Axel Tüting
Christiane Maier-Stadtherr
René Serradeil

Know-how ist blau.

Joomla!-Extensions entwickeln

Eigene Komponenten, Module und Plugins programmieren

> Erweiterungen für das freie Content-Management-System

> Schritt für Schritt von der Idee bis zum Installer

> Die Joomla-API kennenlernen

FRANZIS

Schnelle Erfolge mit eigenen Erweiterungen!

Axel Tüting / Christiane Maier-Stadtherr / René Serradeil

Joomla!-Extensions entwickeln

Axel Tüting
Christiane Maier-Stadtherr
René Serradeil

Joomla!-Extensions entwickeln

Eigene Komponenten, Module und Plugins programmieren

Mit 104 Abbildungen

Bibliografische Information der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie;
detaillierte Daten sind im Internet über http://dnb.ddb.de abrufbar.

Lektorat: Anton Schmid
Satz: DTP-Satz A. Kugge, München
art & design: www.ideehoch2.de
Druck: GGP Media GmbH, Pößneck
Printed in Germany

ISBN 978-3-645-60134-4

Vorwort

Wenn Sie dieses Buch in die Hand nehmen, werden Sie wissen was Sie tun. Wir sparen uns deshalb das übliche »Worum geht es in diesem Buch«. Allerdings müssen wir dennoch schreiben, dass dieses Buch nicht für Menschen gedacht ist, die Programmierung lernen wollen. Wir setzen voraus, dass Sie zumindest weitreichende Grundlagen der PHP-Programmierung besitzen. Auch setzen wir voraus, dass Sie Joomla soweit beherrschen, dass Sie es fehlerfrei installieren können und sich einigermaßen im Backend zurechtfinden. Kenntnisse über SQL und Datenbanken sind wünschenswert. Wir helfen Ihnen bei der Objektorientierten Programmierung – es kann jedoch nicht schaden, wenn Sie bereits ein paar Grundlagen mitbringen.

Als wir anfingen, dieses Buch zu schreiben, lag Joomla noch in der Version 1.6 vor. Eine Dokumentation gab es seinerzeit nicht, also suchten und entdeckten wir, probierten aus und versuchten das neue Framework zu verstehen. Dann kam Joomla! 1.7 heraus, das *Framework* hieß nun *Platform* und Teile unseres Codes funktionierten plötzlich nicht mehr. Also mussten wir nicht nur umprogrammieren, sondern vor allem auch verstehen, warum das so ist. Eine Dokumentation gab es erst zum Ende der Versionsreihe. Dann kam Joomla 2.5 und abermals funktionierten Teile unseres Codes nicht. Oder aber, es stellte sich nun heraus, dass mitten in unserer Entwicklung einige Funktionen *deprecated* wurden, also als veraltet galten und nicht mehr in zukünftigen Entwicklungen Berücksichtigung finden werden. Also auch da mussten wir einiges umprogrammieren, denn wir wollten Ihnen natürlich nichts empfehlen, was nicht weiterentwickelt wird. Dazu kam und kommt eine Dokumentation, die an vielen Stellen unzureichend ist. Oft standen wir fasziniert vor Wörtern wie »unknown use«, die sich an einigen Stellen in der offziellen Joomla-Dokumentation finden, oder konnten Funktionen einfach garnicht auffinden. Zum Beispiel gibt es bislang noch immer keine komplette Liste der Ereignisse, die in Plugins benutzt werden können. In diesem Buch finden Sie allerdings eine solche Liste. Zumindest glauben wir, dass sie einigermaßen vollständig ist.

Durch diese ganzen Schwierigkeiten und die Tatsache, dass so ein Buch ja auch nicht mal eben so nebenbei geschrieben wird, zog sich die Veröffentlichung viele Monate hin.

Nicht zuletzt deshalb einen großen Dank an den Franzis Verlag, der stets geduldig blieb und ganz besonderen Dank an unseren Lektor Anton Schmid, dem wir das Leben phasenweise wahrlich nicht leicht gemacht haben, der jedoch stets hinter uns stand!

Als wir mit dem Buch anfingen, hatten wir eine ganze Schar Testleser. Je länger das Projekt dauerte, je kleiner wurde diese Schar. Dennoch euch allen Dank fürs kritische Lesen und so manchen Tipp oder auch einfach für eure Motivation!

Einen jedoch müssen wir ganz besonders erwähnen: Christian Linsner. Was dieser Mann alles gelesen hat ist an sich schon unglaublich. Das alles stets kritisch, mit

Adleraugen, Unzulänglichkeiten, Fehler und andere Ungereimtheiten findend und dennoch stets konstruktiv kritisierend – einen ganz besonderen lieben Dank an Dich, Christian!

Aufbau des Buches

Der erste Teil hat das Ziel, dass Sie schnell und unkompliziert erste Erfolge und schnelle Lernerfolge haben. Sie erhalten hier jedoch auch die Grundlagen, auf die das Buch im weiteren Verlauf aufbauen wird.

Der zweite Teil vertieft dann eben dieses. Die Komponentenentwicklung wird stetig komplexer, Module und Plugins zeigen, was möglich ist, und die Joomla-API wird natürlich auch genauer betrachtet. Ein bunter Mix aus Theorie und Praxis.

Sie können das Buch von vorn bis hinten durcharbeiten und unserem Aufbau folgen. Sie können sich aber ebenso gut querbeet durch das Buch arbeiten und lesen, was Ihnen gefällt. Wir haben meistens bei den Kapiteln vorangestellt, was Sie im jeweiligen Kapitel finden und auch jeweils auf die Downloads verwiesen, die das Buch begleiten.

Sie können mit diesem Buch lernen, wie Sie eigene Erweiterungen in Joomla erstellen können. Wir haben viele Fragen aufgegriffen, können aber natürlich nicht alles beantworten. Wir waren jedoch bemüht, Ihnen das nötige Rüstzeug zu geben, um letztlich eigene Erweiterungen zu entwickeln. Das beinhaltet auch manchmal »Hilfe zur Selbsthilfe«.

Inhaltsverzeichnis

2 Joomla-Grundwortschatz

Das Joomla! Content Management System (CMS) ist eine recht umfangreiche Anwendung. Das Wissen über den Einsatz und die Bedienung des CMS setzen wir voraus. Wir schauen jetzt aus der Sicht des Entwicklers von Erweiterungen auf diese zahlreichen Verzeichnisse und Dateien.

Im Gesamtsystem Joomla! steckt nicht nur viel praktisches Wissen sondern jede Menge Softwaretheorie. Nun gibt es zwar nichts Praktischeres als eine gute Theorie und auch wenn wir nicht die Absicht haben, Sie mit allen Details dieser Theorien zu belästigen, so möchten wir zu Anfang ein paar wichtige Begriffe abklopfen und ihre Bedeutung klären.

2.1 Internetjargon

Bevor wir uns der Terminologie in und um Joomla widmen, klären wir zuerst ein paar allgemeine Begriffe[5] rund um »Internetseiten« von denen einige immer wieder gerne verwechselt und fälschlich synonym verwendet werden:

1. Domain: Die allgemeine Definition bezeichnet »einen (geografischen) Bereich der unter der Kontrolle einer einzelnen Person oder Organisation steht«. Im World Wide Web entspricht dies »einem Namensbereich, der dazu dient, Computer im Internet zu identifizieren; er ist unter anderem Bestandteil der URL einer Webseite«. Vorzugsweise haben Sie ebenfalls die Kontrolle über Ihr virtuelles Hoheitsgebiet im Internet, um darin bspw. auch *Subdomains* anzulegen und zu verwalten.
2. URL: Abkürzung für *Uniform Resource Locator*, ist die eindeutige Bezeichnung für eine Ressource in einem Computernetzwerk wie dem Internet. Diese *lokalisierbaren Ressourcen* sind dort unter anderem die einzelnen *Webseiten* einer *Website.*
3. Webseite: Ein einzelnes HTML-Dokument, das über eine URL adressiert und mitsamt den darin verknüpften Bildern und Mediendateien z. B. in einem Web-Browser angezeigt werden kann. In Joomla! kann mit Hilfe von Menüeinträgen das Erscheinungsbild einer oder mehrerer Webseiten sowie die Elemente, welche darin angezeigt werden sollen, entscheidend mitbeeinflusst werden. Hierzu dienen zahlreiche Parameterebenen mit denen direkt und indirekt das eigentliche HTML-Seitentemplate, das Layout einer Komponente und die Anzahl und Reihenfolge von Modulen festgelegt werden.

