

*Rapid Web Development mit dem  
MVC-Framework*

Deckt  
CakePHP 1.2 ab

*Webentwicklung mit*

# CakePHP



O'REILLY®

*Dirk Ammelburger  
& Robert Scherer*

---

# Inhalt

<b>Vorwort</b> .....	<b>IX</b>
<b>Einleitung</b> .....	<b>XI</b>
<b>1 CakePHP kennenlernen</b> .....	<b>1</b>
Was ist ein Framework? .....	2
Grundprinzipien des CakePHP-Frameworks .....	3
Das Model-View-Controller-Pattern .....	6
Das Zusammenspiel der CakePHP-Komponenten .....	9
CakePHP-Entwicklung Schritt für Schritt .....	12
CakeJobs – die Beispielanwendung .....	14
<b>2 Installation und Konfiguration</b> .....	<b>17</b>
Installation des Webservers .....	17
Installation von CakePHP .....	24
Installation der CakePHP-Console .....	28
<b>3 Schnell zum Erfolg – Ihre erste Webapplikation mit CakePHP</b> .....	<b>33</b>
Die Datenbank als Grundlage der Applikation .....	33
Die Anwendung starten und Gerüste bauen .....	36
<b>4 Der Controller</b> .....	<b>43</b>
Aufbau des Controllers .....	44
Attribute und Methoden im Controller .....	56
CakePHP-Konventionen .....	61
<b>5 Das Model</b> .....	<b>63</b>
Was ist ein Model? .....	63
Validierung im Model .....	65

Ein zweites Model einbinden .....	71
Model-Relationen herstellen .....	73
Retrieve: Daten finden .....	77
Create und Update: Daten speichern .....	84
Delete: Daten löschen .....	87
Model-Methoden und -Attribute .....	88
<b>6 Das View .....</b>	<b>95</b>
Was ist ein View? .....	95
Die Template-Engine in CakePHP .....	96
Nützliche Helfer im View: Die Helper-Klassen .....	111
Die Mini-Views: Elements .....	123
<b>7 Helper – Lassen Sie CakePHP für sich arbeiten .....</b>	<b>127</b>
Die Standard-Helper von CakePHP .....	128
Der HTML-Helper .....	129
Der Text-Helper .....	141
Der Time-Helper .....	145
Der Number-Helper .....	149
Der Cache-Helper .....	151
Der Paginator-Helper .....	156
Der JavaScript-Helper und CakePHP .....	166
Eigene Helper entwickeln .....	169
<b>8 Session-Handling mit CakePHP .....</b>	<b>173</b>
Session-Konfiguration .....	173
Die Session-Component verwenden .....	175
Der Session-Helper .....	178
Beispiel: Ein Merkzettel für Job-Anzeigen .....	182
Kekse zum Kuchen – Der Einsatz von Cookies .....	186
<b>9 Die CakePHP-Console und die Bake-Shell .....</b>	<b>193</b>
Shells starten .....	194
Die Console-Shell kennenlernen .....	195
Baking – Code automatisch generieren .....	197
Eigene Shells programmieren .....	210
<b>10 Der Controller reloaded – Callbacks, Routes &amp; Co. ....</b>	<b>217</b>
Die Controller-Funktionalität erweitern .....	217
Controller-Callbacks .....	219
URLs und Routes .....	220
Statische Seiten in CakePHP .....	225

<b>11</b>	<b>Components nutzen</b> .....	<b>227</b>
	Die Core-Components von CakePHP .....	228
	Die Email-Component .....	231
	Components selbst entwickeln .....	240
<b>12</b>	<b>Das Model reloaded – Bindings, Behaviors &amp; Co.</b> .....	<b>245</b>
	HABTM – hasAndBelongsToMany-Associations .....	245
	Model-Bindings .....	252
	Model-Callbacks .....	254
	Die AppModel-Klasse .....	257
	Models durch Behaviors erweitern .....	258
<b>13</b>	<b>Ajax und DHTML mit CakePHP</b> .....	<b>263</b>
	Prototype und script.aculo.us installieren .....	265
	Den Ajax-Helper einsetzen .....	266
	Paginierung mit Ajax .....	284
	JavaScript-Event-Handling .....	287
<b>14</b>	<b>Sicherheit, Authentifizierung und Autorisierung</b> .....	<b>293</b>
	Zugriffsschutz mit der Security-Component .....	293
	Authentifizierung mit der Auth-Component .....	300
	Autorisierung mit der Auth-Component .....	306
	ACL – Access Control Lists .....	310
<b>15</b>	<b>Lokalisierung und Internationalisierung</b> .....	<b>329</b>
	Grundlagen .....	329
	Mehrsprachigkeit einrichten .....	332
	Dynamische Daten internationalisieren .....	342
	Die Klasse i18n einsetzen .....	348
<b>16</b>	<b>Testing</b> .....	<b>353</b>
	Die Test-Suite installieren .....	355
	Tests schreiben .....	356
	Tests zusammenfassen .....	370
	Browser-Simulation mit Web Testing .....	371
<b>17</b>	<b>Weitere Kernfunktionalitäten von CakePHP</b> .....	<b>375</b>
	Plugins erstellen und einbinden .....	375
	Fehlerbehandlung in CakePHP .....	377
	Ein eigener ErrorHandler .....	380
	Debugging in CakePHP .....	383
	Die CakePHP-Core-Klassen .....	384

<b>18 Tipps und Tricks</b> .....	<b>395</b>
RSS-Feeds erstellen .....	395
Routes dynamisch aus der Datenbank generieren .....	397
Zugriff auf die Datenbank ohne Model .....	398
Die Auszeichnungssprache Flay nutzen .....	399
 <b>Index</b> .....	 <b>405</b>

---

# Der Controller

Nachdem Sie jetzt einen Einblick in das Rapid Development bzw. Rapid Prototyping mit CakePHP bekommen haben, interessiert es Sie sicher schon, was hinter der Fassade des Scaffolding passiert. Per Scaffolding können Sie Ihrem Kunden sehr schnell einen ersten Prototyp präsentieren, Sie sind aber sehr unflexibel, was Anpassungen am Standardverhalten angeht. In diesem Kapitel geht es darum, den Controller für die Jobverwaltung »von Hand« zu implementieren. Keine Sorge, dafür brauchen Sie nicht viel Code, und es geht auch mehr darum, Ihnen die eigentliche Aufgabe des Controllers zu erläutern.

Der Controller ist der Motor der Anwendung: Er wird anhand der angefragten URL aufgerufen und kümmert sich darum, dass die Daten unter Berücksichtigung der Parameter und Konditionen vom Model geholt werden. Er gibt diese dann an das View weiter, das dem Benutzer angezeigt wird.

