open source library

Ralf Eggert

Das Zend Framework

Von den Grundlagen bis zur fertigen Anwendung

ADDISON-WESLEY

CD

3 Schnellstart mit dem Zend Framework

Nach der Einführung und Informationen zur Einrichtung und Installation des Zend Framework möchte ich Ihnen nun die Gelegenheit geben, direkt einzusteigen. Wir erstellen in diesem Kapitel eine erste kleine Anwendung auf Basis des Model-View-Controller-Entwurfsmusters. Wenn Sie sich noch nicht genau mit der MVC-Architektur auskennen, werfen Sie einen Blick in Anhang B.1, Model-View-Controller.

Wenn Sie nicht alles in dieser Schnellstart-Anleitung auf Anhieb verstehen, ist das nicht schlimm. Sie sollen zum Einstieg zumindest einmal eine Zend Framework-Anwendung gesehen haben. Bei offenen Fragen können Sie jederzeit zu den als Verweis genannten Kapiteln blättern. Als Ergänzung zu diesem Kapitel lohnt sich auch ein Blick in die offizielle Schnellstart-Anleitung für das Zend Framework [03-001].

Auf der CD

Das vollständige Projekt für diese Schnellstart-Anleitung finden Sie auf der CD-ROM im Verzeichnis `kapitel03/schnellstart/`. Alle Listings aus diesem Kapitel finden Sie ebenfalls auf der CD-ROM im Verzeichnis `kapitel03/`. Das erspart Ihnen einiges an Tipparbeit, wenn Sie der Schnellstart-Anleitung Schritt für Schritt folgen.

3.1 Projekt einrichten

Um das Projekt einrichten zu können, legen Sie ein neues Projektverzeichnis an (siehe auch Kapitel 2.3.1, Projektverzeichnis erstellen). Nennen Sie es `schnellstart/`, dann sind die folgenden Ausführungen für Sie einfacher nachzuvollziehen. Kopieren Sie nun die Projektvorlage `verzeichnisstruktur.zip` aus dem Verzeichnis `kapitel02/` von der CD-ROM in Ihr neues Projektverzeichnis `schnellstart/`.

Entpacken Sie die Datei `verzeichnisstruktur.zip` und löschen sie wieder. Prüfen Sie bitte, ob Ihr Entpackungsprogramm im Verzeichnis `schnellstart/` das Verzeichnis `verzeichnisstruktur/` angelegt hat. Falls nicht, fahren Sie mit dem nächsten Absatz fort. Sollte das Verzeichnis jedoch vorhanden sein, verschieben Sie alle Inhalte aus `schnellstart/verzeichnisstruktur/` nach `schnellstart/`, und löschen Sie das Verzeichnis `schnellstart/verzeichnisstruktur/` wieder. Stellen Sie sicher, dass die Datei `schnellstart/public/index.php` vorhanden ist.

Als letzten Schritt kopieren Sie bitte von der CD-ROM alle Inhalte aus dem Verzeichnis `ZendFramework-1.7.4/library/` nach `schnellstart/library/`, um die Einrichtung des Projektes abzuschließen.

Tipp

Alternativ können Sie voraussichtlich ab dem Release 1.8 auch `Zend_Tool` nutzen, um sich die Arbeit beim Anlegen eines neuen Projektes zu vereinfachen. Für weitere Informationen zu den Möglichkeiten schauen Sie bitte in Anhang A.2, Zend_Tool, nach.

Bevor wir mit dem Anlegen weiterer Dateien fortfahren, stelle ich Ihnen einige der bereits vorhandenen Dateien vor. Je nachdem, ob Sie die Projektvorlage von der CD-ROM verwenden oder `Zend_Tool` eingesetzt haben, sehen die angelegten Dateien anders aus.

3.2 Apache-Konfiguration

In diesem Kapitel erläutere ich Ihnen die Konfiguration des Apache Webservers für Ihr Projekt. Wenn Sie einen anderen Webserver verwenden wie z.B. den Microsoft IIS oder Lighttpd, finden Sie einige Konfigurationsanweisungen im Referenzhandbuch [03-002].

In dem öffentlichen Verzeichnis `schnellstart/public/` Ihres Projektes, in dem auch die Setup-Datei liegt, sollte eine Datei `.htaccess` enthalten sein. Listing 3.1 zeigt ein Beispiel für eine `.htaccess`-Datei.