---

[5] Begriffsklärungen frei nach *http://en.wiktionary.org* und *http://en.wikipedia.org*

4. Site: englisch u. a. für Standort, Sitz, Niederlassung. Eine *Website* stellt den funktionalen Teil einer Domain im WWW dar und für die Besucher die Gesamtheit aller darüber verfügbaren öffentlichen Ressourcen bereit. Im allgemeinen Sprachgebrauch werden *Internet* und *Web* synonym verwendet, weshalb gerade im Deutschen aufgrund des gleichen Wortklangs von »Seite« und »Site« die Begriffe Internet*seite* und Web*site* gleichbedeutend verwendet werden. In der Marketingsprache wird die Website zur *Internetpräsenz* oder zum *Webauftritt*[6].
5. Root: englisch für *Wurzel*, bezeichnet in einem Dateisystem allgemein die (hierarchische) oberste Ebene. Im Kontext einer Anwendung verwendet man die »denglischen« Begriffe Root-Ordner/-Verzeichnis ebenso wie Stamm- und Hauptverzeichnis synonym für das Verzeichnis in welchem der ausführbare Teil der Anwendung installiert ist. Bei Web-Anwendungen in PHP ist es der Ordner in dem die Datei *index.php* liegt.

2.2 CMS – Backend und Frontend

Das Joomla! CMS enthält bei der Auslieferung drei ablauffähige Web-Anwendungen:

- Den *Installer* für die Installation des CMS über das Verzeichnis */installation*.
- Das *Backend* für die Administration des CMS im Verzeichnis */administrator*.
- Das *Frontend*, das den öffentlichen Teil der Website repräsentiert.

Der Installer wird üblicherweise mitsamt Ordner gelöscht, nachdem er seine Aufgabe erfüllt hat. Somit verbleiben Backend und Frontend, zwei (nahezu) eigenständige Web-Anwendungen, die mit verschiedenen URLs aufgerufen werden.

Alternativ werden auch gerne die Begriffe *Admin* und *Administrator* für das Backend und *Site* für das Frontend verwendet. Diese Bezeichnungen finden sich auch in unterschiedlicher Schreibweise im PHP-Quellcode wieder, wie Sie im späteren Verlauf des Buches noch sehen werden.

Backend und Frontend haben ihre eigenen Komponenten, Module, Templates und Sprachdateien. Das Frontend ist in seiner Gesamtheit nicht so ganz eigenständig und einige Erweiterungen bedienen sich immer wieder der Programmdateien aus dem Backend.

Zusätzlich zu den drei Web-Anwendungen die im Browser aufgerufen werden, befinden sich im Ordner */cli* drei weitere Anwendungen für den Einsatz auf der Kommandozeile (CLI = command line interface) mit denen ein Administrator des CMS Wartungsaufgaben der Volltextsuche (com_finder) automatisieren kann.

---

[6] Insofern korrekt, da viele derart konzipierte Auftritte und Präsenzen zwar die Auftretenden präsentieren, selten genug aber von praktischem Nutzen für Besucher sind.

Joomla! besteht somit aus sechs PHP-Anwendungen, die zusammen das Content Management System bilden und von denen Sie die beiden wichtigsten Anwendungen, Backend und Frontend, um neue Funktionen erweitern können.

2.3 Zwei Hüte, ein Kopf: Platform vs. Framework

Eine Plattform bezeichnet ein einheitliches Basissystem auf dem Software ausgeführt wird. Programme werden i. d. R. für eine bestimmte Plattform geschrieben. Das Betriebssystem auf Ihrem Rechner oder Smartphone ist solch eine Plattform. Verschiedene Programmiersprachen sorgen über mehr oder minder ausgeklügelte Konstrukte dafür, dass damit geschriebene Programme auch mehr oder minder erfolgreich plattformübergreifend genutzt werden können. Java ist hierbei im Desktop- und Serverbereich verhältnismäßig erfolgreich und so programmieren Menschen unterschiedliche Software für die »Java-Plattform«.

Ein Framework ist ein Gerüst oder Rahmen, mit dessen Hilfe das Programmieren von Software vereinfacht und/oder vereinheitlicht wird. Doch so wenig wie ein Gerüst das fertige Haus oder ein Rahmen das fertige Gemälde darstellt, so wenig ist ein (Software-) Framework ein lauffähiges Programm. Es enthält jedoch eine Menge Bausteine, die Sie als Entwickler verwenden können, um die immer wiederkehrenden Aufgaben in den eigenen Programmen einfacher und schneller umsetzen zu können. Wie jedes Baukastensystem bestimmt das eingesetzte Framework die grundlegende Form der erstellten Programme, will sagen: Ein Gebilde aus Lego-Bausteinen folgt den mechanischen Bedingungen dieser Steine und wird auch immer als Legobausteingebilde zu erkennen sein.

In Joomla! 1.5 wurde ein solches Framework eingeführt, um damit Erweiterungen für das CMS zu programmieren. Seit der Joomla!-CMS-Version 1.7 heißt dieses Framework *Joomla! Platform*. Der Markenname *Joomla!* soll zukünftig nicht mehr nur mit dem CMS und Webseiten alleine in Verbindung gebracht werden, sondern *Joomla!* soll alle nur erdenklichen Arten an PHP-Anwendungen ermöglichen und eventuell auch vereinen, deren gemeinsame Grundlage eben diese *Joomla! Platform* sein wird.

Die *Platform* (und hier auch absichtlich mit einen »t« geschrieben) ist dann auch als ein eigenständiges *Produkt* der Marke *Joomla!* ausgelegt, mit eigener Versionszählung und eigenem Lebenszyklus. Updates der *Platform* erscheinen in einem festen Turnus, etwa alle drei Monate, sodass sich die Versionsnummer mehrmals pro Jahr ändern wird: 11.4, 12.1, 12.2 etc. Diese Versionszählung weicht hier von der semantischen[7], dreistelligen Versionierung ab. Ebenso wie die seit ca. 2010 zu beobachtenden inflationären Versionssprünge bei Web-Browsern liefert die Höhe der Versionsnummer der Joomla Platform kaum mehr einen Hinweis auf die Möglichkeiten und Funktionen. Versionsnummern bekommen mehr und mehr den Charakter eines Verfallsdatums.

---

7 *http://semver.org/* Semantic Versioning Specification

Joomla! CMS Version 2.5.x enthält und verwendet die **Joomla! Platform Version 11.4** (Jahr 2011, Release 4). An dieser Konstellation wird sich auch nichts ändern. Joomla! 3.0.0 wird vermutlich eventuell eine frischere Version der Platform enthalten.

Wenn Sie und wir Ende 2012 mit dem Gedanken spielen, Erweiterungen für Joomla! *3.x* zu programmieren, ist für uns als Entwickler die dann mitgelieferte Version der Platform mitunter wichtiger, als die Versionsnummer des CMS. Für den Anwender dagegen ist es genau umgekehrt. Welche Vor- oder Nachteile diese Trennung der Entwicklungsstränge auf die Kompatibilität und Interoperabilität von CMS-Erweiterungen und Platform-Applikationen haben wird, muss sich erst noch zeigen.