In diesem Kapitel schauen wir uns zunächst den Aufbau eines Controllers an, um dann den *JobsControllers* für *CakeJobs* zu implementieren. Wir beginnen damit, uns Gedanken darüber zu machen, was der JobsController können und konkret machen soll. Dazu überlegen wir uns zunächst ganz untechnisch die Anwendungsfälle, die möglich sein sollen. Diese Anforderungen übersetzen wir dann in Controller-Actions, die wir Schritt für Schritt implementieren, wobei wir im gleichen Durchgang bereits die Templates rudimentär erstellen. Wenn das geschafft ist, steht die Verwaltung. Im darauffolgenden Abschnitt lernen Sie noch die wichtigsten Attribute und Methoden für die Arbeit im Controller genauer kennen.

Zum Abschluss des Kapitels gibt es noch einen Exkurs zu den Konventionen von CakePHP, in dem Sie lernen, wie Sie Dateien und Klassen benennen und platzieren – und platzieren müssen, damit die Automatismen von CakePHP greifen können.

## Aufbau des Controllers

Damit Sie einen Eindruck davon bekommen, wie ein Controller aufgebaut ist und wie einzelne Actions in einem Controller unter unterschiedlichen Bedingungen aufgerufen werden, setzen wir uns in diesem Kapitel das Ziel, das Beispiel der Jobdatenbank von den vorhergehenden Seiten nachzuprogrammieren. Dieses Mal werden Sie auf den »Trick« mit `$scaffold` verzichten und manuell die einzelnen Methoden implementieren. Grundlage dazu sind die beiden PHP-Dateien, die schon in jeweiligen Ordnern der Applikation angelegt sind:

- `App/controllers/jobs_controller.php`
- `App/models/job.php`

Die Model-Klasse werden wir in diesem Kapitel nicht weiter betrachten, da hier im Prinzip schon alle wesentlichen Aspekte berücksichtigt sind. Die Feinheiten des Models werden im Kapitel 4, *Der Controller*, genauer beleuchtet. Es bleibt also die Controller-Klasse, die bisher nur aus den folgenden Zeilen besteht:

```
class JobsController extends AppController {
    var $name = "Jobs";
    var $scaffold;
}
```

Die erste Schritt besteht darin, die Variable `$scaffold` wieder zu entfernen. Fehlt dieses Flag, wird CakePHP nicht mehr automatisiert eine CRUD-Oberfläche auf Basis des Models und der damit assoziierten Datenbanktabelle erstellen. Das werden wir nun manuell erledigen.

Als Vorüberlegung sollten Sie sich an dieser Stelle immer ein paar Gedanken über die folgenden Punkte machen:

- Was soll Ihre Applikation können?
- Welche Actions mit was für Parametern müssen dafür implementiert werden?
- Welche Daten sollen dem User angezeigt werden?

Auch wenn diese Gliederung noch recht grob ist, können Sie so die wesentlichen Fragen beantworten und die Struktur des Controllers festlegen.

## Was soll die Applikation können?

Die Frage ist recht einfach zu beantworten, besonders wenn wir uns noch einmal das Beispiel aus dem Kapitel 3, *Schnell zum Erfolg – Ihre erste Webapplikation mit CakePHP*, im Browser anschauen. Die Applikation soll auf einer einfachen Webseite Jobangebote aus einer Datenbanktabelle auslesen und diese in einem HTML-Interface darstellen. Darüber hinaus sollen die Angebote editiert und gelöscht werden können bzw. auch neu angelegt, sofern der aktuelle Arbeitsmarkt das erlaubt. Wir haben also eine Art Blackboard mit folgenden Funktionen:

- Übersicht aller Jobanzeigen
- Detailsicht einer Jobanzeige
- Editieren einer bestehenden Jobanzeige
- Anlegen einer neuen Jobanzeige
- Löschen einer Jobanzeige

Auch wenn die Anwendung in dieser Form vermutlich nicht wirklich produktiv eingesetzt wird, soll sie uns als Einstieg genügen. Später werden Sie die Applikation erweitern, so dass beispielsweise durch ein User-Management die Anzeigen nicht frei für jeden editierbar sind oder dass Anzeigen auch gezielt gesucht werden können.

## Welche Actions und Parameter werden benötigt?

Ein Controller enthält in der Regel eine Reihe von Methoden, die nach außen als Schnittstellen fungieren. Diese Schnittstellen, auch Actions genannt, können über die URL im Browser aufgerufen werden und lösen damit ein bestimmtes Verhalten im Controller aus. Die Verknüpfung zwischen einer bestimmten URL und einer Controller-Methode geschieht, wie sollte es anders sein, nach CakePHP-Konventionen, die sich im Aufbau der URL niederschlagen. Das URL-Schema ist dabei normalerweise so aufgebaut:

*`http://domain/controller/action/parameter1/parameter2/`*

Nach der Domain, die auf den Server mit Ihrer Anwendung zeigt, folgt ein Slash und dann der Name des Controllers. Dabei kommt wieder die Konvention ins Spiel, die in Kapitel 3, *Schnell zum Erfolg – Ihre erste Webapplikation mit CakePHP*, schon kurz angerissen wurde. Die `JobsController`-Klasse in der Datei `jobs_controller.php` wird in der URL mit `jobs` angesprochen. CakePHP weiß dann automatisch, welche Klasse gemeint ist und wo die nachfolgende Action zu finden ist.

Der Name der Action orientiert sich nach demselben Prinzip an dem Namen der Controller-Methode, die für die Abarbeitung dieses Requests zuständig ist. Die Methode `edit()` wird beispielsweise direkt durch die gleichnamige Action *edit* in der URL angesprochen.

Die nachfolgenden Parameter sind nur erforderlich, sofern die Action diese auch benötigt – was ja nicht immer so sein muss. Die `edit()`-Methode benötigt beispielsweise eine Information, *welchen* Datensatz sie zum Bearbeiten präsentieren soll – im Regelfall ist das die ID des Datensatzes:

*`http://localhost/cakejobs/jobs/edit/24`*

Diese URL stellt beispielsweise an den `JobsController` die Anfrage, die Methode `edit()` mit dem Parameter 24 aufzurufen. Der Zugriff auf so übergebene Parameter in den Actions ist denkbar einfach – sie werden den Methoden einfach als Argumente übergeben:

```
function edit($id = null) {
    echo $id; // ID 24
}
```

Entsprechend der Funktionsdefinitionen oben (im Abschnitt »Was soll die Applikation können?«) bietet es sich an, folgende Methoden für den Controller zu definieren:

Methode	Parameter	Aufgabe
index()	-	Übersicht über alle Jobangebote
view()	id	Ansicht eines bestimmten Job-Datensatzes
edit()	id	Editierfunktion für einen Job-Datensatz
add()	-	Einen neuen Job-Datensatz hinzufügen
delete()	id	Löschen eines Job-Datensatzes

## Das REST-Schema

Der Aufbau der URLs orientiert sich dabei an dem *Representational State Transfer*-Schema, kurz REST, das für den Aufbau verteilter Informationssysteme definiert wurde. Der Aufbau einer Anfrage gliedert sich demnach immer in die gleichen Bestandteile:

- Zieladresse der angefragten Datenquelle
- Schnittstelle für diese Anfrage auf dem entfernten System
- Methode der Anfrage
- Eventuell nötige Parameter für diese Anfrage

Dieser Aufbau lässt sich, wie oben beschrieben, in durch eine einfache, aber trotzdem universelle Syntax ausdrücken, die es erlaubt, beliebige Anfragen zwischen Systemen standardisiert zu definieren. CakePHP nutzt dieses Schema, um URLs abhängig vom Aufbau der Applikation definieren zu können und nicht, wie in der Regel üblich, von der Verzeichnisstruktur des Webservers abhängig zu sein. Möglich wird dies durch eine spezielle Konfiguration des Apache-Webservers, die über *htaccess*-Dateien im Framework definiert wird.