Listing 3.1: Umschreiberegel in einer .htaccess-Datei definieren

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} -s [OR]
RewriteCond %{REQUEST_FILENAME} -l [OR]
RewriteCond %{REQUEST_FILENAME} -d
RewriteRule ^.*$ - [NC,L]
RewriteRule ^.*$ index.php [NC,L]
```

Ich weiß, was Sie jetzt denken, und Sie haben Recht! Wenn Sie sich zum ersten Mal mit einer Umschreiberegel für den Apache Webserver konfrontiert sehen, wird sich Ihnen der Sinn dieser Zeilen nicht sofort erschließen.

Sinn und Zweck dieser Zeilen ist es, dass alle Anfragen an den Webserver an die Setup-Datei `index.php` in Ihrem öffentlichen Verzeichnis umgeleitet werden. Mithilfe der sogenannten RewriteEngine können Sie suchmaschinenfreundliche URLs realisieren und die URL *http://example.com/?controller=pizza&action=buy&id=123* in die URL *http://example.com/pizza/buy/id/123* umschreiben lassen.

In der ersten Zeile wird die RewriteEngine aktiviert. Danach folgt die eigentliche Umschreiberegel. Übersetzt bedeuten diese fünf Zeilen Folgendes: Wenn die angeforderte Ressource `{REQUEST_FILENAME}` ein gültiger symbolischer Link (`-s`) oder eine nicht leere Datei (`-l`) oder ein Verzeichnis (`-d`) ist, liefere die angeforderte Ressource aus (1. RewriteRule). Andernfalls leite die Anfrage an die Setup-Datei `index.php` um (2. RewriteRule). Ignoriere dabei die Groß- und Kleinschreibung (`[NC]`) und schließe die Bearbeitung sofort ab, sobald eine der Regeln erfüllt ist `[L]`.

Sie sollten jedoch beachten, dass Verweise auf nicht vorhandene Dateien somit auch an die Setup-Datei umgeleitet werden. Wenn Sie also ein Bild einfügen und diese Datei existiert nicht auf dem Server, dann wird auch auf die `index.php` umgeleitet. Dies kann sich besonders bei vielen, kleinen Platzhaltergrafiken, die nicht oder nicht mehr existieren, sehr negativ auf die Geschwindigkeit Ihrer Zend Framework-Anwendung auswirken. Sie sollten also immer darauf achten, dass alle referenzierten Dateien auch tatsächlich vorhanden sind.

Ich hoffe, der Inhalt der `.htaccess`-Datei ist Ihnen nun klarer. Wenn Sie weitere Infos zur Verwendung von `mod_rewrite` suchen, empfehle ich Ihnen eine Website [03-003], auf der Sie Hilfestellungen, Tutorials und ein Forum rund um dieses Apache-Modul finden.

Tipp

Sollte das Umschreiben der URLs nicht funktionieren, ist wahrscheinlich das Modul `mod_rewrite` nicht für Ihren Apache aktiviert. Je nach der von Ihnen verwendeten Apache-Version läuft die Aktivierung etwas anders. Beim Apache unter Windows müssen Sie z.B. in der Datei `httpd.conf` folgende Zeilen auskommentieren:

```
LoadModule rewrite_module modules/mod_rewrite.so
AddModule mod_rewrite.c
```

Unter Linux (z.B. Ubuntu) aktivieren Sie das Modul über folgende Eingabe in der Konsole:

```
sudo a2enmod rewrite
```

Danach müssen Sie den Apache noch einmal neu starten.

3.3 Setup-Datei

Die Setup-Datei `index.php` liegt im Verzeichnis `schnellstart/public/` und sollte die einzige ausführbare PHP-Datei in Ihrem Projekt sein. Sie ist für die Konfiguration der PHP-Umgebung und die Ausführung des Front-Controllers zuständig. Der Front-Controller stellt sicher, dass alle Anfragen an den Webserver durch diese Datei verarbeitet werden können. Listing 3.2 zeigt ein Beispiel für eine Setup-Datei.

Listing 3.2: Beispiel für eine index.php-Setup-Datei

```
<?php
define('APPLICATION_PATH', realpath(dirname(__FILE__) . '/../'));

if ($_SERVER['HTTP_HOST'] == 'localhost') {
    define('APPLICATION_ENVIRONMENT', 'development');
} else {
    define('APPLICATION_ENVIRONMENT', 'production');
}

$paths = array(
   APPLICATION_PATH . '/library',
   APPLICATION_PATH . '/application/models',
   '.',
```

```
);
ini_set('include_path', implode(PATH_SEPARATOR, $paths));

error_reporting(E_ALL | E_STRICT);

require_once 'Zend/Loader.php';
Zend_Loader::registerAutoload();

include APPLICATION_PATH . '/application/bootstrap.php';

