Case Study: Wie mit CakePHP und MySQL ein Buchungssystem für 900 Mitarbeiter in nur sechs Wochen umgesetzt wurde

Die Turbo-Backstube

In vielen Unternehmen werden Daten in manueller Form auf Papier oder in einfach strukturierten Excel-Tabellen erfasst und weiterverarbeitet. Eingabe- und Übertragungsfehler sind vorprogrammiert, doch der Einsatz einer Softwareanwendung erscheint auf den ersten Blick aufwändig und teuer. Insbesondere dann, wenn keine Standardanwendung zur Verfügung steht, die mit geringem Customizing-Aufwand und geringen Lizenzkosten genutzt werden kann. Dass dennoch eine Individuallösung innerhalb kurzer Zeit und mit begrenzten finanziellen und personellen Ressourcen umgesetzt werden kann, zeigt dieses Beispiel.

von Ralf Hohoff und Thomas Recke

Das Unternehmen in diesem beispielhaften Projekt übernimmt mit über 1700 Mitarbeitern die deutschlandweite telefonische und schriftliche Kundenbetreuung für namhafte Unternehmen mit speziellem Fokus auf das Versandhandelgeschäft. Abhängig von seinen Fähigkeiten kann ein Mitarbeiter des Servicecenters in verschiedenen Kundenprojekten eingesetzt werden. Die Mitarbeiter erfassen im Kundendialog Informationen zu verschiedenartigen Geschäftsvorgängen wie Bestellannahmen oder auch Beschwerdeanrufe. Je nach Vorfallsart werden dem Auftraggeber unterschiedliche Preise berechnet. Basis für diese Berechnung bilden die Aufzeichnungen des Mitarbeiters, die bisher auf unterschiedlichsten Formularvorlagen, teilweise in Papierform, teilweise in MS Excel oder MS Access erfasst wurden. Im Rahmen der Einführung eines Business Intelligence Tools für alle operativen Einheiten sollten verschiedene Datenquellen eingebunden werden. Insbesondere die von den Mitarbeitern erfassten Aktivitäten bilden einen wesentlichen Bestandteil für die Deckungsbeitragsrechnung, die mit der BI-Anwendung realisiert werden sollte. Die Einbindung der Vorgänge gestaltete

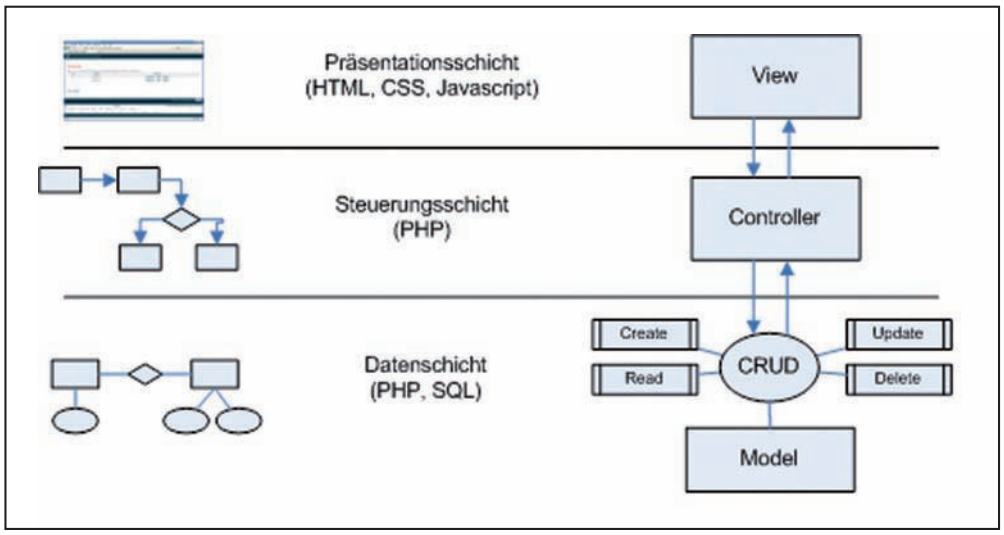

Abb. 1: Übersicht der logischen Schichten im MVC-Konzept

sich in der Umsetzung aufgrund der unterschiedlichen Erfassungsarten und -formate als sehr aufwändig. Zwar war dem Unternehmen bewusst, dass eine einheitliche Datenerfassung und zentrale Speicherung der Informationen signifikante Vorteile gegenüber den existierenden Insellösungen (MS Excel, Papierformulare) bringen würde (Kasten: „Vorteile einer einheitlichen Datenerfassung gegenüber Insellösungen"), aufgrund von begrenzten Ressourcen (Personal, Zeit, Kosten) wurde diese Lösung bisher vom Unternehmen nicht realisiert.

Zudem war weder die Suche nach Standardprodukten erfolgreich noch ein geeigneter technischer Implementierungsansatz bekannt. Da im Zuge des BI-Projekts bereits für die Übernahme der Informationen eine Analyse der Datenstrukturen vorgenommen wurde, ergab sich die Gelegenheit, dies mit geringem Mehraufwand in eine Konzeption für ein zentrales Buchungssystem fortzuführen. Schnell stand die Entscheidung fest, die neue Anwendung auf Basis des empfohlenen Lösungsvorschlages umzusetzen. Wie sah der Lösungsansatz nun konkret aus?

Lösungsansatz

Wenn keine Standardanwendung zur Verfügung steht, die mit geringem Customizing-Aufwand und geringen Lizenzkosten genutzt werden kann, bleibt unter diesen Voraussetzungen nur die Möglichkeit einer Individuallösung. Die begrenzten finanziellen und personellen Ressourcen sind nur einzuhalten, wenn ein standardisiertes Framework eingesetzt wird, das benötigte Basisfunktionen direkt zur Verfügung stellt. Auf diese Weise kann sich das Projektteam auf die Umsetzung der fachlichen Anforderungen konzentrieren. Das Projektteam bestand keinesfalls nur aus erfahrenen externen Entwicklern, sondern zum Hauptteil aus unternehmensinternen Mitarbeitern mit geringer Entwicklungserfahrung. Es war ein erklärtes Ziel, die Umsetzung durch eine externe Beratung begleiten zu lassen und nach dem ersten Rollout die Entwicklung ausschließlich mit internen Ressourcen abbilden zu können.