## Die Übersicht über alle Jobangebote

Nachdem Sie vorhin ja schon einige Datensätze per Scaffolding in unsere Datenbanktabelle geschrieben haben, können Sie jetzt mit der Auflistung der Jobangebote beginnen. Dafür legen Sie eine Methode im `JobsController` an, die Sie `index` nennen.

```
class JobsController extends AppController {
    var $name = 'Jobs';

    function index() {
```

```

        $jobs = $this->Job->find('all', array(
            'order' => 'created',
            'direction' => 'DESC'
        ));

        $this->set('jobs', $jobs);
    }
}

```

Was passiert hier? Sie lesen alle vorhandenen Jobs über die `find`-Methode des Job-Models in die Variable `$jobs` ein. Der `find`-Methode geben Sie per Parameter-Array die Anweisung, die Ergebnisse nach dem Erstellungsdatum absteigend zu sortieren. Dann wird die Variable mit den Ergebnissen mit der `set`-Methode dem View (das heißt dem Template) unter dem Namen `jobs` zugeordnet.

Jetzt möchten Sie natürlich wissen, was in `$jobs` für Daten enthalten sind. Dafür legen Sie erst einmal ein View für die `index`-Action an, indem Sie in `/app/views` den Ordner `jobs` anlegen und dort eine Datei namens `index.ctp` erstellen. Die Dateierweiterung `ctp` steht für *Cake Template* und wird für alle View-Templates verwendet.

Bevor Sie die Daten zur Aufbereitung anzeigen, geben Sie erst einmal die dem View zugewiesenen Daten aus. Schreiben Sie folgenden Code in die `index.ctp`:

```
<?php pr($jobs); ?>
```

Die Funktion `pr()` gehört zu den sogenannten *Convenience Methods* von CakePHP. `pr()` gibt die übergebene Variable mittels `print_r()` aus, wobei die Variable in `<pre>`-Tags gesetzt wird. Diese Funktion ist nützlich für Debug-Ausgaben, da sie nichts mehr ausgibt, wenn der `debug`-Mode auf 0 steht und Ihre Applikation im Produktiv-Modus ist. So laufen Sie nicht Gefahr, dass die Benutzer Ihre Debug-Ausgaben präsentiert bekommen.

Wie Sie sich sicher schon denken können, ist der erste Parameter der `set`-Methode dazu da, den Namen der zugewiesenen Daten im View zu bestimmen – diese sind dort dann immer als lokale Variable verfügbar. Nun können Sie im Browser die URL `http://localhost/jobs/index` aufrufen. Dort sollte Sie eine Ausgabe ähnlich der folgenden erwarten:

```

Array
(
    [0] => Array
        (
            [Job] => Array
                (
                    [id] => 1
                    [company] => Siemens AG
                    [title] => Elektrotechniker
                    [description] => Hier steht eine Job-Beschreibung.
                    [created] => 2008-01-19 16:43:12
                    [modified] => 2008-01-19 16:54:43
                )
        )
)

```

```

    )
    [1] => Array
    (
        [Job] => Array
        (
            [id] => 2
            [company] => Cake Foundation
            [title] => cakePHP-Entwickler
            [description] => Und hier steht noch eine Job-Beschreibung.
            [created] => 2008-01-19 16:43:38
            [modified] => 2008-01-19 16:43:38
        )
    )
)

```

Wie Sie sehen, hat das Job-Model die Daten fein säuberlich in ein Array geschrieben, das nur noch darauf wartet, in ansehnlicher Form ausgegeben zu werden. Dazu werden Sie die Daten einfach tabellarisch ausgeben und pro Datensatz noch den Zugriff auf die verschiedenen Actions gewähren, die mit dem Datensatz möglich sein sollen. Ersetzen Sie jetzt im `index-View` die Debug-Ausgabe durch dieses Template:

```

<h2>Jobs</h2>

<table>
  <tr>
    <th>ID</th>
    <th>Firma</th>
    <th>Titel</th>
    <th>Angebot vom</th>
    <th>zuletzt aktualisiert</th>
    <th>Actions</th>
  </tr>

  <?php foreach($jobs as $job): ?>
    <tr>
      <td><?php echo $job['Job']['id'] ?></td>
      <td><?php echo $job['Job']['company'] ?></td>
      <td><?php echo $job['Job']['title'] ?></td>
      <td><?php echo $job['Job']['created'] ?></td>
      <td><?php echo $job['Job']['modified'] ?></td>
      <td>
        <?php echo $html->link('Details', '/jobs/view/' . $job['Job']['id']) ?>
        <?php echo $html->link('bearbeiten', '/jobs/edit/' . $job['Job']['id']) ?>
        <?php echo $html->link('löschen', '/jobs/delete/' . $job['Job']['id']) ?>
      </td>
    </tr>
  <?php endforeach; ?>
</table>

```

Wie Sie sehen, verwenden Sie hier die `link()`-Methode des HTML-Helper, um die HTML-Links zu den Actions auszugeben. Wenn Sie sich jetzt fragen, warum wir die Links nicht einfach als HTML in das Template schreiben: Die Verwendung der View-Helper in CakePHP hat viele Vorteile, auf die wir in Kapitel 6, *Das View*, in aller Ausführlichkeit eingehen werden.

Wenn Sie jetzt `http://localhost/cakejobs/jobs/index` aufrufen, sollten Sie im Browser die Ansicht geliefert bekommen, die Sie in Abbildung 4-1 sehen.