Zend_Controller_Front::getInstance()->dispatch();
```

Die ersten zehn Zeilen haben die Aufgabe, die PHP-Umgebung zu konfigurieren. Zuerst wird eine Konstante `APPLICATION_PATH` für das Wurzelverzeichnis der Anwendung und eine Konstante `APPLICATION_ENVIRONMENT` anhand des Servernamens definiert. Danach werden einige Verzeichnisse in den `include_path` aufgenommen. Neben dem Verzeichnis, in dem die Zend Framework-Dateien liegen, wird auch das Verzeichnis aufgenommen, das die Modelklassen enthält. Zudem wird das Error Reporting für alle Fehlerklassen `E_ALL` inklusive der strikten Fehler `E_STRICT` aktiviert.

Danach wird die Klasse `Zend_Loader` geladen, und in der `setup()`-Methode wird die Autoload-Funktionalität von `Zend_Loader` registriert. Dies hat den Vorteil, dass Sie bei der Verwendung von Zend Framework-Komponenten nicht mehr darauf achten müssen, ob die entsprechende Klasse bereits geladen wurde. Interessant ist die nächste Zeile, in der die Bootstrap-Datei `bootstrap.php` inkludiert und damit der dort enthaltende PHP-Code auch ausgeführt wird. Was hier genau passiert, sehen Sie im nächsten Kapitel.

Die letzte Zeile bringt Sie das erste Mal einer Komponente des Zend Framework näher. Der Front-Controller wird geladen und deren `dispatch()`-Methode ausgeführt. Mit dieser einzelnen Zeile wird somit der ganze Ablauf des Front-Controllers gestartet. Wenn Sie jetzt schon wissen möchten, was bei diesem Aufruf genau passiert, können Sie einen Blick in Kapitel 5.3, Front-Controller, werfen.

Hinweis

Ab dem Release 1.8 wird voraussichtlich die neue Komponente `Zend_Application` bereitstehen. Damit steht Ihnen eine Alternative zur Konfiguration Ihrer Zend Framework- Anwendung bereit, die sich auch auf die Setup-Datei auswirken wird. Einen ersten Vorgeschmack finden Sie in Anhang A.1, Zend_Application.

3.4 Bootstrap-Datei

Während die Setup-Datei für die Konfiguration der PHP-Umgebung zuständig ist, liegt die Konfiguration der Zend Framework-Umgebung in der Verantwortung der Bootstrap-Datei in `schnellstart/application/bootstrap.php`. Prinzipiell können Sie in Ihrer Bootstrap-Datei sowohl prozeduralen Code verarbeiten lassen oder eine Klasse definieren, welche die Konfiguration objektorientiert durchführt. Das Listing 3.3 zeigt Ihnen den Code für eine prozedurale Implementierung.

Listing 3.3: Beispiel für eine Bootstrap-Datei

```
<?php
$frontController = Zend_Controller_Front::getInstance();
$frontController->setControllerDirectory(
    APPLICATION_PATH . '/application/controllers'
);
$frontController->setParam('root', APPLICATION_PATH);

unset($frontController);
```

Bei dieser Implementierung handelt es sich um prozeduralen Code, der direkt beim Laden ausgeführt wird. Diese Bootstrap-Datei ist recht übersichtlich. Es wird nur das Verzeichnis für die Controller-Klassen unter Verwendung der `APPLICATION_PATH`-Konstante festgelegt. Zudem wird das Applikationsverzeichnis an den Front-Controller übergeben. Am Ende werden alle lokal in der Bootstrap-Datei verwendeten Variablen gelöscht.

In Ihrer Bootstrap-Datei können Sie auch weitere Komponenten initialisieren. Zum Beispiel wäre es sinnvoll, einen Datenbankadapter zu initialisieren, Konfigurationsdaten zu lesen oder den View einzurichten. Eine vollständige Bootstrap-Datei finden Sie in Kapitel 19.2, Projekt einrichten, in **Teil III** dieses Buches.

Alternativ können Sie Ihre Bootstrap-Datei auch objektorientiert gestalten. Ob Sie ein Objekt initialisieren und dessen Methoden aufrufen oder ob Sie mit statischen Methoden arbeiten, bleibt Ihnen überlassen. Es bietet sich hierbei an, für jeden einzelnen Schritt eine eigene Methode zu schreiben und diese dann nacheinander aufzurufen bzw. eine Setup-Methode zu schreiben, die diese Methoden aufruft. So wären einzelne Methoden zum Einrichten des Front-Controllers, zum Laden der Konfigurationsdaten, zum Initialisieren der Datenbank und zum Anlegen eines Cache-Objektes denkbar.

Hinweis

Ab dem Release 1.8 wird voraussichtlich die neue Komponente `Zend_Application` bereitstehen. Wenn Sie diese Komponente einsetzen, wird Ihre Bootstrap-Datei anders aussehen bzw. kann sogar entfallen und durch eine Konfigurationsdatei ersetzt werden. Einen ersten Ausblick finden Sie in Anhang A.1, Zend_Application.

Wenn Sie in Ihrem Projekt mehrere Module verwenden möchten, muss die Bootstrap-Datei ein wenig anders aussehen. Damit der Front-Controller weiß, dass Sie in Ihrem Projekt Module verwenden möchten, müssen Sie ihm dies auch mitteilen. Listing 3.4 zeigt, wie Sie dem Front-Controller das Modulverzeichnis mitteilen können.

Listing 3.4: Dem Front-Controller das Modulverzeichnis mitteilen

```
<?php
$frontController = Zend_Controller_Front::getInstance();
$frontController->setControllerDirectory(
   APPLICATION_PATH . '/application/controllers'
);
```