Warum CakePHP?

Für die Umsetzung des Buchungssystems wurde das quelloffene Framework CakePHP ausgewählt. Die Punkte, die zu dieser Entscheidung beigetragen haben, werden im Folgenden erläutert.

Philosophie: Vergleichbar mit anderen Frameworks versucht auch CakePHP, die Konfiguration auf ein Minimum zu beschränken. Nachdem die Verbindung zur Datenbank erstellt ist, ermittelt CakePHP die Struktur des zugrunde liegenden Datenmodells anhand von Namenskonventionen. Hierbei werden keine Konfigurationsdateien benötigt, dieses Prinzip ist als Convention over Configuration (CoC) bekannt. Doch dazu später mehr (siehe MVC).

Open Source: Durch das quelloffene Framework entstehen keine weiteren Lizenzkosten, die das schmale Projektbudget belasten. Mit dem Releasestand 1.2 verfügt das Framework über eine notwendige Reife und gleichzeitig über das Potenzial zur kontinuierlichen Weiterentwicklung durch die Entwicklergemeinschaft. Da das System nicht proprietär ist, können einfach bei Bedarf externe Entwickler hinzugezogen werden.

Vorteile einer einheitlichen Datenerfassung gegenüber Insellösungen

- Einheitliche Benutzeroberfläche, unabhängig vom Kundenprojekt
- Zentrale Datenspeicherung
  - Auswertungen
  - Datensicherung
  - Kein Zeitversatz zwischen Erfassung und Auswertung
- Einrichtung (Setup) eines neuen Kundenprojekts einfach und schnell, da Struktur vorgegeben und bewährt, Adaption von bestehenden Projekten
- Erweiterungen und Anpassungen finden einmalig im Unternehmen statt und nicht in jeder Insel (Businesslogik)
- Geringerer Aufwand in Wartung und Support (Fehlersuche)

PHP: Bei den Mitarbeitern des Auftraggebers bestanden durch zwei interne Projekte (CMS, Auktionsplattform) Basiskenntnisse in HTML, CSS und PHP sowie MySQL. Deshalb wurden Frameworks, die auf Ruby oder Java basieren, nicht in die Auswahl einbezogen.

MVC: CakePHP ist nach dem Prinzip Model View Controller (MVC) aufgebaut. Das Konzept verfolgt den Ansatz, den Quellcode der Anwendung gezielt in drei logische Schichten zu trennen. Ziel ist ein modularer Programmentwurf, der eine spätere Änderung oder Erweiterung erleichtert und eine Wiederverwendung der einzelnen Komponenten ermöglicht.

In der untersten Schicht des Datenmodells (engl. model) werden die zugehörige Datenbanktabellen und Abhängigkeiten zu anderen Models (und damit anderen Tabellen) gespeichert. Jedes Model bringt eine Anzahl an Methoden mit, die zur Verwaltung der Daten genutzt werden. Mit der Anlage des Models werden die entsprechenden Methoden zum Lesen, Speichern, Verändern und Löschen von Datensätzen bereitgestellt (Create, Retrieve, Update, Delete = CRUD-Funktionalität). Darüber hinaus werden weitere Funktionen mitgeliefert, z. B. zum Filtern oder Sortieren von Daten. Die Programmsteuerung übernimmt der Controller, der Benutzeraktionen aus der Präsentationsschicht (engl. view) entgegennimmt und auswertet. Der Controller führt die verschiedenen Funktionen (CRUD) aus und verbindet auf diese Weise Model und View.

Die View hat einzig und allein die Aufgabe, die Daten darzustellen und Benutzereingaben anzunehmen, um sie an den Controller weiterzureichen. CakePHP verfügt über ein Template-System mit unterschiedlichen Themes und häufig benötigten Funktionen für Oberflächenelemente wie Tabellen oder Links.

Unsere Küchenwerkzeuge

Im Projekt wurden einige praxiserprobte Werkzeuge eingesetzt, die die Erstellung und Verwaltung des Quellcodes und des Datenbanklayouts unterstützen. Als Entwicklungsumgebung wurde PHP Eclipse eingesetzt, welche mittels Plug-in direkt mit dem zentralen Quellcodeverwaltungssystem Subversion (SVN) verbunden war. Mit der Synchronisation der lokalen Arbeitsversionen zum zentralen Repository wurde die Zusammenarbeit im Team – bis zu fünf Entwickler arbeiten parallel – erst möglich gemacht, ohne dass nachher mühevoll einzelne Textdateien zusammengefügt werden mussten oder Arbeit von Kollegen ungewollt überschrieben wurde. Somit waren die Versionierung und das Einfrieren von Versionsständen für das Einspielen auf das Produktivsystem sichergestellt.

Das Datenbankschema wurde mit der MySQL-Workbench erstellt und verwaltet. Dieses Werkzeug ermöglicht zum einen die visuelle Darstellung des Datenbanklayouts mit all seinen Verknüpfungen und zum anderen die direkte Übertragung von Änderungen in die Datenbank. Da die Struktur in einer einzelnen Datei gespeichert wird, können unterschiedliche Versionsstände auch im SVN verwaltet werden.