Die Joomla!-Platform ist *aus* dem Joomla!-CMS und dessen technischen Anforderungen heraus entstanden und hat aufgrund dieser gemeinsamen Historie *noch* eine recht enge funktionale Verbindung mit dem CMS. Dies wird sich mit der fortschreitenden, *unabhängigen* Weiterentwicklung der *Platform* zunehmend ändern und die Unabhängigkeit zum *CMS* und dessen was ein CMS an Funktionalität braucht, wird von Quartal zu Quartal größer.

Was Sie heute in der Kombination CMS 2.5/Platform 11.4 noch als Bestandteil der Platform identifizieren *würden*, kann im CMS 3.x/Platform 12.y gänzlich andere Strukturen haben, nicht mehr existieren, anders heißen, oder, oder. Im Kapitel 11 werden Sie einen ersten Eindruck dieses leicht chaotischen Übergangs kennenlernen, der auch uns beim Schreiben dieses Buches immer wieder überrascht hat.

Insgesamt erlaubt dieses Konzept aber der Joomla!-Platform (also dem PHP-Framework) die schon längst überfälligen neue Wege zu gehen, ohne auf das CMS zu warten oder sich im Funktionsumfang nur auf die Anforderungen von Web-Anwendungen zu beschränken. Das CMS-Team *kann* die technischen Neuerungen der Platform ganz oder teilweise in zukünftige Versionen einbauen. Sie als Entwickler müssen nun jedoch genauer darauf achten, ob sie Erweiterungen *für* eine bestimmte Version des *Joomla! CMS* programmieren oder PHP-Anwendungen *mit* einer bestimmte Version der *Joomla! Platform*.

Der Zweck eines Framework ist abgesteckt, egal wie sein Produktname lautet. Daher werden wir in den Programmier-Kapiteln dessen technische Aspekte beschreiben und weniger auf den Produktnamen »Platform« eingehen. Für die Zwecke, die wir hier verfolgen, sind die feinen Unterschiede auch nicht von Bedeutung. Im Zweifelsfall ist es dann auch einfach Joomla.

2.4 Finger weg vom Core

Der Core ist das Kernstück eines Systems. In den verschiedenen Joomla-Foren ist oft von einem *Core Hack* die Rede, und verwendet wird diese Bezeichnung üblicherweise für das gesamte CMS-Paket, das Sie über *http://joomla.org* herunterladen können. Dieses Paket enthält Sprachdateien, Templates, Komponenten, Module, Plugins sowie weitere PHP-Bibliotheken von Drittanbietern, weshalb man gerade letztere nicht direkt zum

Core zählen dürfte. Jede Änderung an diesen Originaldateien wird allgemein als Core Hack bezeichnet und das Vorgehen als solches (meist) zurecht verdammt.

Das CMS ist offen für Erweiterungen und stellt hierzu eine große Auswahl an Schnittstellen und zahlreiche Einstiegspunkte bereit. Jeder manuelle Eingriff in den Core *sollte* daher von vornherein unnötig sein und birgt im Gegenteil die schleichende Gefahr nicht nur für potenzielle Fehler. Bei jedem offiziellen Update dieses Kernstücks der Website, kann eine gehackte Core-Datei wieder überschrieben werden. Die gehackten Aufrufe von dann geänderten oder veralteten und ungenutzten Schnittstellen können schnell zu einem Sicherheitsrisiko für die gesamte Website werden. Aber wem erzählen wir das hier eigentlich? Da Sie dieses Buch bereits lesen, rennen wir mit der folgenden Empfehlung ohnehin schon offene Türen bei Ihnen ein: Schreiben Sie lieber eine Erweiterung, anstelle eines Core Hacks.

2.5 Erweiterungen in Joomla

Sie wissen als Betreiber einer Joomla-Website natürlich, dass es folgende Arten an Erweiterungen gibt:

- Komponenten
- Module
- Plugins
- Sprachpakete
- Templates
- Bibliotheken / Dateipakete

Wir beschäftigen uns in diesem Buch mit der Programmierung von Komponenten, Modulen und Plugins. Joomla bringt diese Erweiterungen in verschiedenen Verzeichnissen unter: *./components*, *./modules* und *./plugins*.

Mit Templates und Bibliotheken befassen wir uns hingegen nicht, was nicht bedeutet, dass man in und um Templates herum nicht ebenso praktische und spannende Dinge programmieren kann und das Framework deshalb auch dort bestmöglich einsetzen sollte.

Allen Erweiterungen, die einer Joomla-Website nachträglich hinzugefügt werden, ist eins gemeinsam: der Installationsvorgang, durch den nicht nur die notwendigen Dateien kopiert, sondern zusätzliche Verwaltungsinformationen in der Datenbank abgelegt werden. Erst wenn eine Erweiterung in der Datenbank eingetragen ist, kann sie im Gesamtsystem effektiv genutzt werden.

Bibliotheken und Dateipakete sind eher selten anzutreffen oder werden von komplexeren Erweiterungen und Paketen eher klammheimlich mitinstalliert. Sie tauchen (wenn überhaupt) lediglich im Backend in der Gesamtübersicht installierter Erweiterungen auf.

Die Joomla-Platform im Ordner *./libraries/joomla* ist übrigens aus Sicht der Installationsverwaltung seit Joomla! 1.7 auch nur eine Bibliothek.

Früher oder später werden Sie oder Ihr Kunde feststellen, dass die vorhandenen Möglichkeiten des CMS trotzt kreativer Zweckentfremdung nicht ausreichen und unter den Abertausenden existierender Erweiterungen[8] nichts Adäquates zu finden ist, das den neuen Anforderungen genügt. Die erste Frage ist daher: Welche Art von Erweiterung kommt überhaupt in Frage?

2.5.1 Komponenten

Komponenten repräsentieren den *Hauptinhalt* einer einzelnen Webseite und sind gefragt, sobald besondere Datenstrukturen aufzubauen und zu verwalten sind. Aus Entwicklersicht sind Komponenten die größte Herausforderung. Es sind Anwendungen innerhalb einer Anwendung die zwar einige Grundregeln befolgen müssen, denen ansonsten aber nur sehr wenige Schranken auferlegt sind in dem, was sie machen können und dürfen.

Komponenten bestehen in fast allen Fällen aus einer Komponente für das Backend und einer für das Frontend mit demselben Namen. Die Verwaltungsaufgaben übernimmt hierbei die Komponente im Backend, während mit dem Gegenstück im Frontend die erstellten Daten auf unterschiedliche Weise angezeigt werden.

Die Funktionsweise des CMS bestimmt, dass pro URL/Menüeintrag/Webseite exakt eine Komponente ausgeführt wird. Hierzu wird im Frontend die Komponente einem Menüeintrag zugeordnet und eine Präsentationsform festgelegt, welche die Komponente hierfür bereitstellen muss. Aus der Artikelkomponente kennen Sie diese z. B. als Blog-, Kategorien- oder Einzelansicht. Die Daten, welche in diesen Ansichten dargestellt und verarbeitet werden, stammen aus einer oder mehreren Datenbanktabellen.

Die Programmierung von Komponenten erscheint, jedenfalls am Anfang, äußerst komplex und aufwändig und fast für jeden verwirrend, solange das Verständnis für die API, die verschiedenen Namenskonzepte und das MVC-Entwurfsmuster, erst noch aufgebaut wird. Praktischerweise können Komponenten klein und bescheiden anfangen und stetig wachsen.

2.5.2 Module

Ein Modul bietet sich in den Fällen an, wenn irgendwelche Daten aus irgendeiner Quelle unter bestimmten Bedingungen auf einer Webseite anzuzeigen sind. Die Datenquelle *kann* die aktuelle Joomla-Datenbank sein, muss aber nicht. Die Bedingungen legt der Administrator im Backend fest und entscheidet dort auch, welche Module auf einer bestimmten Webseite angezeigt werden und an welcher Stelle. So kann auf einer Web-

---

[8] Joomla Extension Directory (JED) *http:/extensions.joomla.org*

seite zwar nur eine Komponente ausgeführt werden, aber (nahezu) beliebig viele Module, um weitere Daten um die Komponente herum anzuzeigen.