```
$frontController->addModuleDirectory(
   APPLICATION_PATH . '/application/modules'
);

[...]
```

Weitere Informationen zum Einsatz mehrerer Module finden Sie auch in den Kapiteln 2.6, Mehrere Module verwenden, und 18, Modularisierung von Anwendungen.

3.5 Controller

Nun wenden wir uns den Controllern und damit der `Zend_Controller`-Komponente zu. Informationen über Controller in einer MVC-Architektur finden Sie in Anhang B.1.7, Der Controller, sowie Informationen zu `Zend_Controller` in Kapitel 5, Controller.

3.5.1 Begriffe abgrenzen

Zum Einstieg müssen Sie wissen, dass Sie in einer Zend Framework-Anwendung alle Aktionen auf Ihrer Website in Action-Controller zusammenfassen können. Innerhalb eines Action-Controllers können Sie beliebig viele Aktionsmethoden einrichten. Theoretisch könnten Sie auch alle Aktionen Ihrer Website in einem einzigen Controller ablegen, dies würde ich aber zur Steigerung der Übersichtlichkeit nicht empfehlen.

Ein kleines Beispiel zur Abgrenzung der Begriffe Action-Controller und Aktionsmethoden. Stellen Sie sich für Luigis Pizzaservice einen Warenkorb vor. Sie müssen:

- neue Pizzen hinzufügen,
- bestehende Bestellpositionen ändern oder entfernen,
- den Warenkorb anzeigen
- und den Warenkorb als Bestellung absenden können.

Somit benötigen Sie einen Action-Controller für den Warenkorb, der die Aktionsmethoden zum Hinzufügen, Ändern, Entfernen, Anzeigen und Absenden enthält. Für die Pflege der Produkte (also der Pizzen) würde ich einen eigenen Action-Controller empfehlen. Auch für die Verwaltung der Bestellungen lohnt sich ein eigener Action-Controller. Alternativ können Sie auch mit Modulen arbeiten (siehe Kapitel 2.6, Mehrere Module verwenden) und die Controller in ein öffentliches Modul und ein `admin`-Modul aufteilen.

3.5.2 IndexController

In der `schnellstart/application/controllers/IndexController.php`- Datei finden Sie die Klasse `IndexController`. Diese enthält eine Aktionsmethode `indexAction()`. Diese Aktion wird beim Aufruf der Startseite ausgeführt. In jedem Projekt sollten Sie mindestens diesen Action-Controller anlegen.

Die Aktionsmethode `indexAction()` ist noch leer, was wir nun ändern. Wir möchten auf der Startseite unseres Beispielprojektes eine Liste von Pizzen ausgeben. Ändern Sie die `indexAction()`-Methode wie in Listing 3.5 angegeben. Zuerst werden die Daten definiert und danach an den View übergeben. Außerdem sehen Sie die neue Aktionsmethode `showAction()` zur Anzeige einer Pizza. Auch hier definieren wir feste Daten.

Listing 3.5: Action-Controller für Startseite

```
class IndexController extends Zend_Controller_Action
{
   public function indexAction()
   {
      $pizzaList = array(
         array(
            'pizza_id'          => '1',
            'pizza_name'        => 'Pizza Margarita',
            'pizza_description' => 'Pizza mit Tomaten und Käse.',
         ),
         array(
            'pizza_id'          => '2',
            'pizza_name'        => 'Pizza Hawaii',
            'pizza_description' => 'Pizza mit Tomaten, Käse, Ananas und '.
                                   'Kochschinken.',
         ),
      );
      $this->view->pizzaList = $pizzaList;
   }

   public function showAction()
   {
      $pizzaData = array(
         'pizza_id'          => '1',
         'pizza_name'        => 'Pizza Margarita',
         'pizza_description' => 'Pizza mit Tomaten und Käse.',
      );
      $this->view->pizzaData = $pizzaData;
   }
}
```

3.5.3 ErrorController

Neben dem `IndexController` sollten Sie auch einen `ErrorController` anlegen. Dieser hat die Aufgabe, Fehler in Ihrer Anwendung so auszugeben, dass der Benutzer keine direkten Fehlermeldungen ausgegeben bekommt. Das Zend Framework leitet bei allen Ausnahmen auf die Aktionsmethode `errorAction()` im `ErrorController` um. Dort können Sie eine allgemeine Meldung für den Benutzer ausgeben und die tatsächliche Fehlermeldung zum Beispiel in ein Log speichern (siehe auch 4.6, Zend_Log).

In der `schnellstart/application/controllers/ErrorController.php-` Datei finden Sie den `ErrorController` und die leere `errorAction()`-Methode. Ändern Sie diese Methode wie in Listing 3.6 gezeigt. Zuerst wird das interne Fehlerobjekt aus dem Request-Objekt geholt. Anhand des Typs werden Statuscodes für den Apache versandt bzw. ein Fehlertext an den View übergeben. Zum Schluss werden der aktuelle Wert der Umgebung sowie die Ausnahme an den View übergeben. Dazu gleich mehr.

Listing 3.6: Action-Controller für Fehlerverarbeitung