**CakePHP: the rapid development php framework**

**Jobs**

ID	Firma	Titel	Angebot vom	zuletzt aktualisiert	Aktionen
1	Siemens AG	Elektrotechniker	2008-01-19 16:43:12	2008-01-19 16:54:43	<a href="#">Details</a> <a href="#">bearbeiten</a> <a href="#">löschen</a>
2	Cake Foundation	cakePHP-Entwickler	2008-01-19 16:43:38	2008-01-19 16:43:38	<a href="#">Details</a> <a href="#">bearbeiten</a> <a href="#">löschen</a>
3	Volkswagen	Kundenberater	2008-01-19 16:57:18	2008-01-19 16:57:18	<a href="#">Details</a> <a href="#">bearbeiten</a> <a href="#">löschen</a>

Nr	Query	Error	Affected	Num. rows	Took (ms)
1	SELECT `Job`.`id`, `Job`.`company`, `Job`.`title`, `Job`.`description`, `Job`.`created`, `Job`.`modified` FROM `jobs` AS `Job` WHERE 1 = 1 ORDER BY `created` ASC		3	3	0

Abbildung 4-1: Die `index()`-Ansicht

Die Übersicht der Jobangebote ist nun erst einmal abgeschlossen. Weiter geht es mit der Detailansicht für die Jobangebote.

## Detailansicht der Jobangebote

Sagen wir, Sie interessieren sich für die Stelle als CakePHP-Entwickler bei der Cake Software Foundation und möchten dazu die Stellenbeschreibung lesen, die ja für Ihre Jobwahl nicht ganz unwesentlich ist. Wenn Sie jetzt aber auf *Details* klicken, werden Sie eine Ansicht wie in Abbildung 4-2, präsentiert bekommen, die für Ihre Jobwahl vermutlich nicht hilfreich ist.

Diese CakePHP-Fehlermeldung sagt uns sehr direkt, was wir machen sollen: die View-Action im JobsController implementieren. Sogar der benötigte Code wird Ihnen vorgegeben – was will man mehr? Hier brauchen Sie mehr als eine leere Methode – aber nicht viel mehr.

```
function view($id) {
    $job = $this->Job->read(null, $id);
    $this->set('job', $job);
}
```



Abbildung 4-2: Die `view()`-Methode wurde im `JobsController` nicht gefunden.

Was passiert an dieser Stelle? Sehen Sie sich erst einmal die aufgerufene URL an: <http://localhost/cakejobs/jobs/view/2>. Alle URL-Parameter, die nach dem Namen der Action `view` kommen, werden der Methode einfach als Funktionsargumente übergeben – ohne dass Sie mehr dafür machen müssen, als den Namen des Parameters in der Methodensignatur anzugeben. Mit der Model-Methode `read()` holen Sie sich hier den Datensatz mit der übergebenen ID aus der Datenbank und weisen ihn, wie Sie es schon in der `index()`-Methode gemacht haben, dem View zu.

Die Struktur des Arrays, das Ihnen vom Model übergeben wird, ist dieselbe wie die der `index()`-Methode, aber mit nur einem einzigen Datensatz. Erstellen Sie nun ein Template namens `view.ctp` im `View`-Ordner des `JobsControllers`, und lassen Sie sich wieder die übergebene Template-Variable ausgeben:

Inhalt der Datei:

```
<?php pr($job); ?>
```

Ausgabe:

```
Array
(
    [Job] => Array
        (
            [id] => 2
            [company] => Cake Foundation
            [title] => cakePHP-Entwickler
            [description] => Hier steht eine Job-Beschreibung.
            [created] => 2008-01-19 16:43:38
            [modified] => 2008-01-19 19:22:29
```

Mit diesem Array als Grundlage können Sie jetzt sehr einfach eine entsprechende Ausgabe erstellen:

```

<h2>Job-Details</h2>
<dl>
  <dt>ID</dt>
  <dd><?php echo $job['Job']['id']; ?></dd>

  <dt>Firma</dt>
  <dd><?php echo $job['Job']['company']; ?></dd>

  <dt>Titel</dt>
  <dd><?php echo $job['Job']['title']; ?></dd>

  <dt>Beschreibung</dt>
  <dd><?php echo $job['Job']['description']; ?></dd>

  <dt>Angebot vom</dt>
  <dd><?php echo $job['Job']['created']; ?></dd>

  <dt>zuletzt aktualisiert</dt>
  <dd><?php echo $job['Job']['modified']; ?></dd>
</dl>
    
```

Die Ansicht im Browser sollte nun anstatt einer Fehlermeldung die Detail-Ansicht des gewünschten Jobangebots präsentieren, die Sie in Abbildung 4-3 sehen.

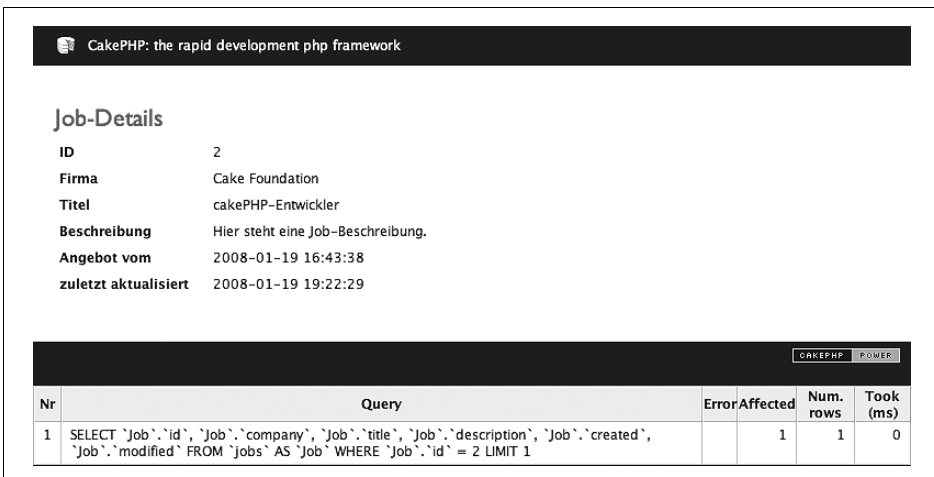


Abbildung 4-3: Die view()-Ansicht

Jetzt haben Ihre Benutzer die Möglichkeit, sich in aller Ruhe zu überlegen, ob sie ihre Bewerbungsunterlagen losschicken sollen oder nicht. Ein Blindtext als

Beschreibung ist hier aber doch nicht sinnvoll – jetzt benötigen Sie die Möglichkeit, das Jobangebot zu bearbeiten und eine aussagekräftige Stellenbeschreibung einzugeben.

## Bearbeiten eines Jobangebots

Bisher haben Sie ohne Hilfe des *Scaffoldings* nur Daten ausgegeben – jetzt ist es an der Zeit, auch Daten zu speichern, und zwar mittels eines Eingabeformulars und der `edit()`-Methode, die folgendermaßen aussieht:

```
function edit($id = null) {
    if(!empty($this->data)) {

        if($this->Job->save($this->data) {
            $this->Session->setFlash('Der Job wurde gespeichert.');
```

```
            $this->redirect(array('action'=>'index'));
        } else {
            $this->Session->setFlash('Der Job konnte nicht gespeichert werden.');
```

```
        }

    } else {
        $this->data = $this->Job->read(null, $id);
    }
}
```

Beim Bearbeiten eines Job-Angebots gibt es drei mögliche Fälle:

- Der Benutzer hat die Ansicht gerade erst geöffnet und das Formular noch nicht abgesendet.
- Der Benutzer hat die gewünschten Eingaben gemacht und das Formular abgeschickt, es konnte aber aus irgendeinem Grund nicht gespeichert werden, und der Benutzer landet wieder auf derselben Ansicht mit seinen eingegebenen Daten in den Feldern und eventuellen Fehlermeldungen.
- Der Benutzer hat das Formular abgeschickt, und es wurde gespeichert. Der Benutzer wird dann mit einer Meldung zur Listenansicht weitergeleitet.