Das Backen beginnt

In der Konzeption für das zentrale Buchungssystem war bereits das Datenbankschema entworfen worden, und es wurde am ersten Projekttag mittels MySQL-Workbench erstellt und in die Datenbank übertragen. Per Kommandozeile (Console Applications) konnten in CakePHP die Datenbankverbindungen ebenso wie die Models, Views und Controller eingerichtet werden. Mit der Verbindung zur Datenbank hat das Framework deren Struktur analysiert und die Abhängigkeiten zwischen den Tabellen ermittelt. Auf diese Weise konnten alle 27 Models und Controller der Anwendung in sehr kurzer Zeit erstellt werden. Hierbei legt das Framework automatisch die entsprechenden PHP-Dateien an. Über den Parameter *$scaffold* wurden die Controller zu diesem Zeitpunkt angewiesen, die Basisfunktionen (CRUD) dynamisch für die View-Schicht bereitzustellen. Das bedeutet, dass zur Laufzeit die Oberflächen ohne manuelle Erstellung der Views mit den Standardfunktionalitäten aufgebaut werden. Dazu gehört eine Listenansicht mit Blätter- und Sortierfunktion sowie Masken zur Eingabe und Bearbeitung von Datensätzen. Bei der Bearbeitung eines Datensatzes werden für die Felder, die vom Framework als Fremdschlüsselreferenzen identifiziert wurden, Auswahllisten über die referenzierten Inhalte angeboten. Dabei werden nicht etwa kryptische Schlüsselwerte (*ID*) sondern aussagekräftige Bezeichnungen angezeigt. Bei der Erfassung eines Autos könnten so in einer Auswahlliste die Automarken dargestellt werden. Auch die Links zum Anlegen des Datensatzes und der referenzierten Datensätze stehen direkt zur Verfügung.

Mit dieser Vorgehensweise und dem gewählten Lösungsansatz stand am Abend des ersten Projekttags eine vollständige Verwaltungsoberfläche mit Basisfunktionen zur Verfügung. So konnte parallel zur weiteren Entwicklung mit der Erfassung von Stammdaten begonnen und die Praxistauglichkeit des Datenbankmodells bewiesen werden. Optimierungsansätze wurden direkt sichtbar und konnten umgehend in die Entwicklung einfließen.

Anlernen der Küchenhelfer

Parallel zu diesem initialen Setup wurden die Mitarbeiter in einem eintägigen Workshop mit den fachlichen und technischen Rahmenbedingungen des Projekts und mit den Werkzeugen vertraut gemacht. Durch den Model-View-Controller-Ansatz konnten die anstehenden Aufgabenbereiche optimal entsprechend der unterschiedlichen Qualifizierungen der Mitarbeiter aufgeteilt werden – die Gestaltung der Oberfläche (Views) erfordert gute Kenntnisse in HTML, CSS und JavaScript sowie grundlegendes Wissen in PHP. Für die Implementierung der Programmlogik (Controller) sind sehr gute PHP-Kenntnisse notwendig und ein Verständnis der Objektorientierung für den Zugriff auf das Model. Der Zugriff erfolgt abstrakt,

international PHP2010 conference

Rheingoldhalle Mainz

11. – 13. Oktober 2010

www.phpconference.com

INTERNATIONAL PHP CONFERENCE 2010

Very Early Bird:
Bis zum **12. August** anmelden und **gratis** **Intellibook-Notebook** sichern!

Gold-Sponsoren:

MAYFLOWER · Microsoft · zend The PHP Company

Silber-Sponsoren:

inovex · it-agile · Peter Prochaska Internet Applikationen Security Consulting Schulungen

Bronze-Sponsor:

mitp

Special-sponsor:

jWebSocket www.jwebsocket.org

Media-Sponsoren:

CREATE OR DIE · entwickler magazin · m360bile

Präsentiert von:

PHPmagazin

Veranstalter:

S&S Software & Support Media

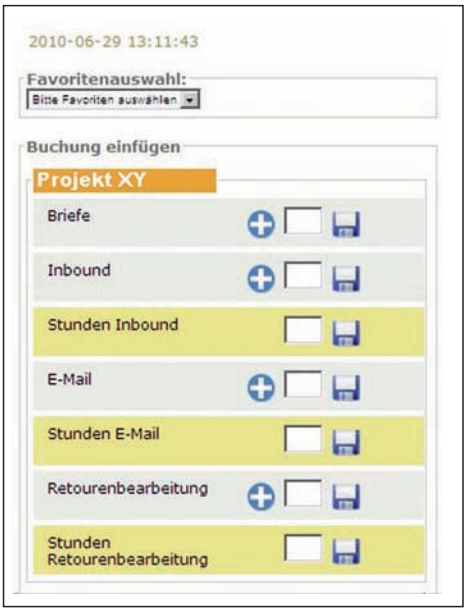

Abb. 2: Intuitive Erfassungsmaske mit AJAX vermeidet ein Neuladen der Seite und hält die Last auf dem Server niedrig

Datenbankwissen wird weder für den Controller und noch weniger für die View benötigt.

Die Mitarbeiter der externen Beratung hatten das Datenbankschema entworfen und damit die Basis für die Models vorbereitet. Dies erforderte sowohl Kenntnisse in der Konzeption von Softwareanwendungen als auch im Bereich Datenbankmodellierung. Durch die parallele Durchführung von Workshop und initialem Aufsetzen konnten die Mitarbeiter direkt am nächsten Tag auf ein vorbereitetes Entwicklungsprojekt mit vorhandenen Models, Controller und einer Datenbank mit der Arbeit beginnen. Während der Einarbeitung der Mitarbeiter in das Framework und die verschiedenen Werkzeuge wurde die Feinabstimmung des Datenbanklayouts auf Basis des Prototyps durchgeführt. Mit der Kommandozeile wurden nun die Controller mit der vollständigen CRUD-Funktionalität erstellt und das Scaffolding entfernt. Auf dieser Basis wurden dann alle Views für die Standardfunktionen erstellt. Hierdurch konnten die Listen- und Detailansichten für die Datensätze und die Navigation innerhalb der Webapplikation an die spezifischen Anwendungsfälle und die Bedürfnisse der Benutzer angepasst werden. Im Laufe der nächsten sechs Wochen erweiterte das Projektteam die Anwendung unter anderem um folgende Funktionalitäten:

- Anpassung der Navigationsstruktur für die einzelnen Benutzergruppen
- Benutzerabhängige Filtermöglichkeiten für die Listenansichten (z. B. alle Mitarbeiter in der Abteilung des angemeldeten Anwenders)
- Verbesserung des Antwortverhaltens der Anwendung durch intuitive Oberfläche mit AJAX, was ein Neuladen der Seite vermeidet
- Zentrale Verwaltung der verwendeten Texte und Beschriftungen (Lokalisierung)
- Nachvollziehbarkeit der Änderungen an Datensätzen mit automatischer Protokollierung des Anlegers und des letzten Bearbeiters (*Who dit it*)
- Berechtigungskonzept auf Basis von Access Control Lists (ACL)
- Einbindung eines Plug-ins für die einfache Administration der ACL
- Anmeldemöglichkeit als beliebiger Anwender (zum Überprüfen der Berechtigungen im Testsystem)

Starthilfe beim ersten Kuchen

Bei der Umsetzung wurden die Mitarbeiter des Unternehmens permanent durch erfahrene Entwickler der externen Beratung begleitet. Diese übernahmen neben dem allgemeinen Projektmanagement vor allem die fachliche Unterstützung des Teams. Durch regelmäßige Reviews des Quellcodes wurde sichergestellt, dass komplizierte Lösungsansätze vermieden wurden und der Code einheitlich gestaltet wurde, sodass er auch langfristig erweiterbar und wartbar bleibt. Die Begleitung durch die externe Beratung hat auch so manchen Entwickler davor bewahrt, tagelang nach einer Lösung für ein bekanntes Problem zu suchen. Die Kombination aus erfahrenen Entwicklern der externen Beratung und hauseigenen Mitarbeitern des Unternehmens hat sich in mehrfacher Hinsicht als Erfolg herausgestellt:

- Direkter Wissenstransfer in das Unternehmen des Auftraggebers
- Einhaltung des straffen Zeitplans durch professionelle Unterstützung
- Niedrige Investitionen durch Einbindung eigener Ressourcen

Gerade die Integration von Mitarbeitern in das Entwicklungsteam, die sonst im täglichen Projektgeschäft tätig sind, hat sich bei der Einführung des Systems in die Organisation ausgezahlt. Ihre Erfahrungen aus dem Tagesgeschäft konnten direkt in den Entwicklungsprozess einfließen und so die allgemeine Akzeptanz des Systems erheblich fördern.

Eine Sache der Perspektive

Die Entwicklung mit CakePHP orientiert sich stark am Datenbanklayout, auf dessen Basis die Models, Controller und Views mit den Standardfunktionalitäten generiert wurden. Dies ist von großem Vorteil, um schnell die Funktionalität der Anwendung und des zugrunde liegenden Konzepts zu validieren. Ist das Datenmodell jedoch eher generisch angelegt, sind die automatisch erzeugten Views besser nur von Fachanwendern zu bedienen, die mit dem Datenkonzept vertraut sind. Im konkreten Projekt ist beispielsweise das Datenmodell so angelegt, dass alle Buchungen in einer Tabelle abgelegt werden. Dabei kann es sich um Buchungen handeln, die bestimmten Mitarbeitern zugeordnet werden (z. B. Urlaubs- und Krankheitszeiten) oder um Buchungen, die bestimmten Auftraggebern zugeordnet werden (z. B. Anzahl Ka-

talogbestellungen). Hierbei wird wiederum zwischen Buchungen in Stückzahl und Buchungen in Stunden unterschieden. Während für Erstere nur ganzzahlige Werte zugelassen werden, dürfen Stunden auch anteilig erfasst werden. Zudem soll ein Mitarbeiter die Möglichkeit haben, mehrere Buchungen (auch inkrementell) gleichzeitig über eine Maske erfassen zu können.

Um diesen Anforderungen gerecht zu werden, ist ein Perspektivenwechsel notwendig. Losgelöst vom Datenmodell wurden im Projekt konkrete Anwendungsfälle für bestimmte Benutzergruppen definiert. Auf Basis einer allgemeinen Beschreibung des abzubildenden Workflows wurden entsprechende Anforderungen an Views und Navigationsstruktur abgeleitet und implementiert. An dieser Stelle wurde die Vorteilhaftigkeit der Schichtentrennung erneut deutlich. So konnten die Entwickler, die sich primär mit der Oberflächengestaltung beschäftigten, das endgültige Design der Anwendungsfälle vornehmen, ohne sich dabei zu sehr vom Datenmodell leiten zu lassen.

Wer war's?

Da in dem im Projekt umgesetzten Buchungssystem abrechnungsrelevante Informationen abgelegt werden, ist es wichtig, Änderungen an Datensätzen auch im Nachhinein nachvollziehen zu können. Dabei kommt es nicht nur auf den Zeitpunkt des Erstellens und der letzten Änderung eines Datensatzes an. Diese Informationen werden von CakePHP automatisch am Datensatz abgelegt, wenn in der zugrunde liegenden Tabelle eine Spalte *created* bzw. *modified* vom Typ *timestamp* existiert. Vielmehr ist die Information darüber, wer die letzte Änderung durchgeführt hat, relevant, um z. B. zu erkennen, dass ein von einem Mitarbeiter eingetragener Datensatz im Nachhinein von einem Teamleiter korrigiert wurde.

Grundlegende Voraussetzung für das Ablegen dieser Information ist, dass sich ein Anwender am System authentifizieren muss. Auch das gehört zu den Standardanforderungen, die sich in Cake sehr komfortabel und schnell abbilden lassen. Als Erstes wird eine Tabelle *users* benötigt, die mindestens die Felder *username* und *password* enthält, in der die Zugangsdaten aller Anwender abgelegt werden. Wie gehabt werden auch für diese Tabelle Model, Controller und View generiert. Als Zweites sind im *UsersController* die Methoden *login()* und *logout()* zu ergänzen:

```
class UsersController extends AppController {

    var $name = 'Users';

    function login() {
    }

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

Das Abarbeiten der Methode *login()* überlassen wir der Magie von CakePHP. Als Drittes wird eine einfache View für den *UsersController* mit Namen *login.ctp* benötigt, über die sich der Anwender am System authentifiziert:

```
<?php
    echo $session->flash('auth');
    echo $form->create('User', array('action' => 'login'));
    echo $form->input('username');
    echo $form->input('password');
    echo $form->end('Login');
?>
```

Zur Verschlüsselung des Passworts verwendet CakePHP im Standard einen Hash-Algorithmus (SHA1). Dieser wird nicht nur auf das zur Authentifizierung eingegebene Passwort angewandt, sondern ebenso bei der Anlage oder Veränderung eines Benutzerdatensatzes, ohne dass dies explizit im Code erwähnt werden müsste. Zuletzt wird durch den Befehl

```
class AppController extends Controller {
    var $components = array('Auth');
}
```

die Authentifizierungskomponente von CakePHP im AppController bekannt gemacht und so über den Vererbungsmechanismus auf alle anderen Controller übertragen. Sofort wird der Zugriff auf alle Seiten der Anwendung für nicht authentifizierte Anwender gesperrt. Durch eine zusätzliche Zeile in der Callback-Methode *beforeFilter()* des entsprechenden Controllers können gezielt bestimmte Views wieder für die Öffentlichkeit freigegeben werden:

```
function beforeFilter() {
  $this->Auth->allow('index','view');
}
```

Nachdem nun also bekannt ist, welcher Anwender gerade mit dem System arbeitet, muss diese Information nur noch in den neu erstellten oder veränderten Datensätzen gespeichert werden. Hierzu werden in den entsprechenden Tabellen die Spalten *modified_by* und *created_by* angelegt, die einen Primärschlüssel der Tabelle *Users* referenzieren sollen. Eine naheliegende Idee wäre jetzt, die von CakePHP in der Session abgespeicherten Anwenderinformationen immer beim Speichern eines Datensatzes aus der Session zu holen und mit abzuspeichern. Im Prinzip ist das auch eine gute Idee. Die Frage ist nur, wo der entsprechende Code dazu abgelegt wird. CakePHP bietet hier das Konzept der so genannten „Behaviors" an. In solch einem frei definierbaren Behavior kann das Verhalten eines Models an zentraler Stelle definiert werden. Für unseren Zweck verwenden wir die von Daniel Vecchiato unter [1] bereitgestellte Klasse *WhoDidItBehavior*, deren Quellcode im Ordner */mo-*

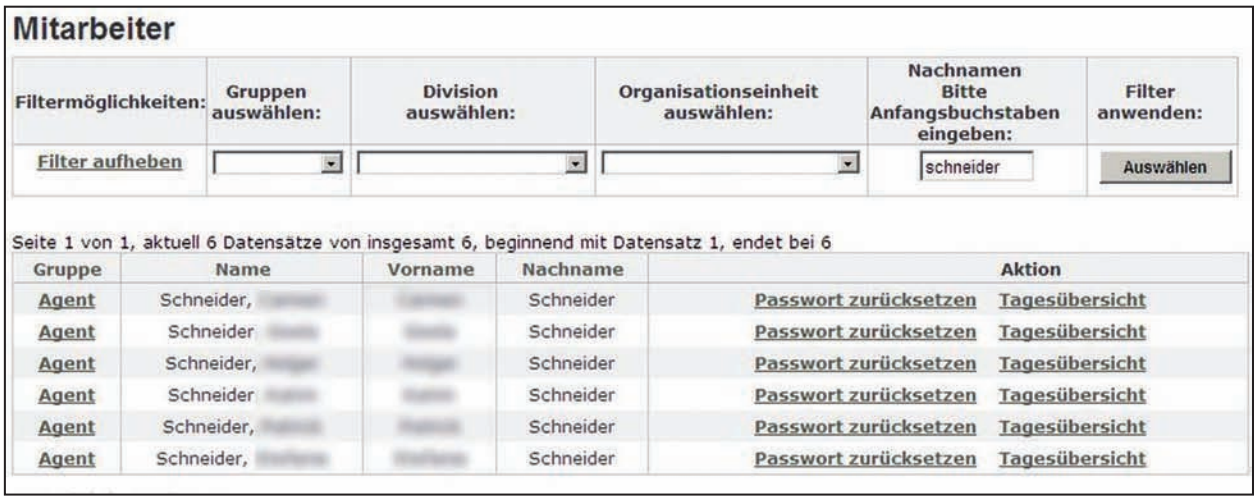

Abb. 3: Listenansicht mit aktiviertem Filter auf den Nachnamen „Schneider“

dels/behaviors/ des Cake-Projekts abgelegt wird. Nun muss CakePHP noch mitgeteilt werden, welche Models sich entsprechend dem *WhoDidItBehavior* zu verhalten haben. Dazu wird in den Models folgende Zeile eingefügt: *var $actsAs = array('WhoDidIt');*. Und schon bei der nächsten Änderung eines Datensatzes wird die Information, wer diesen Datensatz geändert hat, in dem zuvor angelegten Feld *modified_by* abgespeichert.

Gemäß des Prinzips Convention over Configuration (CoC) funktioniert diese Funktionalität (wie viele andere) in CakePHP out of the box, wenn man sich an die vorgegebenen Konventionen für die Benennung von Tabellen und Spalten etc. hält. Auch wenn es immer eine Möglichkeit gibt, diese Konventionen zu überlagern, sollte man sich bereits beim Design des Datenbankmodells mit diesen Konventionen auseinandersetzen und sie so weit wie möglich einhalten.

Kleinere Portion gefällig?

Einfache Listen zur Ansicht der Datensätze sind mit CakePHP schnell erstellt. Doch nicht immer sollen alle Datensätze, die in einer Tabelle gespeichert sind, auch angezeigt werden. Die Suche eines bestimmten Mitarbeiters in einer Liste von etwa 900 Mitarbeitern wird auch bei alphabetischer Sortierung recht mühsam. Zudem sollen die Teamleiter in der Listenansicht schnellen Zugriff auf genau die Mitarbeiter haben, die zu ihrem Team gehören. Also müssen die aus der Datenbank geladenen Datensätze für diese Zwecke gefiltert werden. Dies wird mit CakePHP keinesfalls mit dem Schreiben von entsprechenden SQL-Queries realisiert, was den Austausch der genutzten Datenbank (MySQL, Oracle, SQLServer etc.) sehr aufwändig oder unmöglich machen würde. Stattdessen bietet CakePHP die universelle Controller-Methode *find()* an, der zwei Parameter übergeben werden. Als Erstes wird der Typ der Suche erwartet. Die am häufigsten genutzten sind *all* und *list*. Während *all* alle gefundenen Datensätze samt der assoziierten Datensätze zurückgibt, liefert *list* ein indiziertes Array mit den Anzeigenamen, das sich besonders zur Darstellung von Auswahlboxen eignet.

Über das zweite Argument kann die Abfrage nahezu beliebig konfiguriert werden. Eine einfache Textsuche nach allen Anwendern mit einem bestimmten Namen zeigt das folgende Beispiel aus dem *UsersController*:

```
function index() {
  $searchstring = $this->data['User']['searchstring'];
  $conditions = array('conditions' => array('User.name' => $searchstring));
  $this->set('users', $this->User->find('all', $conditions));
}
```

In der ersten Zeile der Methode *index()* wird die Eingabe des Benutzers aus dem Feld *searchstring* ausgelesen, das in der View */users/index.ctp* als einfaches Eingabefeld definiert wurde. Als Nächstes wird die Abfragebedingung bestimmt, durch die die folgende Suche auf alle Datensätze eingeschränkt wird, bei denen das Feld *name* in der Tabelle *User* dem vorgegebenen String entspricht. Die dritte und letzte Zeile führt letztlich die Datenbankabfrage aus und übergibt das Ergebnis an die Variable *users*, die anschließend an die View *index* übergeben wird. Um in der Suche Wildcards zu ermöglichen, genügt eine kleine Modifikation der zweiten Zeile in der Controller-Methode *index()*:

```
$conditions = array('conditions' => array('User.name LIKE' =>
                                          $searchstring));
```

Um nun dem Teamleiter nur die Mitarbeiter anzuzeigen, die zu seinem Team gehören, wird die Abfrage nach den Mitarbeitern entsprechend eingeschränkt. Die Information, zu welchem Team ein Anwender gehört, wird dank CakePHP automatisch bei der Authentifizierung eines Anwenders in der Session gespeichert und kann jederzeit ausgelesen werden. Die Abfragebedingung sieht dann wie folgt aus:

```
$conditions = array('conditions' => array('User.team_id' =>
                                  $this->Session->read('Auth.User.team_id')));
```

Neben der hier gezeigten Suche nach bestimmten Datensätzen kann eine Abfrage mit der Methode *find()*

durch weitere Parameter mit den üblichen Operationen wie Gruppierung, Sortierung oder Limitierung auf eine bestimmte Anzahl von Datensätzen manipuliert werden. Auch kann explizit angegeben werden, welche Felder in der Ergebnismenge enthalten sein sollen, was insbesondere bei breiten Tabellen die Ergebnismenge erheblich reduzieren kann. Insbesondere in der Entwicklungsphase ist es sehr hilfreich, CakePHP im Debug-Modus auszuführen (Einstellung in */core/config.php*). Dann werden auf dem Bildschirm im unteren Bereich die von CakePHP generierten SQL-Statements angezeigt.

Du kommst hier nicht rein!

In der Anwendung werden aktuell vier unterschiedliche Gruppen vom Agenten bis zum Administrator unterschieden. Jeder Mitarbeiter erhält abhängig von seiner Gruppe den Zugriff auf ihm zugeordnete Funktionen wie die Erfassung einer Buchung (Agent) oder Anlage eines Benutzerkontos (Administrator). Vor dem Zugriff auf das Programm erfolgt im ersten Schritt eine Authentifizierung durch Eingabe des Benutzernamens und Passworts. Nach erfolgreicher Anmeldung wird für diese Sitzung des Anwenders seine zugeordnete Gruppe ermittelt und er kann seine Aufgaben wahrnehmen. Für diese Umsetzung stellt CakePHP geeignete Komponenten zur Verfügung. Komponenten sind eine Sammlung von Funktionen für allgemeine Aufgaben wie Handhabung von Sessions oder E-Mail.

Für die Authentifizierung nutzen wir, wie oben beschrieben, die *AuthComponent*, mit der der Benutzer Zutritt zur Anwendung erhält. Damit wissen wir, wer jemand ist (Authorisierung) und ergänzen diese Funktionalität, um zu steuern, was jemand darf. Dafür ergänzen wir im *AppController* noch das Attribut *$components* um den Eintrag '*Acl*'. Mit dieser zweiten Komponente werden die mächtigen Zugangskontrolllisten (Access Control List, ACL) aktiviert. Diese Eigenschaft des AppControllers wird wieder auf alle anderen abgeleiteten Controller vererbt:

```
class AppController extends Controller {

  var $components = array('Auth', 'Acl');
  function beforeFilter () {
    //Configure AuthComponent
   $this->Auth->authorize = 'actions';
   $this->Auth->loginAction = array('controller' => 'users', 'action' => 'login');
   $this->Auth->logoutRedirect = array('controller' => 'users', 'action' =>
                                                                    'login');