```
class ErrorController extends Zend_Controller_Action
{
   public function errorAction()
   {
      $errors = $this->getRequest()->getParam('error_handler');

      switch ($errors->type) {
         case Zend_Controller_Plugin_ErrorHandler::EXCEPTION_NO_CONTROLLER:
         case Zend_Controller_Plugin_ErrorHandler::EXCEPTION_NO_ACTION:
            $this->getResponse()->setHttpResponseCode(404);
            $this->view->message = 'Seite konnte nicht gefunden werden';
            break;
         default:
            $this->getResponse()->setHttpResponseCode(500);
            $this->view->message = 'Es ist ein Fehler aufgetreten';
            break;
      }
      $this->view->environment = APPLICATION_ENVIRONMENT;
      $this->view->exception   = $errors->exception;
   }
}
```

Mit diesen beiden Controllern können wir erst einmal starten. Die Anzahl der Action-Controller und Aktionsmethode pro Action-Controller ist wie erwähnt nicht begrenzt.

3.6 View und Layout

Als Nächstes ist der View an der Reihe. Informationen über den View in einer MVC-Architektur finden Sie in Anhang B.1.5, Der View. Hierfür setzen wir `Zend_View` und `Zend_Layout` ein, um ein zentrales Layout definieren zu können. Informationen zu beiden Komponenten finden Sie in Kapitel 6, View.

3.6.1 Zentrales Layout einrichten

Mit `Zend_Layout` können Sie ein zentrales Layout einrichten. Somit können Sie alle HTML-Angaben, die auf allen Seiten benötigt werden, an zentraler Stelle ablegen. Die Views können sich somit nur auf die Ausgabe für die jeweilige Aktion konzentrieren.

Um ein zentrales Layout für Ihre Website einrichten zu können, legen Sie zuerst die Datei `schnellstart/application/layouts/scripts/layout.phtml` wie in Listing 3.7 an. Entscheidend sind die Ziele `$this->layout()->content`, weil hier mithilfe eines View-Helpers von `Zend_Layout` der Inhalt der Aktion ausgegeben wird.

Listing 3.7: Zentrale Layout-Datei für Ihre Website

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
          "http://www.w3.org/TR/html4/loose.dtd">
<html>
```

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Luigis Pizza Service</title>
</head>
<body>
<div id="page" style="width: 800px;">
<div id="title" style="border-bottom: 1px solid;"><h1>Luigis Pizzas</h1></div>
<div id="main">
<div id="menu" style="float:left; width: 120px; margin-right: 10px;">
<ul style="list-style: none; margin: 0; padding: 0;">
<? $url = $this->url(
   array('controller' => 'index', 'action' => 'index'), null, true
); ?>
<li><a href="<?= $url ?>">Pizzaliste</a></li>
</ul>
</div>
<div id="content" style="border-left: solid 150px #fff; background: #eee;">
<?= $this->layout()->content ?>
</div>
<br class="clear" />
</div>
<div id="footer" style="text-align: center; border-top: 1px solid;">
&copy; 2009 Luigis Pizza Service
</div>
</div>
</body>
</html>
```

Damit die Zend Framework-Anwendung auch weiß, dass Sie `Zend_Layout` einsetzen möchten, müssen Sie noch die folgende Zeile an das Ende Ihrer Bootstrap-Datei in `schnellstart/application/bootstrap.php` hinzufügen:

```
Zend_Layout::startMvc(APPLICATION_PATH . '/application/layouts/scripts');
```

Für weitere Details zu `Zend_Layout` schauen Sie bitte in das Kapitel 6.5, Zend_Layout.

3.6.2 View-Skript für Startseite mit Pizzaliste

Für die Startseite mit der Pizzaliste öffnen Sie die Datei `index.phtml` im Verzeichnis `schnellstart/application/views/scripts/index/`. Löschen Sie nun die Inhalte, und fügen Sie den Programmcode wie in Listing 3.8 ein. Damit werden unsere Pizzen auf der Startseite ausgegeben. Dieses View-Skript besteht aus HTML-Code sowie einigen PHP-Codefragmenten. Details finden Sie in Kapitel 6.2, Zend_View-Skripte.

Listing 3.8: View-Skript für die Ausgabe der Pizzaliste

```
<h2>Pizzaliste</h2>
<p>Bitte wählen Sie eine Pizza aus!</p>
<table border="1">
<tr><th>Pizza</th><th>Beschreibung</th></tr>
<? foreach ($this->pizzaList as $pizzaData) : ?>
```