Diese Fälle decken Sie mit mehreren Bedingungen ab. Wenn `$this->data`, der temporäre Datenspeicher für POST-Daten aus Formularen, bereits Daten enthält, wird versucht, diese Daten mittels der `save()`-Methode des Job-Models abzuspeichern.

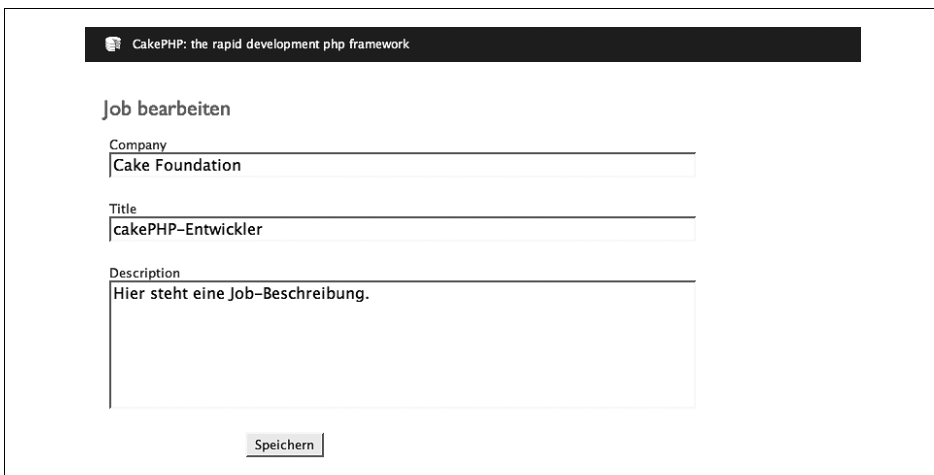
Wenn das gelingt, wird eine Meldung in die Session geschrieben, und der Benutzer wird mit der `redirect()`-Methode zur `index()`-Action des JobsControllers weitergeleitet, wo diese Meldung dann angezeigt wird. `$this->Session->setFlash()` in Kombination mit `$this->redirect()` im Controller einer Applikation ist eine einfache Möglichkeit, um solche Abläufe zu steuern – mehr dazu erfahren Sie in Abschnitt »Den Fluss steuern – `redirect()` und `flash()`« weiter unten in diesem Kapitel.

Wenn der Speichervorgang aus irgendeinem Grund misslingt (das kann beispielsweise an einer gescheiterten Validierung der Daten liegen) wird eine Fehlermeldung in die Session geschrieben, und der User bekommt wieder die *Bearbeiten*-Ansicht mit der Meldung angezeigt. In diesem Fall werden seine eingegebenen Daten wieder in den Formularfeldern angezeigt. Oder aber der erste unserer Fälle tritt ein – der Benutzer hat die Ansicht gerade erst geöffnet und hat noch keine Daten eingegeben. In diesem Fall weisen Sie dem Datenspeicher `$this->data` mit der `read()`-Methode, wie wir es schon aus der `view()`-Action kennen, die Daten des Datensatzes mit der übergebenen Action zu.

Nun aber zum spannenderen Teil: Wie erstellen Sie ein Webformular, das die Daten bereits in den Feldern anzeigt? Dies ist eine Aufgabe, die Sie sicher schon viel zu oft wiederholen mussten. Erstellen Sie eine Datei namens `edit.ctp` in `/app/views/jobs`, und schließen Sie mit folgendem Code das `edit()`-View ab:

```
<h2>Job bearbeiten</h2>
<?php
    echo $form->create('Job');
    echo $form->input('id');
    echo $form->input('company');
    echo $form->input('title');
    echo $form->input('description');
    echo $form->end('Speichern');
?>
```

Wie Sie sehen, verwenden Sie für das Formular keine einzige Zeile HTML, vom Titel einmal abgesehen. Wenn Sie sich jetzt fragen, warum Sie PHP-Code zur Erstellung von HTML-Quelltext verwenden sollten, sehen Sie sich einfach das Resultat im Browser an (Abbildung 4-4).



The screenshot shows a web browser window displaying a form titled "Job bearbeiten". At the top, there is a black banner with the text "CakePHP: the rapid development php framework". Below the banner, the form has three input fields: "Company" with the value "Cake Foundation", "Title" with the value "cakePHP-Entwickler", and "Description" with the value "Hier steht eine Job-Beschreibung.". At the bottom of the form, there is a button labeled "Speichern".

Abbildung 4-4: Die `edit()`-Ansicht

CakePHP hat Ihnen mithilfe des Form-Helpers die ganze Arbeit, so ein Formular zu erstellen, abgenommen. CakePHP kennt die Datentypen der Tabellen, mit denen Sie arbeiten, und erstellt dann beispielsweise für ein TEXT-Feld auch eine `<textarea>`. Wie Ihnen vielleicht auffällt, sehen Sie kein Feld für die ID, obwohl Sie auch ein Feld dafür definiert haben. Eine ID soll ja schon per Definition nicht geändert werden, deshalb gibt CakePHP dafür auch ein `input`-Feld mit dem Typ `hidden` aus. Wenn Sie nun das Formular abspeichern, sollten Sie wieder bei der Auflistung der Jobangebote landen – mit einer Meldung, dass die Daten gespeichert wurden (Abbildung 4-5).

The screenshot shows a web browser window with the title 'CakePHP: the rapid development php framework'. Below the title, a message reads 'Der Job wurde gespeichert.' (The job was saved). Underneath, there is a section titled 'Jobs' containing a table with three columns: 'ID', 'Firma', 'Titel', 'Angebot vom', 'zuletzt aktualisiert', and 'Aktionen'. The table lists three job offers from Siemens AG, Cake Foundation, and Volkswagen.