Backend und Frontend haben ihren eigenen Satz an Modulen, wobei die Großzahl sicherlich im Frontend im Einsatz ist. Ihre Programmierung ist im Vergleich zu Komponenten deutlich einfacher. Dennoch – oder gerade deshalb – sind sie sehr vielseitig einsetzbar und eine gern genutzte Beilage von Komponenten. Unabhängig davon, welche Komponente für den Hauptinhalt der aktuellen Webseite tatsächlich verantwortlich ist, präsentieren Module ihre eigenen Daten bzw. die »ihrer« Komponente.

2.5.3 Plugins

Plugins sind ereignisgesteuerte Routinen und gehören Frontend und Backend gleichermaßen an. An allen Ereignissen, die im CMS zwischen der Anfrage durch den Browser bis zur Auslieferung der fertigen Webseite auftreten, können Plugins teilnehmen. Zu diesen Ereignissen zählen bspw. das Lesen und Speichern der Datensätze, Suchanfragen, Anzeigen der Inhalte, An- und Abmelden der Benutzer. Mit wenigen Zeilen Code können Sie das CMS oder bestimmte Komponenten durch (eigene) Plugins ergänzen und an deren Ereignissen teilhaben, um »Dinge zu tun«.

2.6 Die Datenbank

Joomla hält seine Daten standardmäßig in einer MySQL-Datenbank. Die sogenannte »Community Edition« des MySQL-Server ist für alle gängigen Betriebssysteme kostenfrei verfügbar und wird u. a. mit Komplettpaketen wie MAMP,[9] WAMP[10] oder XAMPP[11] installiert. Bei Linux-Distributionen ist MySQL ebenso wie der Apache-Webserver i. d. R. bereits enthalten und mit hoher Wahrscheinlichkeit läuft auch Ihr Webspace »unter« Apache und »mit« MySQL.

Bei der Installation des CMS wird eine Datenbank mit allerlei Basis- und Verwaltungsdaten eingerichtet. Dazu gehören der erste Benutzereintrag für den Administrator, einige Benutzergruppen und Zugriffsrechte, die Menüs für das Backend und die im Standardpaket enthaltenen Erweiterungen.

Die Tabellen der Datenbank erhalten bei der Installation ein zufälliges Präfix, etwa »a5j6_«. Alle Tabellen mit demselben Präfix gehören zu einer Joomla-Installation. So können sich mehrere CMS eine einzelne Datenbank teilen. Das Präfix ist auch in der Konfigurationsdatei *configuration.php* hinterlegt.

---

9 *http://www.mamp.info*

10 *http://www.wampserver.com*

11 *http://www.apachefriends.org*

Die etwas kryptische Zeichenfolge, die Ihnen das Installationsprogramm als Präfix vorschlägt, dient als Sicherheitsmaßnahme. Jeder Webseitenbetreiber kann zudem ein eigenes Präfix vergeben. Bei einem Live-System sollte unbedingt vermieden werden, ein Standardpräfix wie »jos_« anzugeben.

Hinweis: Wir verwenden für unsere SQL-Beispiele immer das generische Tabellenpräfix »#__«. Zum einen wird es von Joomla bei der Code-Ausführung automatisch in das richtige Präfix Ihrer Installation übersetzt, zum anderen verhindern wir damit, dass Sie den Code einfach auf Ihre Datenbank anwenden und mitunter die falsche Tabelle/Installation erwischen. Sie werden es hassen.

Diese Datenbank ist nach der Installation darauf vorbereitet, Kategorien, Beiträge, Kontakte und andere Benutzerdaten aufzunehmen. Komponenten benötigen meist eine oder mehrere eigene Datenbanktabellen.

Wenn Sie noch keine Erfahrung mit dem Aufbau einer Joomla-Datenbank haben, sollten Sie sich eine kleine Erkundungsreise gönnen und sich die Tabellen nach einer Neuinstallation von Joomla z. B. mit phpMyAdmin anschauen.

Jeder Satz in der Tabelle hat einen eindeutigen Schlüssel. Die betreffende Spalte heißt entweder einfach `id` oder `tabellenname_id`. Über diese identifizierenden Schlüssel verweisen die Datenbanktabellen zum Teil auch auf Inhalte anderer Tabellen und das CMS baut auf diese Art auch hierarchische Strukturen auf.

Tipp: Für Interessierte ein paar Stichworte für die Suchmaschine ihres Vertrauens: *Datenbanknormaliserung* und *Nested Sets*. Ein sehr interessantes Konzept zur Abbildung hierarchischer Datenstrukturen, auf das wir leider nicht allzu detailliert eingehen können.

SQL-Datenbanken werden über die *Structured Query Language* (SQL) abgefragt und bearbeitet. Die aktuellste Dokumentation für MySQL findet sich unter *http://dev.mysql.com/doc/*.

Die gute Nachricht: Man braucht nicht viel SQL-Wissen aufbauen, um einfache Abfragen zu erstellen und Daten auszulesen. Das Framework stellt mehrere Klassen zur Verfügung, mit denen auch ein Anwender ohne detaillierte MySQL-Kenntnisse einfache Abfragen formulieren sowie Daten einfügen und ändern kann. Wer komplexere Auswertungen erstellen und Inhalte mehrerer Tabellen verknüpfen will, wird um eine Einarbeitung in die Konzepte von SQL langfristig nicht herumkommen.

Joomla lässt sich auch mit einem Microsoft-SQL-Server für die Datenbank oder unter Windows Azure einsetzen. Die technischen Voraussetzungen[12] weichen jedoch sehr von

[12] Joomla mit Microsoft Technologien *http://joomla.org/technical-requirements.html*

den klassischen *AMP-Umgebung ab und bergen eigene Tücken in sich, die wir hier leider nicht behandeln können.

2.7 Das MVC-Entwurfsmuster

Ein *Entwurfsmuster* beschreibt einen Weg, ein immer wieder auftretendes Problem standardisiert zu lösen. Der Begriff selbst wurde in der Architektur zuerst eingeführt und für die Softwarearchitektur übernommen. Das Entwurfsmuster, nach dem die Komponenten in Joomla aufgebaut sind, nennt sich MVC: Model-View-Controller. Es ist ein klassisches Muster aus der objektorientierten Programmierung (OOP) und orientiert sich an den besagten, wiederkehrenden Aufgaben interaktiver und interoperabler Applikationen (wie einer Website) und trennt ein Softwareprogramm in drei Schichten:

- Datenbasis (Model)
- Präsentation der Daten (View)
- Programmsteuerung (Controller)

Durch die Trennung der Aufgaben möchte man den Code in erster Linie wiederverwendbar und änderungsfreundlich machen – und geändert wird beim Programmieren viel und ständig.

Das Ganze erschließt sich eigentlich erst im Nachhinein nach dem Motto »Das Leben wird vorwärts gelebt, aber rückwärts verstanden«. Lesen Sie den Abschnitt dann später nochmal wenn die erste Komponente fertig ist, und Sie werden Ihre Aha-Erlebnisse haben!

In einer Web-Anwendung sind ganz typische, wiederkehrende Aufgaben:

1. Anfrage empfangen (URL)
2. Daten von der Datenbasis holen
3. Daten ausgeben

oder

1. Eingabe des Benutzers annehmen
2. Daten prüfen und ggf. speichern
3. Bestätigung ausgeben

und alle dazwischen liegenden Varianten.

Das Prinzip sieht jedoch stets so aus:

- Das Model verwaltet die Daten und enthält die sogenannte *Geschäftslogik*
- Die View präsentiert diese Daten und nimmt Benutzereingaben entgegen

- Der Controller steuert und koordiniert den ganzen Ablauf und die Verbindung zur »Außenwelt«

Die Rollenverteilung der drei Akteure ist somit geklärt. Die Geschäftslogik gehört ins Model, denn beim Speichern eines Datensatzes sind etwa Berechnungen durchzuführen und andere Datensätze / Datenquellen zu aktualisieren. Das kann heute eine XML-Datei sein, morgen eine relationale Datenbank und übermorgen ein hipper, neuer Web-Service in der allmächtigen Wolke.