```
<? $url = $this->url(array(
   'controller' => 'index', 'action' => 'show', 'id' => $pizzaData['pizza_id']
)); ?>
<tr>
<td><a href="<?= $url ?>"><?= $pizzaData['pizza_name'] ?></a></td>
<td><a href="<?= $url ?>"><?= $pizzaData['pizza_description'] ?></a></td>
</tr>
<? endforeach; ?>
</table>
```

Der Zugriff auf die an den View übergebenen Variablen mit den Pizzadaten erfolgt über `$this->pizzaList`. Bei `$this` handelt es sich um die Instanz von `Zend_View`, sodass Sie auf alle übergebenen Variablen als Eigenschaften dieser Instanz zugreifen können.

Erwähnenswert ist auch `$this->url(...)`, womit ein View-Helper aufgerufen wird. View-Helper werden benutzt, um sich wiederholende Ausgaben in View-Skripten an zentraler Stelle zu kapseln. Bei diesem eingesetzten View-Helper wird anhand von Parametern für Controller, Aktion usw. eine URL erstellt. Mehr Informationen zu View-Helpern finden Sie in Kapitel 6.3, View-Helper.

3.6.3 View-Skript für Anzeige einer Pizza

Legen Sie für die Anzeige einer einzelnen Pizza die Datei `show.phtml` im Verzeichnis `schnellstart/application/views/scripts/index/` an, und fügen Sie den Programmcode aus Listing 3.9 ein. Der Datenzugriff erfolgt über `$this->pizzaData`.

Listing 3.9: View-Skript für die Ausgabe einer Pizza

```
<h2><?= $this->pizzaData['pizza_name'] ?></h2>
<p><?= $this->pizzaData['pizza_description'] ?></p>
<hr />
<? $url = $this->url(
   array('controller' => 'index', 'action' => 'index'), null, true
); ?>
<p><a href="<?= $url ?>">zurück zur Pizzaliste</a></p>
```

Da die übergebenen Daten vom Controller fest definiert sind, würden bei jedem Aufruf des View-Skriptes immer dieselben Daten ausgegeben werden. Dazu gleich mehr.

3.6.4 View-Skript für die Anzeige von Fehlern

Als letztes View-Skript benötigen wir noch eines für die Ausgabe der Fehler. Öffnen Sie `schnellstart/application/views/scripts/error/error.phtml`, und ändern Sie den Programmcode wie in Listing 3.10 angegeben.

Listing 3.10: View-Skript zur Ausgabe von Fehlern

```
<h2>Uups</h2>
<p><?= $this->message ?></p>
<? if ('development' == $this->environment): ?>
```

```
<h3>Ausnahme:</h3>
<p><b>Nachricht:</b> <?= $this->exception->getMessage() ?></p>
<h3>Stack trace:</h3>
<pre><?= $this->exception->getTraceAsString() ?></pre>
<? endif ?>
```

Abhängig vom übergebenen Wert für die Umgebung werden auch Details zur Ausnahme ausgegeben. Auf dem Produktionsserver sieht der Besucher nur die Fehlermeldung.

Hinweis

Nach dem Anlegen der View-Skripte ist es nun an der Zeit, dass Sie die Anwendung einmal in Ihrem Browser aufrufen. Je nachdem, wo Sie das Projekt abgelegt haben, kann sich der Aufruf unterscheiden. Liegt das Verzeichnis `schnellstart/` direkt in dem Webserver- Verzeichnis, könnte der Aufruf wie folgt aussehen:

http://localhost/schnellstart/public/

Wenn Sie die passende URL aufrufen, sollten Sie eine Pizzaliste sehen. Beim Klick auf eine der beiden Pizzen sollte diese wieder angezeigt werden. Rufen Sie eine fehlerhafte URL auf, und Sie sollten auch die Fehlerseite sehen können:

http://localhost/schnellstart/public/falsch/

3.7 Konfiguration

Jede moderne Webanwendung benötigt Konfigurationsdaten. Sei es für die Datenbank, für zentrale E-Mail-Adressen oder für den Dateipfad der Logdateien. Das Zend Framework hält für diese Zwecke die `Zend_Config`-Komponente bereit. Damit können Sie Ihre Konfigurationsdaten in einer INI-Datei, als XML oder in einem PHP-Array verwalten.

3.7.1 Konfigurationsdatei erstellen

Wir entscheiden uns für das Speichern unserer Konfigurationsdaten im INI-Format. Legen Sie die Datei `schnellstart/application/config/config.ini` neu an, und fügen Sie den Code aus Listing 3.11 ein.

Listing 3.11: Konfigurationsdatei im INI-Format

```
[production]
db.adapter = Pdo_Sqlite
db.params.dbname = "/local/database/server/luigiblog.sqlite"
db.params.sqlite2 = true