ID	Firma	Titel	Angebot vom	zuletzt aktualisiert	Aktionen
1	Siemens AG	Elektrotechniker	2008-01-19 16:43:12	2008-01-19 16:54:43	<a href="#">Details</a> <a href="#">bearbeiten</a> <a href="#">löschen</a>
2	Cake Foundation	cakePHP-Entwickler	2008-01-19 16:43:38	2008-01-20 11:56:08	<a href="#">Details</a> <a href="#">bearbeiten</a> <a href="#">löschen</a>
3	Volkswagen	Kundenberater	2008-01-19 16:57:18	2008-01-19 16:57:18	<a href="#">Details</a> <a href="#">bearbeiten</a> <a href="#">löschen</a>

Abbildung 4-5: Die Auflistung der Jobangebote nach Abspeichern des Formulars

## Anlegen eines Jobangebots

Nachdem Sie die Bearbeitungsfunktion für die Jobangebote gemeistert haben, ist die Funktionalität zum Anlegen eines Angebots schnell erledigt. Der Code für den Controller nutzt die `add()`-Methode, er ist dem der `edit()`-Methode sehr ähnlich:

```
function add() {
    if (!empty($this->data)) {
        $this->Job->create();
        if ($this->Job->save($this->data)) {
            $this->Session->setFlash('Der Job wurde gespeichert.');
```

Die Logik ist dieselbe, mit dem Unterschied, dass beim Anlegen von Jobangeboten in keinem Fall ein Jobangebot per `read()` aus der Datenbank gelesen werden muss. Vor dem Speichern wird das Job-Model mit `create()` noch darauf vorbereitet, dass ein neuer Datensatz angelegt wird. Das Formular für das View ist fast identisch mit dem der `edit()`-Methode – mit dem Unterschied, dass Sie beim Neuanlegen dieser `add()`-View kein Feld mit einer ID-Information benötigen.

```

<h2>Neuer Job</h2>
<?php
    echo $form->create('Job');
    echo $form->input('company');
    echo $form->input('title');
    echo $form->input('description');
    echo $form->end('Speichern');
?>

```

Jetzt können Sie unter *http://localhost/jobs/add* nach Belieben neue Jobangebote anlegen.

## Jobangebote löschen

Bereits besetzte Stellen sollten Sie aus dem Job-Katalog löschen können. Das Problem lösen wir mit der `delete()`-Action im `JobsController`. Die Methode dazu sieht folgendermaßen aus:

```

function delete($id = null) {
    if (!$id) {
        $this->Session->setFlash('Ungültige Job-ID');
        $this->redirect(array('action'=>'index'));
    }
    if ($this->Job->del($id) {
        $this->Session->setFlash("Der Job mit der ID {$id} wurde gelöscht.");
        $this->redirect(array('action'=>'index'));
    }
}

```

Erst einmal kontrollieren Sie, ob auch wirklich eine ID übergeben wurde. Falls keine übergeben wurde, benutzen Sie wieder `setFlash()` und `redirect()`, um den Benutzer mit einer Fehlermeldung zurück zur Auflistung zu leiten. Andernfalls löschen Sie den Job mittels der `del()`-Methode des Job-Models und geben dem Benutzer über der Auflistung den Hinweis aus, dass die gewünschte Aktion abgeschlossen wurde.

Vielleicht fragen Sie sich jetzt, ob jede Methode ein View haben muss – die Antwort ist Nein. Wenn Sie beim Löschen eines Datensatzes keine weiteren Sicherheitsabfragen machen möchten (vielleicht abgesehen von einer JavaScript-basierten Bestätigungsbox), macht eine View für die `delete()`-Methode keinen Sinn. Die Views werden von CakePHP automatisch nach dem Ende der entsprechenden Methode generiert (»gerendert«). Mit der `redirect()`-Methode wird die Action standardmäßig aber abgebrochen. Das heißt, die `delete()`-Methode wird gar nicht bis zum Schluss ausgeführt und das Rendering benötigt kein View dafür.

Wenn Sie jetzt ein Jobangebot aus der Auflistung löschen, sollten Sie direkt wieder bei der Auflistung landen und mit einer Meldung informiert werden, dass das Jobangebot gelöscht wurde (Abbildung 4-6).

CakePHP: the rapid development php framework

**Der Job mit der ID 5 wurde gelöscht.**

**Jobs**

ID	Firma	Titel	Angebot vom	zuletzt aktualisiert	Aktionen
1	Siemens AG	Elektrotechniker	2008-01-19 16:43:12	2008-01-19 16:54:43	<a href="#">Details</a> <a href="#">bearbeiten</a> <a href="#">löschen</a>
2	Cake Foundation	cakePHP-Entwickler	2008-01-19 16:43:38	2008-01-20 11:56:08	<a href="#">Details</a> <a href="#">bearbeiten</a> <a href="#">löschen</a>
3	Volkswagen	Kundenberater	2008-01-19 16:57:18	2008-01-19 16:57:18	<a href="#">Details</a> <a href="#">bearbeiten</a> <a href="#">löschen</a>

Abbildung 4-6: Die Meldung nach dem Löschen eines Jobangebots

So, geschafft: Sie haben eine komplette CRUD-Funktionalität für Jobangebote selbst programmiert, und das mit sehr wenig Code.

## Attribute und Methoden im Controller

Im Controller stehen Ihnen eine Reihe von Methoden und auch Attributen zur Verfügung, die Sie zum Teil schon im Beispiel oben kennengelernt haben. Die wichtigsten Attribute und Methoden sollen an dieser Stelle noch einmal zusammengefasst und erklärt werden. Darüber hinaus werden wir noch auf einige Punkte eingehen, die bisher noch keinen Platz in diesem Kapitel gefunden haben.

### Controller-Attribute

In diesem Abschnitt finden Sie ausführlichere Erläuterungen zu den Standardattributen, die im Controller verwendet werden.

#### Components einbinden – \$uses, \$helpers und \$components

Diese drei Attribute sind optional in der Controller-Klasse und sind bisher in unserem Beispiel auch noch nicht vorgekommen. Sie sind dafür zuständig, dem Controller die benötigten Components mitzuteilen. Alle drei Attribute sind Arrays, denen Sie die Namen der gewünschten Klassen übergeben. CakePHP wird diese dann laden, instanziiieren und an der richtigen Stelle zur Verfügung stellen.

Mit der Variable \$uses teilen Sie dem Controller mit, welche Models Sie benutzen möchten. Wenn Sie sich an die CakePHP-Konventionen halten, steht im *JobsController*, wie Sie es ja schon gesehen haben, automatisch das *Job*-Model zur Verfügung. Sie sind an diese Konventionen aber nicht gebunden. Sie könnten beispielsweise auch statt des *Job*-Models ein anderes Model laden oder zusätzliche Models mit einbinden. Dazu genügt ein einfaches Array.

```
$uses = array('Job', 'AnderesModel');
```

CakePHP lädt so automatisch die gewünschten Klassen aus dem *models*-Ordner, instanziiert sie und stellt sie im Controller so zur Verfügung:

```
$this->AnderesModel
```

Ähnlich verhält es sich mit den Attributen `$helpers` und `$components`. Sie können die Funktionalität eines Controllers und der darin verwendeten Views erweitern, indem Sie entsprechende Helper (für das View) oder Components (für den Controller) einbinden.