Aber ein Entwurfsmuster ist keine Zwangsjacke und MVC ist vor allem kein Allheilmittel für *jedes* Softwareproblem. Als Entwickler sollten Sie immer die Lösung verwenden, die sich nicht störend auf das Gesamtsystem auswirkt und einer Aufgabe am besten gerecht wird.

Um es mal ehrlich zu sagen: Für kleine Komponenten schießt man mit MVC schon mit der Kanone auf Spatzen. Aber wenn diese kleinen Komponenten groß werden, und das geht schneller, als man anfangs denkt, dann weiß man ein solch standardisiertes Vorgehen zu schätzen, bei dem insbesondere das Framework dem Entwickler schon viel Basisarbeit abnimmt – weil dieses MVC-Dings tatsächlich *änderungsfreundlich* ist.

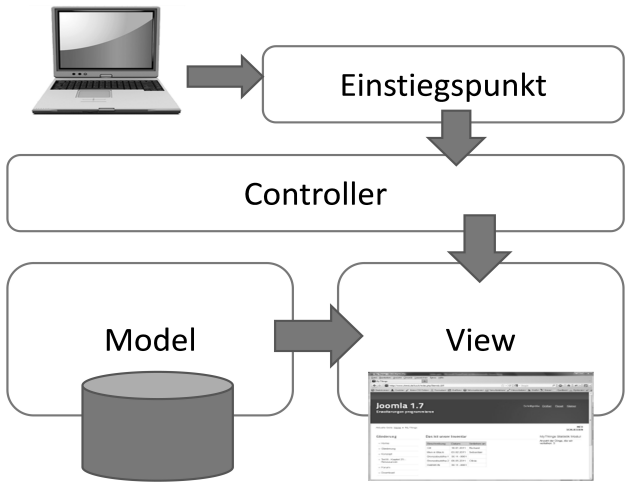


Bild 2.1: Model-View-Controller

Ganz nebenbei erleichtert diese Aufteilung auch die Arbeit im Team. Nehmen wir an, ein Designer hat ein wunderschönes Design entworfen, der HTML-Spezialist hat das Design in ein Template nach allen Regeln der Kunst umgesetzt und alles funktioniert aufs Schönste. Nun beschließt aber die Datenbankspezialistin, dass in der Datenbank ein paar Spalten zusätzlich notwendig sind, um zum Beispiel Statistiken mitzuführen. Sie kann ruhig neue Spalten einfügen – der HTML-Spezialist braucht sich nicht darum kümmern. Seine *Views* erscheinen weiterhin korrekt, auch wenn das Daten*model* umstrukturiert wird. Vielleicht wird auch anstelle der Datenbank eine ganz andere Datenquelle eingesetzt, etwa eine Schnittstelle zu einem anderen System oder eine Datei.

Die Präsentationsschicht bleibt, wie sie ist, denn das Model bietet der View in gewohnter Weise Zugriff auf die benötigten Daten. Wie aber das Model diese Daten letztlich beschafft oder welche Daten sonst noch verarbeitet werden, geht die View nichts an. Umgekehrt kann der Webdesigner die Daten, die vom Model angeboten werden, präsentieren, wie er will – einspaltig, mehrspaltig, als Blog, als Zahlenkolonne in endlosen Tabellen oder als interaktive Infografik – und dies wahlweise eingebunden im seriösen CI-Design des Unternehmen, einem bunten Layout für Kinder oder einem barrierearmen Layout für Blinde; dem Model ist es ganz egal.

Der Controller stellt im allereinfachsten Fall eine Verbindung zwischen Model und View her und steuert deren Dialog und Wechsel anhand der eingehenden Anfragen.

3.2.4 Die berühmte Datei index.html

Sie kennen diese Datei, sie ist in jedem Verzeichnis einer Joomla-Installation enthalten. Jedesmal, wenn Sie ein (Unter-)Verzeichnis anlegen, sollte Ihre erste Aktion das Kopieren dieser Datei aus einem übergeordneten Verzeichnis sein. Sie enthält normalerweise nur eine Zeile:

```
<html><body bgcolor="#FFFFFF"></body></html>
```

Diese Datei schützt davor, dass ein Neugieriger mit dem Browser direkt auf ein Verzeichnis zugreift, sie ist eine Sicherheitsmaßnahme.

Diese *index.html*-Datei ist ein Ärgernis, sie bläht die Verzeichnisse auf und macht sie unübersichtlich. Sie wird nur auf dem Server gebraucht und auch nur dann, wenn der Schutz der Verzeichnisse mittels *.htaccess* nicht möglich ist.

Tipp: Vergessen Sie diese Datei, solange Sie lokal arbeiten. Wenn die Komponente weitergegeben werden soll, ist die Datei notwendig. Die zukünftigen Anwender könnten ja mit unsicheren Servern arbeiten.
Also nehmen Sie diese Datei dann aus irgendeinem der Joomla-Verzeichnisse und kopieren sie am Ende in einem Rundumschlag in alle Ihre Unterverzeichnisse hinein.

3.3 Programmieren – die erste Ausgabe

Joomla ist nach dem Model-View-Controller-Entwurfsmuster aufgebaut. Das wurde im Kapitel 2.7 erklärt.

Jetzt geht es gleich zur Sache mit dem Programmieren – Hands On Code! Als erstes erstellen Sie in *components* (nicht in *adminstrator/components!*) ein neues Verzeichnis *com_mythings.* Das ist ab jetzt Ihr Arbeitsplatz, alle weiteren Verzeichnisse und Dateien werden dort erstellt.

Download: Die Dateien zu diesem Kapitel können Sie herunterladen: *com_mythings_kap03.zip.*

3.3.1 Der Einstiegspunkt der Komponente

Als Erstes programmieren Sie das Einstiegsskript für die Komponente im Frontend, das grundsätzlich den Komponentennamen trägt. Dieses Skript kann man direkt mit einem Link aufrufen: *http://localhost/verzeichnisname/index.php?option=com_mythings*

Hinweis:
Wir verzichten hier im Buch auf alle Kommentare innerhalb des Codes zugunsten einer knappen Darstellung und schreiben die Erläuterungen getrennt.
In den heruntergeladenen Programmen ist der Code ausführlich kommentiert. Lesen Sie dazu auch die Abschnitte zu Programmierkonventionen in Kapitel 7.1.

Erstellen Sie diese Datei:

components/com_mythings/mythings.php

```
<?php
defined('_JEXEC') or die;
jimport('joomla.application.component.controller');

$controller = JController::getInstance('mythings');

$input = JFactory::getApplication()->input;
$controller->execute($input->get('task'));
```

Erklärungen zum Code:

```
<?php
defined('_JEXEC') or die;
```

Joomla setzt beim Start einer Anwendung die globale Konstante `_JEXEC` auf 1. Die Abfrage dieser Konstanten stellt sicher, dass ein Skript nur innerhalb einer Anwendung abgearbeitet wird. Alle globalen Werte sind dann richtig gesetzt, alle Filter sind aktiv. Wenn ein Programm außerhalb dieses Kontexts direkt aufgerufen würde, könnte es zu Fehlern kommen und sogar zur ungewollten Anzeige sensibler Daten.

Die Übersetzung ist sinngemäß: »Ist _JEXEC nicht gesetzt, dann stirb!« Ab hier werden wir auf die Beschreibung dieser Zeile verzichten. Sie steht immer und unbedingt am Anfang jedes PHP-Skripts.

```
jimport('joomla.application.component.controller');
```

`jimport` lädt hier die Klasse `JController`, sodass wir ab sofort deren Variablen und Methoden verwenden können. Dieses `jimport` wird Ihnen noch oft begegnen – im Kapitel 7.2.3 finden Sie mehr dazu.

```
$controller = JController::getInstance('mythings');
```

Der Controller ist für den Ablauf und die Steuerung von Komponenten zuständig. Jede Komponente instanziiert sich als Erstes einen eigenen Controller.