[development : production]
db.params.dbname = "../data/pizza.sqlite"
```

Die Konfigurationsdaten werden getrennt für die aktuelle Umgebung definiert. Hierbei werden die beiden Umgebungen `production` und `development` unterschieden. Durch Angabe

der Umgebung in eckigen Klammern können Sie die Konfigurationsdaten zuordnen. Zusätzlich können Sie auch mit Vererbung arbeiten, indem Sie den Doppelpunkt verwenden. In unserem Beispiel sind die Daten für die beiden Umgebungen somit mit Ausnahme der Angabe `db.params.dbname` identisch.

3.7.2 Konfigurationsdatei laden und verfügbar machen

Als Nächstes müssen wir die Konfigurationsdatei laden und so verfügbar machen, dass wir von jeder Stelle in unserer Anwendung auf diese Konfigurationsdaten zugreifen können. Hier kommt nun `Zend_Config` zum Einsatz. Fügen Sie am Anfang Ihrer Bootstrap-Datei in `schnellstart/application/bootstrap.php` folgende Zeilen ein:

```
$config = new Zend_Config_Ini(
   APPLICATION_PATH . '/application/config/config.ini', APPLICATION_ENVIRONMENT
);
Zend_Registry::set('config', $config);
```

Zuerst werden die Konfigurationsdaten geladen, um eine Instanz von `Zend_Config` zu erstellen. Um diese Instanz auch in der gesamten Anwendung verfügbar zu machen, setzen wir `Zend_Registry` ein. Damit können Sie z. B. in einem Action-Controller oder in einem View-Skript auf die Daten zugreifen:

```
$config = Zend_Registry::get('config');
echo $config->db->adapter;
```

Weitere Informationen zu `Zend_Config` finden Sie in Kapitel 4.4, Zend_Config, und über `Zend_Registry` in Kapitel 4.3, Zend_Registry.

3.8 Models und Datenbanken

Da das Arbeiten mit statisch definierten Daten irgendwie langweilig ist, geht es diesen nun an den Kragen. Wir richten die Models für unsere kleine Beispielanwendung ein. Informationen über die Models in einer MVC-Architektur finden Sie in Anhang B.1.3, Das Model. Zur Umsetzung unseres Models setzen wir `Zend_Db_Adapter` und `Zend_Db_Table` ein. Informationen zu diesen Komponenten finden Sie in Kapitel 7, Datenbanken.

3.8.1 Vorbereitungen

Als Erstes benötigen wir ein paar Daten. Kopieren Sie dafür von der CD-ROM die Datei `files/database/pizza.sqlite` nach `schnellstart/data/pizza.sqlite`. In dieser Datei ist eine SQLite-Datenbank enthalten, die zum Testen ausreichend ist. Alternativ können Sie auch den Dump aus `files/database/pizza.sql` verwenden.

Ändern Sie die `schnellstart/application/bootstrap.php`-Datei, um den Datenbankadapter einzurichten. Fügen Sie ans Ende die folgenden Zeilen ein:

```
Zend_Registry::set('db', Zend_Db::factory($config->db));
Zend_Db_Table_Abstract::setDefaultAdapter(Zend_Registry::get('db'));
```

Mit der ersten Zeile erstellen wir eine Datenbankadapter-Instanz. Dabei kommen die Konfigurationsdaten aus dem vorherigen Kapitel zum Einsatz. Dieser Datenbankadapter wird ebenfalls mithilfe von `Zend_Registry` applikationsweit verfügbar gemacht. Mit der zweiten Zeile legen wir fest, dass alle Instanzen von `Zend_Db_Table` diesen Datenbankadapter als Standardadapter verwenden sollen.

3.8.2 Models einrichten

Um die Sache zu vereinfachen, setzen wir `Zend_Db_Table` als Basis für unsere Models ein. Dass Models mehr sein können als nur Datenbanktabellen, auf die Sie zugreifen, finden Sie in Kapitel 17, Models umsetzen, erläutert. Sie sollten sich dieses Kapitel genauer anschauen, wenn Sie mehr Erfahrungen mit dem Zend Framework haben.

Legen Sie nun die Datei `schnellstart/application/models/Pizzas.php` an. Dort fügen Sie den Inhalt aus Listing 3.12 ein. Mithilfe der Klasse `Pizzas` können wir nun auf die Inhalte der Tabelle `pizzas` in unserer SQLite-Datenbank zugreifen.

Listing 3.12: Model zum Zugriff auf die Tabelle pizzas

```
class Pizzas extends Zend_Db_Table_Abstract
{
   protected $_name    = 'pizzas';
   protected $_primary = 'pizza_id';

   public function fetchRowWithIngredients($id)
   {
      $id = Zend_Filter::get($id, 'Int');
      $data = $this->find($id)->current();
      if (empty($data)) {
         throw new Zend_Db_Exception('Pizza ID "' . $id . '" ist ungültig');
      }

      $data = $data->toArray();

      $select = $this->getAdapter()->select();
      $select->from('ingredients');
      $select->join(
         'pizza_ingredients', 'pi2in_ingredient_id = ingredient_id', array()
      );
      $select->where('pi2in_pizza_id = ?', $id);