}
}
```

Prinzipiell ganz einfach, kann für die Einarbeitung in dieses Themenfeld schnell die eine oder andere Stunde vergehen – die sich aber aufgrund der späteren Flexibilität lohnt. So ermuntert auch die Onlinedokumentation [2]:

Abb. 4: Verwaltung der Controller und ihrer Funktionen (in Klammern die Angabe zur Anzahl der Funktionen)

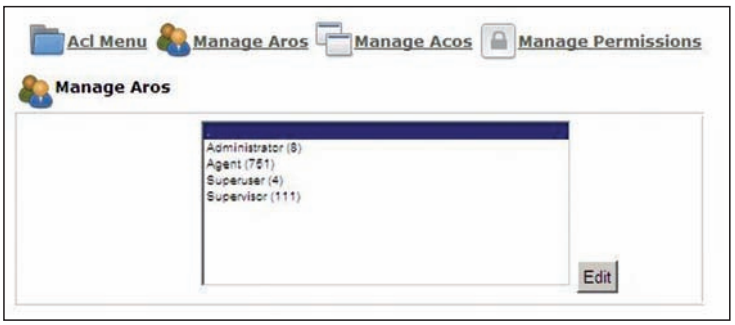

Abb. 5: Verwaltung der Gruppen und ihrer Benutzer (in Klammern die Angabe zur Anzahl der Benutzer)

„Bleib' tapfer und halte durch, auch wenn's sicherlich noch hart für Dich wird. Sobald Du es einmal verstanden hast, sind ACLs extrem hilfreiche Werkzeuge, um die Kontrolle über Deine Anwendungen und deren Entwicklung zu behalten."

Zugangskontrolllisten stellen eine erprobte Technik dar, Zugriffe auf Daten und Funktionen eingrenzen zu können. Berechtigungen für Anwendungen können sehr granular verwaltet werden und bleiben gleichzeitig leicht wartbar. In dem flexiblen Model der ACL werden generell nur zwei Objekte unterschieden:

- *ACO* (Access Control Object): Etwas, das angefragt wird
- *ARO* (Access Request Object): Etwas, das etwas anderes anfragt

Um gleich die Hürde der Abstraktion zu nehmen, gehen wir auf unser Beispielprojekt ein. Ausgewählte Funktionalitäten (Controller) der Anwendung (ACO) sollen bestimmten Benutzergruppen (ARO) zur Verfügung gestellt werden. Hierfür wurde mittels