### Seitentitel setzen – `$pageTitle`

Das Attribut `$pageTitle` erlaubt es, dynamisch aus dem Controller heraus den Titel der Seite zu setzen. Wenn Sie die Beispielapplikation in einer der Methoden im Controller um die folgenden Zeile Code erweitern, wird der Seitentitel im Browser entsprechend geändert:

```
$this->pageTitle = 'Das ist ein neuer Titel';
```

Dieser Effekt basiert auf einer dynamischen Zuweisung im Template, die über das verwendete Layout von CakePHP gesteuert wird. Wie das genau geschieht, werden Sie in Kapitel 6, *Das View*, erfahren.

### Zugriff auf die Parameter der Anfrage – `$params`

Das Array `$params` wird automatisch zur Laufzeit der Applikation im Controller durch CakePHP gesetzt, ohne dass Sie etwas dazu tun müssen. In diesem Array stellt Ihnen CakePHP eine Reihe von Informationen zur Verfügung, die Auskunft über den aktuellen Request geben. Erweitern Sie das Beispiel einfach um den folgenden Befehl in einer der Action-Methoden, damit das Array ausgegeben wird. So gewinnen Sie am besten einen Eindruck vom Aufbau der Daten.

```
pr($this->params);
```

Wenn Sie das `$params`-Array nun beispielsweise im View der `add()`-Action im Jobs-Controller ausgeben, bekommen Sie folgendes Array:

```
Array
(
    [pass] => Array
        (
            [0] => 2
        )

    [named] => Array ()

    [controller] => jobs
    [action] => view
    [plugin] =>
    [form] => Array ()
)
```

```

        [url] => Array
        (
            [url] => jobs/view/2
        )

        [bare] => 0
        [webservices] =>
    )

```

Die Daten sind zum großen Teil selbsterklärend. Trotzdem sollen die einzelnen Punkte hier kurz erklärt werden.

*\$this->params['form']*

Dieses Array speichert Formulardaten, die nicht mit einem Model assoziiert werden sollen, inklusive Daten aus *FILE*-Auswahlfeldern, die normalerweise im nativen Array `$_FILES` verfügbar sind.

*\$this->params['bare']*

Ein Flag zur Bestimmung des Layouts. Wenn ein Layout verwendet wurde, steht dieses Flag auf 0 (false), ansonsten auf 1 (true).

*\$this->params['controller']*

Speichert den Namen des aktuellen Controllers. In unserem Fall also *jobs*.

*\$this->params['action']*

Der Name der Action, die in der URL aufgerufen wurden. In unserem Fall also *view*.

*\$this->params['pass']*

Dieses Array speichert die im CakePHP-Stil übergebenen Parameter in der übergebenen Reihenfolge. Hier würde beispielsweise der in der *view()*-Action verwendete ID-Parameter an erster Stelle stehen (siehe obiges Beispiel).

*\$this->params['url']*

In diesem Array speichert CakePHP die klassisch per GET übergebenen Variablen. Hier steht immer der Schlüssel URL, da CakePHP intern die URL zu *controller/action/parameter* der *index.php* als normalen URL-Parameter übergibt.

*http://localhost/jobs/view/2* würde in *http://localhost/index.php?url=/jobs/view/2* resultieren.

Wenn Sie jetzt der URL noch zusätzlich GET-Parameter im »normalen« Stil übergeben würden, würden diese in `$this->params['url']` aufgelistet werden.

## Auf Daten zugreifen – `$data`

Das Controller-Attribut `$data` haben wir in den vorangegangenen Beispielen schon mehrfach verwendet, beispielsweise um auf die Daten zuzugreifen, die aus einem abgeschickten Formular kommen. Immer wenn eine Methode im Controller durch einen POST-Request aufgerufen wird, enthält dieses Array alle Daten des aufrufenen Formulars im View. Das Array ist immer in Controller, View und Model verfü-

bar. Wenn die Daten durch den Form-Helper übergeben wurden, sind diese entsprechend dem dazugehörigen Model aufbereitet.

## Controller-Methoden

Neben den verschiedenen Attributen gibt es auch eine Reihe von Standardmethoden, die im Controller verwendet werden.

### Dem View Daten übergeben – set()

Auch die Methode set() haben Sie bereits im Beispiel kennengelernt. Mit ihr werden Daten vom Controller an das View übergeben und einem bestimmten Bezeichner zugeordnet. Dabei spielt es keine Rolle, um welche Art von Daten es sich handelt – sei es ein primitiver Datentyp, ein Array oder ein Objekt, mit set() wird die Variable direkt im View als lokal zugängliche Variable verfügbar gemacht.

Die mit set() gesetzten Variablen sind grundsätzlich im gesamten View-Objekt verfügbar. Das heißt, über set() zugewiesene View-Variablen sind auch im Layout verfügbar und nicht nur im Template der jeweiligen Action. Dieses Verhalten ist beispielsweise dann nützlich, wenn Sie aus dem Controller heraus eine Überschrift, die außerhalb des Action-Templates liegt, dynamisch setzen möchten.

### Der Action ein Ende machen – render()

Im normalen Ablauf einer Anfrage an das CakePHP-Framework wird ein großer Teil der Schritte automatisch im Hintergrund erledigt. In den meisten Fällen bekommen der User und auch Sie als Entwickler wenig davon mit, was CakePHP für Sie erledigt. Den Prozess und den Ablauf einer solchen Anfrage haben wir bereits am Anfang dieses Kapitels genau besprochen.

Es gibt allerdings Situationen, in denen es erforderlich ist, in den Ablauf einzugreifen und mehr Kontrolle über ihn zu bekommen. Mit der Funktion render() können Sie das Rendern des Views aus dem Controller heraus direkt starten, ohne das Ende der Methode »abzuwarten«. Das ist beispielsweise dann sinnvoll, wenn Sie anstelle des per Konvention dem Controller zugehörigen Views ein anderes Template oder auch ein anderes Layout als das standardmäßige verwenden wollen.

### Den Fluss steuern – redirect() und flash()

Mit der Methode redirect() können Sie den Fluss der Anwendung unterbrechen und auf eine andere URL weiterleiten. Sie werden diese Methode vermutlich recht häufig brauchen, da Weiterleitungen in einer Applikation ein gern verwendetes Mittel sind, um den User zu steuern. Die Funktion erwartet dafür drei Parameter, wobei die letzten beiden optional sind:

```
Controller::redirect ( $url, $status = null, $exit = true )
```

Der Parameter `$url` kann ein String oder ein Array sein. Im ersten Fall kann es eine beliebige relative oder absolute URL sein, auf die CakePHP dann weiterleitet. Wenn ein Array übergeben wird, dann muss es im folgenden Format sein, wie Sie es beispielsweise auch schon in der `edit()`-Action verwendet haben:

```
array('controller' => 'jobs', 'action' => 'view', 'param1')
```

In diesem Array werden der Controller und die dazugehörige Action über assoziative Schlüssel definiert. Danach werden die eventuell nötigen Parameter als weitere Array-Elemente übergeben.