      $data['ingredients'] = $this->getAdapter()->fetchAll($select);
      return $data;
   }
}
```

Als Erweiterung haben wir die Methode `fetchRowWithIngredients()` erstellt:

- Der übergebene Parameter wird mithilfe von `Zend_Filter` gefiltert, damit nur Ziffern verwendet werden (siehe auch Kapitel 4.8, Zend_Filter).
- Die Daten für die aktuelle Pizza werden gelesen.

- Konnten keine Daten gelesen werden, wird eine Ausnahme geworfen.
- Die gelesenen Daten werden ins Array-Format umgewandelt.
- Es wird eine Datenbankabfrage mithilfe von `Zend_Db_Select` erstellt, um die Zutaten für die Pizza zu lesen (siehe auch Kapitel 7.3, Zend_Db_Select).
- Die Daten für die Zutaten werden gelesen und mit den Daten für die Pizza kombiniert.

Sie sehen hierbei, dass Sie Ihre von `Zend_Db_Table` abgeleitete Klasse auch mit eigenen Methoden erweitern können. Alternativ können Sie auch die Unterstützung von Relationen zwischen Datenbanktabellen verwenden (siehe Kapitel 7.7, Tabellenrelationen).

3.8.3 Models verwenden

Nun möchten wir das schöne neue Model auch einsetzen. Dafür müssen wir die Aktionsmethoden in unserem Action-Controller IndexController wie in Listing 3.13 anpassen. In beiden Methoden wird eine Instanz der `Pizzas`-Klasse initialisiert. Danach werden jeweils die gewünschten Daten gelesen und an den View übergeben.

Listing 3.13: Geänderter Action-Controller, der Model einsetzt

```
class IndexController extends Zend_Controller_Action
{
   public function indexAction()
   {
      $pizzas = new Pizzas();
      $pizzaList = $pizzas->fetchAll()->toArray();
      $this->view->pizzaList = $pizzaList;
   }

   public function showAction()
   {
      $id = $this->getRequest()->getParam('id');
      $pizzas = new Pizzas();
      $pizzaData = $pizzas->fetchRowWithIngredients($id);
      $this->view->pizzaData = $pizzaData;
   }
}
```

Somit können die echten Daten ausgegeben werden. Wenn Sie in Ihrem Webbrowser die Anwendung unter *http://localhost/schnellstart/public/* aufrufen, sollte Ihnen eine Liste mit fünf Pizzen ausgegeben werden. Sobald Sie auf eine der Pizzen klicken, werden Ihnen die Daten dieser Pizza auch ausgegeben. Geben Sie eine falsche Adresse ein, sollte die Fehlerseite ausgegeben werden. Probieren Sie es selber mit *http://localhost/schnellstart/public/index/show/id/abc* aus.

3.8.4 View anpassen für Ausgabe der Zutaten

Als letzten Schritt müssen Sie nun noch das View-Skript für die Ausgabe einer Pizza anpassen, damit Sie auch die gelesenen Zutaten anzeigen können. Ändern Sie die Datei `show.phtml` in `schnellstart/application/views/scripts/index/` wie in Listing 3.14 dargestellt.

Listing 3.14: Geändertes View-Skript zur Anzeige der Zutaten pro Pizza

```
<h2><?= $this->pizzaData['pizza_name'] ?></h2>
<p><?= $this->pizzaData['pizza_description'] ?></p>
<h3>Zutaten dieser Pizza</h3>
<ul>
<? foreach ($this->pizzaData['ingredients'] as $ingredient) : ?>
<li><?= $ingredient['ingredient_name'] ?></li>
<? endforeach; ?>
</ul>
<hr />
<? $url = $this->url(
   array('controller' => 'index', 'action' => 'index'), null, true
); ?>
<p><a href="<?= $url ?>">zurück zur Pizzaliste</a></p>
```

Rufen Sie erneut *http://localhost/schnellstart/public/* auf, und klicken Sie eine Pizza an. Nun sollten auch die Zutaten in einer Liste ausgegeben werden.

3.9 Zusammenfassung

Herzlichen Glückwunsch! Sie haben Ihre erste kleine Zend Framework-Anwendung erstellt. Dabei haben Sie nach der Einrichtung des Projektes gelernt, welche Aufgaben die Setup-Datei und die Bootstrap-Datei haben. Sie haben die wichtigsten Bausteine Controller, View und Model einer MVC-Architektur kennengelernt und wissen auch, wie Sie Konfigurationsdaten verwenden und die Registry zum applikationsweiten Ablegen von Objekten einsetzen können.

Auf der CD-ROM im Verzeichnis `kapitel03/schnellstart/` finden Sie übrigens auch die komplette Anwendung zum Vergleich. Um das Gelernte zu untermauern, können Sie nun in **Teil II** des Buches die vielen Komponenten des Zend Framework kennenlernen. Oder Sie machen gleich weiter mit **Teil III** und schlagen bei Bedarf die Informationen zu den Komponenten in **Teil II** nach.