Für Einsteiger in die objektorientierte Programmierung mit PHP
`JController` ist eine Basisklasse der Joomla-Platform. Um zu verstehen, was wir hier machen, hilft Ihnen vielleicht ein Vergleich: Nehmen wir an, wir müssten ein Schiff in einen Hafen bringen. Dafür brauchen wir einen Lotsen. Wir haben die Stellenbeschreibung für Lotsen an Bord. Die holen wir aus der Schublade (`jimport`).

Wir brauchen aber jetzt einen echten Lotsen für unser Schiff, nicht nur eine Stellenbeschreibung. Deshalb fordern wir einen an (`getInstance`). Die Hafenbehörde schickt uns den Herrn MyThingsController. Er ist ausgebildeter Lotse und kann alles, was die Stellenbeschreibung angibt. Er ist nur für unser Schiff zuständig, die anderen Schiffe haben ihren eigenen Lotsen.

```
$input = JFactory::getApplication()->input;
```

Das Input-Objekt hält die Eingaben in der Applikation.

```
$controller->execute($input->get('task'));
```

Hiermit bitten wir die Applikation, uns zu sagen, welche Aufgabe ansteht, und den Controller, sie auszuführen. In diesem Fall ist es denkbar einfach: »Standard-View ausgeben«. Das Joomla-Framework weiß ebensogut wie ein Lotse, was seine Standard-Aufgabe ist, dazu braucht es keine weiteren Anweisungen.

In unserem Vergleich wäre `JInput` der Funker. Diese Klasse von Joomla nimmt alle eingehenden Kommandos und Nachrichten an, säubert sie, hebt sie auf und gibt sie bei Verlangen weiter. Der Funker filtert dabei die Nachrichten, entfernt Störsignale, entsorgt den Spam – der Kapitän fragt den Funker, »Was liegt an?«, und bekommt eine saubere und verständliche Ausgabe des Funkspruchs.

3.3.2 Erweiterung der Klasse JController

Nun erweitern wir die Klasse `JController` für MyThings.

Dass Zusatzprogramme bei Joomla »Erweiterungen« heißen, ist kein Zufall. Natürlich erweitern Komponenten auch den Funktionsumfang des Systems. Aber der Entwickler erweitert buchstäblich die Klassen des Joomla-Frameworks. Das ist jetzt allerdings eine sehr kleine Erweiterung. Genauer gesagt, unser `MyThingsController` erbt einfach alles von der Joomla-Klasse `JController`. Schreiben Sie nun folgenden Code

components/com_mythings/controller.php

```
<?php
defined('_JEXEC') or die;
jimport('joomla.application.component.controller');

class MyThingsController extends JController
{ }
```

Standardverhalten eines jeden Controllers ist die Ausgabe einer View, das können wir ausnutzen und müssen nichts weiter dazu schreiben.

Hinweis: Die Schreibweise von MyThings oder mythings oder MYTHINGS ist an dieser Stelle eigentlich egal. Joomla verwendet Filter, die alles in Kleinbuchstaben umwandeln. Die gemischte Groß-/Kleinschreibung (CamelCaps oder CamelCase genannt) dient nur der Lesbarkeit des Codes.
Für Dateinamen gilt das aber nicht! Sie werden ausnahmslos in Kleinbuchstaben geschrieben, um Probleme mit unterschiedlichen Betriebssystemen zu vermeiden.

Im Model-View-Controller-Entwurfsmuster haben Sie jetzt die Klasse `JController` erweitert und nun sind Model und View dran.

Hinweis: Als PHP-Programmierer fragen Sie sich vielleicht, wo in unseren Skripts das schließende `?>` geblieben ist. Dies und vieles andere ist im Kapitel 7 zu den Programmierkonventionen beschrieben (speziell 7.1.2 PHP-Tags).

3.3.3 Erweiterung der Klasse JModel

`JModel` ist die Klasse, welche Daten aus der Datenbasis bereit stellt. In unserem ersten Beispiel ist die Datenbasis eine MySQL-Datenbank (das ist Standard in Joomla, muss aber nicht sein).

Erstellen Sie ein Verzeichnis *models* in *components/com_mythings* und schreiben Sie folgenden Code

components/com_mythings/models/mythings.php

```
<?php
defined('_JEXEC') or die;
jimport('joomla.application.component.modellist');

class MyThingsModelMyThings extends JModelList
{
   protected function getListQuery()
  {
    $db = $this->getDbo();
    $query = $db->getQuery(true);
    $query->from('#__mythings');
    $query->select('*');
    return $query;
  }
}
```

Erklärungen zum Code:

```
<?php
defined('_JEXEC') or die;
jimport('joomla.application.component.modellist');
```

Für die Behandlung von Listen haben die Joomla-Entwickler eine spezielle Klasse mitgeliefert. Sie enthält Methoden, die alles fix und fertig haben, was man mit Listen so machen kann, z. B. alle Elemente der Tabelle auslesen.

```
class MyThingsModelMyThings extends JModelList
{
```

Es ist eine Namenskonvention bei Joomla, dass der Komponentenname vor der zu erweiternden Klasse steht und danach der Name der speziellen Erweiterung. Unser Model *MyThings* erweitert die Klasse `JModelList`.

```
protected function getListQuery() {
```

Sie müssen nur eins machen, damit Joomla den Rest praktisch von selbst erledigen kann: eine Abfrage definieren. Dafür überschreiben Sie die Methode `getListQuery()` von `JModelList`.

```
$db = $this->getDbo();
$query = $db->getQuery(true);
```

Diese beiden Zeilen sind »zum Auswendiglernen« und der Beginn jeder Datenbankabfrage. Als Erstes benötigen Sie ein Datenbankobjekt und von diesem verlangen Sie ein neues Query-Objekt, das dann die eigentliche Abfrage aufnimmt.

```
$query->from('#__mythings');
$query->select('*');
```

Joomla erzeugt daraus eine Abfrage: `SELECT * from #__mythings`. Und ersetzt den Platzhalter für das Datenbankpräfix aus der Konfigurationsdatei. Warum schreibt man die Abfrage nicht einfach als sql-statement? Joomla baut sich Abfragen selbst zusammen. Es könnte ja auch ein anderer Datenbanktyp als MySQL verwendet werden, dort wäre die Syntax der Abfrage eventuell anders. `JDatabaseQuery` bietet eine Menge Methoden, um eine Abfrage aufzubauen, von denen Sie hier nur zwei sehen: `from` nimmt den Tabellennamen an, `select` nimmt die Spaltennamen an, auf die sich die Suche beziehen wird.

```
return $query;
}
}
```

Nun steht die Abfrage fertig im Query-Objekt und wartet darauf, benutzt zu werden. Das ist die Aufgabe der Klasse `JView`, die Sie als Nächstes erweitern.

3.3.4 Erweiterung der Klasse JView

Erzeugen Sie ein Verzeichnis *views*. So sieht das Verzeichnis *com_mythings* dann aus:

```
com_mythings
    |-- mythings.php
    |-- controller.php
    |-- models
        |-- mythings.php
    |-- views
```

Und im Verzeichnis *views* geht es jetzt weiter.

Eine Komponente kann viele Ansichten bzw. Views haben, und jede Ansicht kann verschiedene Darstellungen bzw. Layouts haben. Nehmen Sie als Beispiel die Komponente *com_content*. Sie kennt die Ansichten »Einzelner Artikel«, »Blog« usw. Bei einer kleinen Komponente mit nur einer View verwendet man gewöhnlich den Namen der Komponente selbst für diese View. Damit bekommen Sie von Joomla vieles gratis, was Sie andernfalls selbst programmieren und bedenken müssen. Es lohnt sich immer, sich an Standards zu halten – jede Abweichung rächt sich furchtbar!