```
$this->Acl->Aco->create(array('parent_id' => null, 'alias' => 'controllers'));
$this->Acl->Aco->save();
```

auf Hauptebene ein ACO *controllers* angelegt. Alle Controller und deren Funktionen wurden nun unterhalb dieses ersten ACO gespeichert. In unserem Beispiel ergibt sich folgende Struktur:

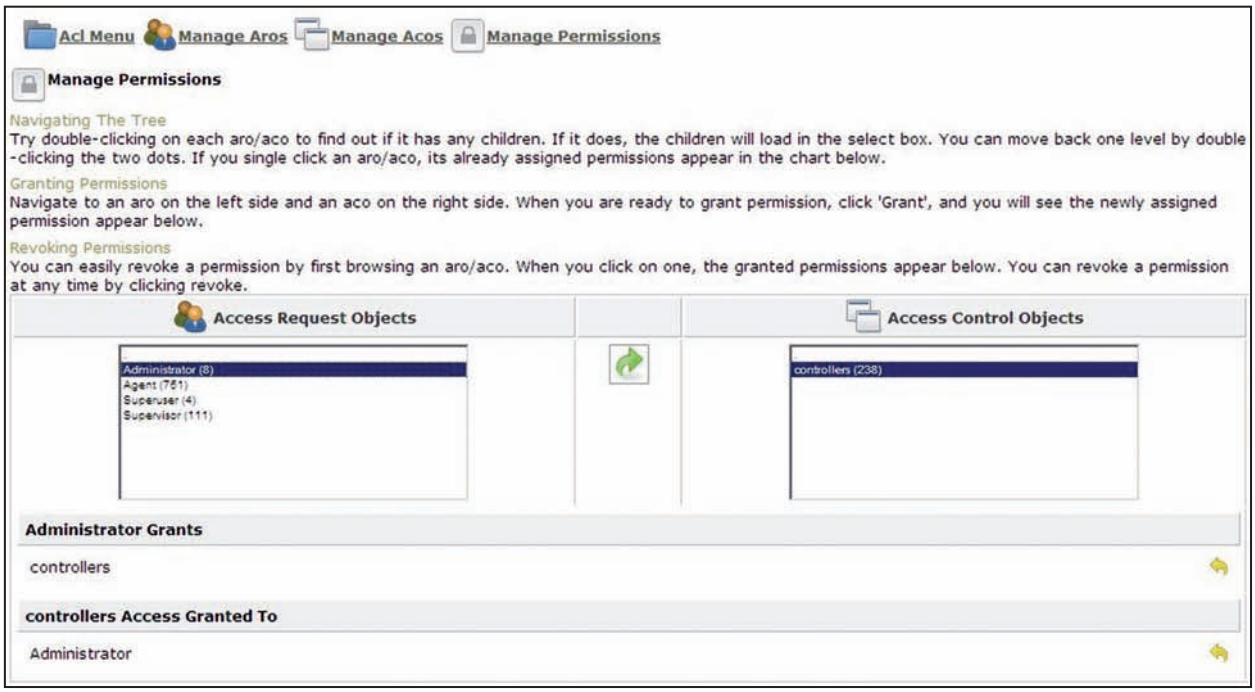

Abb. 6: Einfache Verwaltung der Berechtigungen mit dem ACL-Plug-in: Die Gruppe der Administratoren besitzt Vollzugriff auf alle Controller und deren Funktionen

- controllers
  - AccountRegulations
    - index
    - view
    - add
    - edit
    - delete
  - Customers
    - …

In vergleichbarer Weise wurden die AROs gefüllt:

- Agent (Group)
  - Username1 (User)
  - Username2 (User)
  - …
- Supervisor (Group)
- Superuser (Group)
- Administrator (Group)

In Klammern befindet sich die Angabe, welchem Model der Datensatz zugeordnet ist. Es wäre auch möglich, Gruppen unter Gruppen einzuhängen oder gänzlich andere Modelle wie Kunden oder Produkte zu verwalten. Die genaue Einrichtung der Authentifizierung und der Zugangskontrolllisten kann unter [3], [4] und in dem Tutorial unter [5] nachgelesen werden.

Sowohl ACOs als auch AROs werden in sehr effizienten Baumstrukturen (verschachtelte Mengen, Nested Sets, [6]) gespeichert. Diese Logik kann auch in eigenen Controllern genutzt werden, um z. B. hierarchische Produktgruppen abzubilden. Die auf diese Weise gespeicherten ACO und ARO werden über eine Verknüpfungstabelle in Verbindung zueinander gesetzt, d. h. der Zugriff auf Funktionen wird erlaubt oder entzogen. Um die Übersicht bei der Verwaltung von drei Tabellen zu wahren, wurde auf ein Plug-in zurückgegriffen. Plug-ins sind eine Kombination aus mehreren Controllern, Models und Views. So ist diese Sammlung von Skripten wiederverwendbar und kann auf einfache Weise in andere Anwendungen integriert werden. Und genauso ein Plug-in hat Jeff Loiselle für die Verwaltung der Zugangskontrolllisten erstellt, das CakePHP ACL Management Plug-in [7]. Es nutzt AJAX, um das Arbeiten mit den Baumstrukturen einfacher zu gestalten und ständiges Neuladen der gesamten Seite zu vermeiden. Das Plug-in kann über github heruntergeladen werden [8] (etwas versteckt ist oben ein Download Source-Button zu finden) und wird dann vollständig im Verzeichnis */plugins/* entpackt. Entgegen der älteren Anleitung [9] sind sowohl die notwendigen JavaScript-Bibliotheken von Prototype (AJAX-Funktionalität) als auch die Icon-Bibliothek Tango bereits im Unterverzeichnis */vendors/* enthalten. Nach einer Aktivierung des Routings für *admin* in der Konfigurationsdatei */config/core.php* konnte das Plug-in über http://yourserver/admin/acl/ aufgerufen werden: *Configure::write('Routing.admin', 'admin');*.

Im Firefox lief das Plug-in nun sofort problemlos, lediglich für den Internet Explorer musste eine aktuellere Datei von Prototype [10] eingespielt werden.

Danach wurde über diesen Administrationsbereich der Gruppe der Administratoren der Zugriff auf den obersten Controller-Eintrag (Root) zugeordnet. Dies entspricht einem Vollzugriff auch auf jeden darunter liegenden Controller und dessen Funktionen. Alle anderen Gruppen erhielten nur Zugriff auf ausgewählte Controller bzw. Funktionen. Es wäre auch umgekehrt möglich gewesen, vergleichbar mit den Administratoren einen „Vollzugriff“ einzurichten und dezidiert die Berechtigungen für einige Controller oder auch nur

einzelne Funktionen zu entziehen. Hier wird deutlich, wie auch ein mächtiges Berechtigungssystem übersichtlich verwaltet werden kann.

Über einige zusätzliche Einstellungen in den Controllern für User und Gruppen werden beim Anlegen neuer Benutzer automatisch die entsprechenden Einträge in der Tabelle *ARO* vorgenommen. Da das Zugriffssystem auf den Gruppenberechtigungen aufsetzt, ist kein zusätzlicher Eintrag für jeden Benutzer erforderlich, und neue Mitarbeiter können nach der Anlage direkt in der Anwendung arbeiten. CakePHP nimmt mit den beiden Komponenten Authentifizierung (*AuthComponent*) und Zugangskontrolllisten (*AclComponent*) viel Arbeit ab, der Programmierer kann sich so auf die Einrichtung der Berechtigungen konzentrieren.

Fast Food ohne Bauchschmerzen und schlechtes Gewissen

Im beschriebenen Projekt konnte CakePHP seine Potenziale als Rapid-Application-Development-Framework voll ausspielen. In kurzer Zeit entstand eine leistungsfähige Anwendung, die durch das MVC-Konzept und die Abstraktion zur Datenbank (ORM) einen schlanken, langfristig wartbaren Programmcode ermöglicht. Das Framework gibt hierfür eine Struktur vor, ohne Einbußen in der Flexibilität in Kauf nehmen zu müssen. Mit dem Werkzeug CakePHP wird der Programmierer von Routineaufgaben entlastet und kann sich voll und ganz auf die individuellen Ausprägungen der zu erstellenden Anwendung konzentrieren. Auf diese Weise wird die Erstellung von Webanwendungen in PHP ein Kinderspiel, halt ein „piece of cake“.

Fazit

Für die schnelle und kostengünstige Erstellung von Anwendungen benötigt man zwingend geeignete und erprobte Werkzeuge wie das Framework CakePHP und eine Entwicklungsumgebung wie Eclipse mit angeschlossenem Quellcodeverwaltungssystem. Aber ohne ein funktionierendes Team und einen Auftraggeber, der bei aufkommenden Fragen direkt Entscheidungen treffen kann, sind diese technischen Hilfsmittel wertlos. Treffen aber engagierte Teamplayer auf geeignete Werkzeuge und werden kurze Abstimmungswege eingehalten, so entwickelt sich – mit Unterstützung durch erfahrene Entwickler und Projektmanager – eine agile Vorgehensweise mit schnellen Fortschritten in der Entwicklung. Nur auf diese Weise war es in unserem Projekt möglich, innerhalb von nur sechs Wochen eine komplexe Anwendung zu erstellen, die allen Benutzern einen Mehrwert liefert und langfristig im Sinne des Unternehmens weiter entwickelt werden kann.

Ralf Hohoff ist seit mehr als zehn Jahren in der Informationstechnik und dem Projektmanagement tätig. Als technischer Berater der buw consulting GmbH aus Osnabrück unterstützt er Kunden bei der Konzeption und Implementierung von IT-Systemen mit den Schwerpunkten CRM und Wissensmanagement.

Thomas Recke unterstützt als erfahrener Experte im Bereich Business Intelligence Kunden bei der Konzeption und Umsetzung von BI-Projekten.

Links & Literatur

[1] http://bakery.cakephp.org/articles/view/whodidit-behavior-automagic-created_by-and-modified_by-fields

[2] http://book.cakephp.org/de/view/171/Access-Control-Lists

[3] http://book.cakephp.org/de/view/172/Authentication

[4] http://book.cakephp.org/de/view/171/Access-Control-Lists

[5] http://book.cakephp.org/de/view/641/Simple-Acl-controlled-Application

[6] http://de.wikipedia.org/wiki/Nested_Sets

[7] http://jeff.loiselles.com/?p=12

[8] http://github.com/phishy/acl_plugin/tree/master

[9] http://bakery.cakephp.org/articles/view/acl-management-plugin

[10] http://www.prototypejs.org/