diese zurückzugreifen, ohne die Welt von CakePHP zu verlassen. In Kapitel 6, *Das View*, werden wir uns ausführlich mit diesen Möglichkeiten befassen.

Wenn Sie mithilfe von phpMyAdmin einen Blick in die Tabelle werfen, werden Sie sehen, dass alle Einträge tatsächlich so in der Datenbank gelandet sind. Nach allen Einträgen, Löschaktionen und Änderungen sollte die Tabelle einige Änderungen erfahren haben.

Während Sie die Eingaben im Browser gemacht haben, sind Ihnen sicher die SQL-Ausgaben unter der Tabelle aufgefallen. Jede Datenbankaktion, die im Hintergrund von CakePHP ausgeführt wird, erscheint automatisch im unteren Bereich der Seite. Diese Debug-Meldungen können Sie deaktivieren, indem Sie in der Datei *core.php* im Ordner *app/config* den Wert für `debug`auf 0 setzen. Während der Entwicklung sollten Sie diese Ausgaben allerdings bestehen lassen, da auf diese Weise die Fehlersuche wesentlich leichter fällt. Mehr zu den Debug- und Testmöglichkeiten von CakePHP folgt in Kapitel 16, *Testing*.

Bleibt an dieser Stelle nur noch eine Frage: Warum heißt dieser Abschnitt »Gerüste bauen«? Die Variable `$scaffold` (englisch für »Gerüst«) im Controller gibt uns den richtigen Hinweis: Wenn diese Variable im Controller für eine Model-Klasse definiert wird, wird automatisch die Ausgabe wie oben besprochen generiert. Das sogenannte Scaffolding – Gerüstebauen – erlaubt es also, auf Basis einer simplen Datenbanktabelle eine funktionsfähige Applikation zu generieren. Dieses Verhalten ist zwar für die letztendlich geplante Applikation in der Regel eigentlich nicht brauchbar, ermöglicht es uns aber, für den Prototyp einer Applikation schnell einen funktionsfähigen Dummy zu erstellen.

Sie als Entwickler haben dann die Aufgabe, dieses Gerüst nach und nach mit dem Aussehen und der Logik zu füllen, die vom Auftraggeber gewünscht sind.

Der Quelltext des Beispiels

Das ganze Skript besteht nur aus zwei Dateien, die jeweils nur wenige Zeilen Code enthalten. Für das bessere Verständnis der Grundlagen von CakePHP werden wir trotzdem

noch einmal einen Blick auf die beiden Dateien werfen, da sie sehr beispielhaft für den Aufbau größerer Applikationen sind.

Grundsätzlich kann man sagen, dass jede Datenbanktabelle in CakePHP durch eine korrespondierende Model-Klasse repräsentiert wird. Entsprechend den Konventionen des Frameworks werden der Name der Datei und auch der Name der Klasse gewählt, wobei zu beachten ist, dass das Model immer im Singular bezeichnet wird, im Gegensatz zur Tabelle, die CakePHP im Plural erwartet. Das Framework ist über den *Inflector*, eine Klasse zur Umsetzung der CakePHP-Namenskonventionen, in der Lage, automatisch unterschiedliche Schreibweisen nachzuvollziehen, auch wenn die Schreibweise unregelmäßig ist. Schönes Beispiel: Die Datenbanktabelle *Royalities* wird automatisch auf die Klasse `Royality` in der Datei *Royality.php* gematcht.

Genauso ist es in unserem Beispiel passiert: Die Klasse `Job` basiert auf der Framework-Klasse `AppModel` (bzw. `Model`) und übernimmt die Funktionalität des Frameworks, indem sie aus dieser Klasse abgeleitet wird. Mehr ist eigentlich nicht notwendig, wenn die oben beschriebenen Konventionen eingehalten werden. Die Definition der Variable `$name` geschieht nur aus Gründen der Kompatibilität zu PHP 4, da die Möglichkeiten der *Class reflections* erst mit der Version 5 eingeführt wurden.

```
<?php
class Job extends AppModel {
   public $name = 'Job';
}
```

Die Model-Klasse kann wesentlich mehr, als in diesem Beispiel gezeigt wurde. Sie werden im Laufe des Buches alle Möglichkeiten dieser Klasse kennenlernen, um sie dem gewünschten Verhalten Ihrer Applikation anzupassen.

Jede Model-Klasse hat in der Regel einen oder auch mehrere Controller, die auf die Daten zugreifen und sie verarbeiten. Entsprechend den CakePHP-Konventionen wird auch hier der Controller mit *Jobs* bezeichnet. Dabei heißt die Datei *jobs_controller.php* und die Klasse selbst `JobsController`. Diese Schreibweise nennt sich *CamelCase*, da jedes Wort innerhalb des Strings wieder großgeschrieben wird und so ein »buckliger« Eindruck entsteht, der an die Silhouette eines Kamels erinnert.

Wesentlich an dieser Stelle ist die Elternklasse `AppController`, die wiederum von der Framework-Klasse `Controller` abgeleitet ist. Auch hier wird das Framework in Ihre Applikation übernommen, indem Sie die Vererbungsmöglichkeiten der objektorientierten Programmierung nutzen.

```
<?php
class JobsController extends AppController {
   public $name = 'Jobs';
   public $scaffold;
}
```

In diesem Fall wird wieder eine Variable `$name` definiert, die den Bezeichner des Controllers speichert. Wie schon in der Model-Klasse geschieht das aus Gründen der Kompatibilität zu PHP 4. Darüber hinaus wird hier noch die Variable `$scaffold` definiert, die eigentlich nur die Aufgabe hat, als Flag dem Framework anzuzeigen, dass automatisch eine Prototypfunktionalität erfolgen soll. Wenn Sie diese Zeile entfernen, gibt der Controller lediglich eine leere Seite aus.

In den nächsten Kapiteln werden Sie den Controller, der das Herz der Applikationslogik darstellt, Stück für Stück mit Leben füllen und die Möglichkeiten von CakePHP in Interaktion mit Usereingaben erleben. An dieser Stelle werden Sie dann nicht mehr das Scaffolding des Frameworks verwenden, sondern selbst die Methoden innerhalb des Controllers schreiben.