Der zweite Parameter ermöglicht es Ihnen, einen HTTP-Status-Code festzulegen, der an den Browser während der Weiterleitung gesendet wird, beispielsweise 301 für *moved permanently*. Der dritte Parameter ist standardmäßig auf `true` gesetzt und gibt an, ob das Skript nach der Weiterleitung gestoppt werden soll oder nicht. Falls Sie aus irgendwelchen Gründen das Skript serverseitig bis zum Ende der Methode laufen lassen wollen, können Sie diesen Parameter auf `false` setzen.

Die Methode `flash()` funktioniert im Prinzip ähnlich wie `redirect()`, mit dem Unterschied, dass vor der Weiterleitung eine Nachrichtenseite geladen wird, die dem User eine Nachricht ausgibt. Danach wird auf die definierte URL weitergeleitet. Der Aufbau der Funktion ist wie folgt:

```
Controller::flash ( $message, $url, $pause = 1 )
```

Der erste Parameter übergibt die auszugebende Nachricht an das Framework. Der zweite Parameter ist die Redirect-URL, auf die nach der Flash-Nachricht weitergeleitet werden soll. Hierbei gelten dieselben Regeln wie für die Methode `redirect()`. Als Letztes folgt der optionale Parameter `$pause`, der schlicht festlegt, wie lange die Flash-Nachricht eingeblendet werden soll.

Bitte beachten Sie, dass `flash()` im Debug-Modus von CakePHP nicht automatisch weiterleitet, sondern stehen bleibt. Der automatische Redirect funktioniert nur, wenn CakePHP produktiv läuft.

Die letzte Funktion, die für Sie noch hilfreich sein wird, ist `referer()`. Diese Methode gibt einfach die URL der vorhergehenden Seite zurück, von der auf diese Seite verlinkt wurde. Mit den zwei optionalen Parametern kann das Verhalten der Funktion gesteuert werden:

```
Controller::referer ( $default = null, $local = false )
```

Der erste Parameter übernimmt eine Standard-URL, die zurückgegeben wird, falls CakePHP den tatsächlichen Referer nicht auslesen kann. Wenn Sie den zweiten Parameter, `$local`, auf `true` setzen, gibt die Methode nur dann Referer-URLs aus, sofern sich die URL auf demselben Server befindet.

# CakePHP-Konventionen

Konventionen in CakePHP sind einer der wichtigsten Punkte in diesem Framework, denn sie erlauben es, ohne große Konfigurationsdateien eine Applikation zu erstellen. Sie sind quasi das Salz in der Suppe, das Schmieröl im Getriebe oder – um es mit den Worten des CakePHP-Manuals auszudrücken – *Conventions in Cake are what make the magic happen, read it automatic*. Nicht umsonst sind die Konventionen in den Grundsätzen des Frameworks durch die Aussage *Convention over Configuration* fest definiert worden.

CakePHP zwingt Sie nicht dazu, die vorgeschlagenen Konventionen einzuhalten. Sie dürfen alle Dateien, alle Methoden und alle Klassen benennen, wie Sie möchten. Allerdings werden Sie dann nicht um eigentlich unnötigen Konfigurationsaufwand herumkommen, weil Sie dem Framework jede Möglichkeit nehmen, automatisch Verknüpfungen zwischen den Bestandteilen der Applikation herzustellen.

Wenn Sie sich an Konventionen halten, werden Sie feststellen, dass es erstaunlich schnell möglich ist, eine Applikation zu schreiben, ohne dabei Einfluss oder Flexibilität zu einzubüßen. Auch wenn die folgenden Ansätze auf den ersten Blick vielleicht gewöhnungsbedürftig scheinen, CakePHP geht einen sehr einfachen und auch intuitiven Weg, der schnell zu adaptieren ist.

In diesem und auch im vorherigen Kapitel haben wir an den verschiedenen Stellen schon auf die Möglichkeiten der Konventionen für dieses Framework hingewiesen. Auch wenn im Wesentlichen die Syntax und die darauf basierenden Annahmen von CakePHP klar sind, möchte ich an dieser Stelle noch einmal alle Konventionen kurz zusammenfassen, damit sie an einer Stelle übersichtlich verfügbar sind.

CakePHP verwendet die zwei in der Programmierung gängigsten Schreibweisen: auf der einen Seite *underscoped*, also eine Schreibweise mit Unterstrichen, und auf der anderen Seite *CamelCased*, also ohne Unterstriche und mit Großschreibung am Beginn eines jeden Wortes. Im Folgenden wird die Anwendung der Konventionen im jeweiligen Kontext kurz beschrieben.

## Dateinamen

Dateinamen werden immer kleingeschrieben, und mehrere Wörter in Dateinamen werden immer durch Unterstriche getrennt. Eine Klasse mit dem Namen `MyClass` wird immer in einer Datei mit dem Namen `my_class.php` liegen.

## Model-Konventionen

Model-Namen beziehen sich standardmäßig immer auf die zugehörige Datenbanktabelle, wobei das Model immer im Singular und die Datenbanktabelle immer im Plural geschrieben wird. Darüber hinaus wird das Model groß und CamelCased

geschrieben, während die Tabelle mit Kleinbuchstaben und mit Unterstrichen bezeichnet werden muss.

Die Tabelle *jobs* hat also ein Model mit dem Namen *Job*. Ein Model mit der Bezeichnung *JobPublisher* greift automatisch auf eine Datenbanktabelle mit dem Namen *job\_publishers* zu.

## Controller-Konventionen

Der Name des Controllers wird immer im Plural notiert, wobei der Klassenname CamelCased und großgeschrieben ist und immer um das Suffix *Controller* ergänzt werden muss. Die Controller-Klasse für das Model *JobPublisher* würde also *JobPublishersController* heißen. Der Dateiname für diese Klasse würde nach den Konventionen der Dateinamen *job\_publishers\_controller.php* lauten.

Der Aufruf eines Controllers in der URL geschieht nach identischen Regeln: Der *JobPublishersController* wird über die *Underscore*-Schreibweise angesprochen, allerdings ohne das Suffix *controller* und ist so eindeutig definiert:

*http://your\_domain/job\_publishers/my\_action*

Das Gleiche gilt für den Aufruf einer Action im Controller, die durch eine Methode festgelegt ist. Jeder Methodenname muss mit Unterstrichen im Controller definiert werden, damit er direkt aus der URL über das Framework angesprochen werden kann. Die Beispiel-URL oben zeigt den Aufruf der Methode *my\_action()* in der Controller-Klasse *JobPublishersController*.

Den Controller haben Sie jetzt kennengelernt. Sie haben CRUD für Ihre Jobangebote komplett implementiert, mit allem, was dazugehört. Im nächsten Kapitel sehen wir uns das Model genauer an. Dieses haben Sie in diesem Kapitel bereits eingesetzt. Es kann aber noch viel mehr als in den hier gezeigten Einsätzen.