Der View oder die View?
Die Ausgabe heißt im Englischen *the view*. »Denglisch« also **die** View.
Oder etwa nicht? Wie wäre es mit der Übersetzung »der Anblick« sodass es »denglisch« **der** View heißen würde? Es gab da eine kleine Diskussion im Autorenteam (*hüstel*) – aber tatsächlich ist »die Ansicht« die korrekte Übersetzung von *the view* und demnach schreiben wir »**die View**«.

Ihre View heißt *mythings* und dafür erstellen Sie wieder ein eigenes Verzeichnis, unterhalb *views*.

Los geht's mit der Datei *view.html.php*. Dieses Skript stellt die Verbindung zum Model her und besorgt die Daten. Die Ausgabe in Form von HTML-Code, also das Layout, liegt im Verzeichnis *tmpl* innerhalb von *views*. *default.php* ist das Standardlayout.

```
com_mythings
    |-- mythings.php
    |-- controller.php
    |-- models
        |-- mythings.php
    |-- views
        |-- mythings
            |-- view.html.php
```

Speichern Sie in den folgenden Code:

components/com_mythings/views/mythings/view.html.php

```
<?php
defined('_JEXEC') or die;
jimport('joomla.application.component.view');

class MyThingsViewMyThings extends JView
{
   protected $items;

   function display($tpl = null)
   {
      $this->items = $this->get('Items');
      parent::display($tpl);
   }
}
```

Erklärungen zum Code:

```
class MyThingsViewMyThings extends JView
{
```

Die View namens MyThings erweitert die Klasse `JView`.

```
protected $items;
```

Dies sind die Datensätze, welche die View präsentieren wird.

```
function display($tpl = null)
{
```

`display` ist eine Standardfunktion der Klasse `JView`. Diese Methode überschreiben Sie hier.

```
$this->items = $this->get('Items');
```

Wir müssen die Tabellenzeilen erst einmal vom Model bekommen.

Die Methode `get` der Klasse `JView` wendet sich an das Model mit dem Auftrag `get('Items')` – also »Zugriff auf die Datenbank und alle Sätze auslesen«. `JModelList` führt das auch gleich aus, denn die Abfrage ist ja schon definiert worden, und gibt als Ergebnis die Treffer zurück. Damit haben Sie ein Array mit allen Tabellensätzen (Zeilen), die die Abfrage zurückgeliefert hat.

```
parent::display($tpl);
}
}
```

Und jetzt aktivieren Sie die `display`-Methode der Eltern-Klasse `JView`, damit die Zeilen endlich in einem HTML-Dokument ausgegeben werden. `$tpl = null` bedeutet: Default-View ausgeben.

Tipp: Programmierer haben auch ihre Vorlieben. Manche schreiben get('Items') andere wieder get('items'). Die get- und set-Methoden der platform bügeln sich das schon zurecht. Aber verwenden Sie jedenfalls immer dieselbe Schreibweise, das erleichtert dem Leser das Verständnis.

3.3.5 Die Listenansicht

Erstellen Sie in *views/mythings* ein weiteres Verzeichnis *tmpl.*

com_mythings
|-- *mythings.php*
|-- *controller.php*
|-- ***models***
|-- *mythings.php*
|-- ***views***
|-- ***mythings***
|-- *view.html.php*
|-- ***tmpl***
|-- *default.php*

Schreiben Sie den nachstehenden Code. Es ist die Ausgabe der Datensätze im Browser. Wir machen es uns einfach und produzieren eine ganz normale Tabelle, aber geben nur ein paar Spalten unserer Datensätze aus.

Hinweis: Der Code hier enthält einige `style`-Angaben. Das ist »igitt«, solche Sachen gehören in die css-Datei, nicht in den Quellcode. Hier verwenden wir das ausnahmsweise, um schnell zu einem Ergebnis zu kommen. Zu CSS kommen wir später.

components/com_mythings/views/mythings/tmpl/default.php

```
<?php
defined('_JEXEC') or die;

$nullDate = JFactory::getDbo()->getNullDate();
?>
<h1>Wir verleihen viele Dinge</h1>

<?php if ($this->items){ ?>
```

```
<table>
<tr>
  <th style="background: #ccc;">Bezeichnung</th>
  <th style="background: #ccc;">Kategorie</th>
  <th style="background: #ccc;">Ausgeliehen an</th>
  <th style="background: #ccc;">Ausgeliehen am</th>
</tr>

<?php foreach ($this->items as $item) : ?>
<tr>
  <td><?php echo $item->title; ?></td>
  <td><?php echo $item->category; ?></td>
  <td><?php echo $item->lent_by; ?> </td>
  <td>
      <?php
      if ($item->lent != $nullDate) {
          echo JHtml::_('date', $item->lent, 'd.m.Y');
      } ?>
  </td>
</tr>
<?php endforeach;?>

</table>
<?php } ?>
```

Joomla verwendet die *default.php*, wenn beim Aufruf der View nichts anderes angegeben ist.

Erklärungen zum Code:

```
$nullDate = JFactory::getDbo()->getNullDate();
```

Diese Methode liefert den Nullwert der Datenbasis für einen Zeitwert zurück, er könnte bei verschiedenen Datenbasen ja völlig verschieden aussehen.

```
if ($this->items){ ?>
```

Falls Datensätze gefunden wurden, bauen Sie eine Tabelle auf.

Der Anfang ist ganz normaler HTML-Code und braucht keine Erläuterung. Erst ab der foreach-Schleife wird es interessant.

```
<?php foreach ($this->items as $item) : ?>
```

Erinnern Sie sich? In *view.html.php* haben Sie `$items` von der Datenbank geholt und als Eigenschaft der View (`$this`) gespeichert. Jede Zeile ist ein Array mit dem Inhalt eines Datenbanksatzes, auf dessen einzelne Elemente Sie direkt über den Spaltennamen zugreifen können.

```
<?php
if ($item->lent != $nullDate) :
   echo JHtml::_('date', $item->lent, 'd.m.Y');
endif;
?>
```

Hier geben Sie das Datum der Ausleihe aus, falls es gesetzt ist. `JHtml` hat zahlreiche Methoden, mit denen Elemente einer HTML-Seite aufbereitet werden. Sie werden diese im Streifzug durch die API näher kennenlernen.

3.3.6 Einen Menütyp anlegen

Fast geschafft! Aber Sie brauchen noch einen Menüpunkt, damit Sie die View auch aufrufen können. Dazu brauchen Sie die im Backend einen Menütyp. Schreiben Sie diesen Code (UTF-8 ohne BOM):

components/com_mythings/views/mythings/tmpl/default.xml

```
<?xml version="1.0" encoding="utf-8"?>
<metadata>
  <layout title="My Things "></layout>
</metadata>
```

So sieht die Struktur jetzt aus:

com_mythings
- |-- *mythings.php*
- |-- *controller.php*
- |-- ***models***
  - |-- *mythings.php*
- |-- ***views***
  - |-- ***mythings***
    - |-- *view.html.php*
    - |-- ***tmpl***
      - |-- *default.php*
      - |-- *default.xml*

Glückwunsch! Ihre erste Komponente mit einer ersten View im Frontend ist fertig. Sie können jetzt im Backend einen Link auf diese View anlegen.

Bild 3.6: Einen Link zu MyThings anlegen

Und jetzt ins Frontend um zu gucken. Die Übersicht ist da!

Bild 3.7: Erste View: Die Übersicht

Allerdings kann man noch nicht viel damit anfangen, zumindest möchte man ja die Detailangaben zu den Dingen sehen können, und deshalb fügen Sie noch schnell eine zweite View hinzu.

3.4 Zweite View: Detailansicht

Durch Klick auf eins der Dinge soll die Detailansicht aufrufbar sein. Diese View bekommt den Namen »mything«. Falls Sie über diese Namen den Kopf schütteln – das Joomla-Namenskonzept erklärt, warum ich mir hier nichts Originelleres einfallen lasse. In den folgenden Kapiteln erfahren Sie mehr dazu.

3.4.1 Link zur Detailansicht einfügen

Zunächst einmal brauchen wir in der Übersicht einen Link, den der Anwender anklicken kann, um zur Detailansicht zu gelangen. Dieser Link adressiert die Komponente *com_mythings*, die View *mything* und übergibt den Schlüssel des Satzes, der gezeigt wer-

den soll. Dazu muss nur die Default-View an einer Stelle erweitert werden: Ändern Sie nur die fett markierte Zeile im Skript.

components/com_mythings/views/mythings/tmpl/default.php

```
. . . .
<?php foreach ($this->items as $row) : ?>
<tr>
<td><?php echo $item->category; ?></td>
<td><?php echo $item->title; ?></td>
<td><?php echo $item->lent_by; ?> </td>
  . . . .
```

Ersetzen Sie diese Markierung durch folgenden Code:

```
<td>
<?php
  $link =
  JRoute::_("index.php?option=com_mythings&view=mything&id=" .$item->id);
  echo '<a href="' .$link .'">' . $item->title .'</a>'; ?>
</td>
```

Erklärung zum geänderten Code

Der eigentliche Link zur Detailansicht wird von `JRoute::_()` je nach den SEO-Einstellungen der Anwendung erzeugt und in einem `<a>`-tag verwendet.

Bild 3.8: Links zur Detailansicht

3.4.2 Das Model MyThing

Auch bei der Entwicklung von Komponenten führen viele Wege zum Ziel. Um die Details eines Dings anzuzeigen, erstellen wir eine zweite View, ganz nach dem MVC-Entwurfsmuster, mit einem Model *MyThing*, einer View *MyThing* und dort einem Standardlayout *default*.

Irgendwie kommt einem das doch recht abwegig und überdimensioniert vor. Ein Lastwagen, um eine Streichholzschachtel zu transportieren? Ein extra Model und eine extra View nur um einen einzelnen Satz anzuzeigen, der sowieso schon da ist? Ein zweites Layout anstelle einer View würde es doch auch tun? Ganz recht. In meiner ersten

Version dieses Kapitels stand an dieser Stelle nur ein zweites Layout namens »thing«. Aber die Komponente wird größer werden, wer weiß, was wir mit dieser Detailansicht noch machen, und deshalb gibt es nun doch ein extra Model, mit dem der gewünschte Satz aus der Datenbank gelesen wird. Die Basisklasse dafür ist `JModelItem`.

components/com_mythings/models/mything.php

```
<?php
defined('_JEXEC') or die;
jimport('joomla.application.component.modelitem');

class MyThingsModelMyThing extends JModelItem
{
    public function getItem($id = null)
    {
      $app = JFactory::getApplication();
      $requested_id = $app->get('input')->get('id', 0, 'int');
      if ($requested_id > 0) {
        $db = $this->getDbo();
        $query = $db->getQuery(true);
        $query->from('#__mythings');
        $query->select('*');
        $query->where('id = ' .$requested_id );
        $db->setQuery($query);
        $result = $db->loadObject();
      }
      return $result;
    }
}
```

Erklärung zum Code:

```
class MyThingsModelMyThing extends JModelItem
{
    public function getItem($id = null)
    {
```

Die Methode `getItem()` liest einen einzelnen Satz aus der Datenbank aus.

```
$input = JFactory::getApplication()->input;
$requested_id = $input->get('id', 0, 'int');
```

Wenn der Anwender auf einen Link geklickt hat, steht die `id` des angeforderten Satzes im Model MyThings.

```
      if ($requested_id > 0) {
        $db = $this->getDbo();
        $query = $db->getQuery(true);
        $query->from('#__mythings');
```

```
        $query->select('*');
        $query->where('id = ' .$requested_id );
```

Die Abfrage selbst ist im Prinzip dieselbe wie schon im Model MyThings, nur ist die Suche auf den einen gewünschten Satz eingeschränkt.

```
        $db->setQuery($query);
        $result = $db->loadObject();
```

Hier wird die Abfrage ausgeführt. Das Ergebnis ist ein einziger Datensatz, der an die View zurückgegeben wird.

```
      }
      return $result;
    }
}
```

3.4.3 View für die Detailansicht erstellen

Sie haben jetzt nur noch eine View *MyThing* für die Detailansicht zu erstellen, zunächst wieder ohne Anspruch auf Schönheit. Am Anfang steht das Einrichten der Verzeichnisstruktur: Unter *views* erstellen Sie ein weiteres Verzeichnis *mything*.

component/com_mythings/views/mything/view.html.php

```
<?php
defined('_JEXEC') or die;
jimport('joomla.application.component.view');

class MyThingsViewMyThing extends JView
{
    protected $item;

    function display($tpl = null)
    {
        $this->item = $this->get('Item');
        parent::display($tpl);
    }
}
```

Dieser Code unterscheidet sich von dem Code der View *MyThings* nur durch ... Haben Sie es bemerkt? Nur durch einen einzigen Buchstaben! MyThing statt MyThings und Item statt Items.

Weiter geht's mit dem Layout, genau wie bei der Übersicht: Das neue Verzeichnis *tmpl* nimmt das Layout in der Datei *default.php* auf. Die Tabellendarstellung ist eine Peinlichkeit und ist der Übersichtlichkeit auf Papier geschuldet – seien Sie ruhig kreativ bei Ihrem Design!

component/com_mythings/views/mything/tmpl/default.php

```
<?php
defined('_JEXEC') or die;
$item = $this->item;
?>

  <h1><?php echo $item->title; ?></h1>
  <strong><?php echo $item->description; ?></strong>
  <table>
  <tr>
    <td>Zustand: </td>
    <td><?php echo $item->state; ?></td>
  </tr>
  <tr>
    <td>Wert: </td>
    <td><?php echo $item->value; ?></td>
  </tr>
  <tr>
    <td>Gewicht: </td>
    <td><?php echo $item->weight; ?></td>
  </tr>
  </table>
  <h1>
  <?php
  if ($item->lent == $nullDate) {?>
    Ist ausleihbar
  <?php  } else {?>
    Ist gerade verliehen.
  <?php }?>
  </h1>
```

Erklärungen zum Code:

```
<?php
defined('_JEXEC') or die;
$item = $this->item;
```

Die View zeigt den Inhalt des Items.

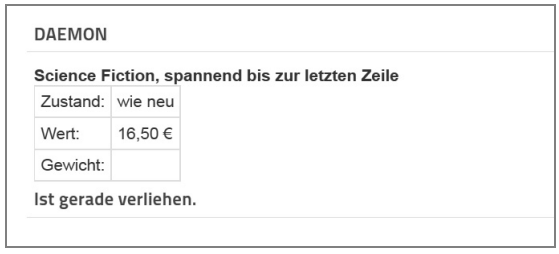
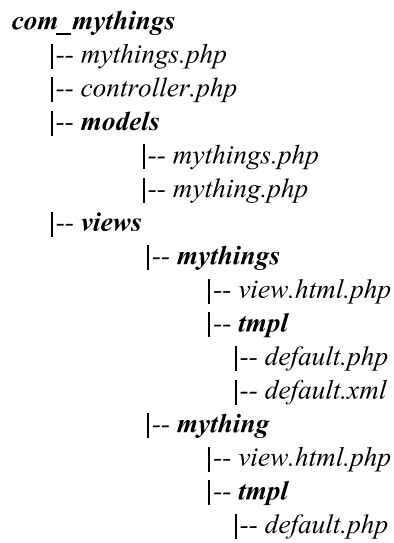


Bild 3.9: Die zweite View: Detailansicht

Weiter gibt es nichts zu erklären. Um die Kosmetik der Ansichten kümmern wir uns später. Das Framework bietet dafür viele und interessante Funktionen, die eine detaillierte Behandlung in einem eigenen Kapitel (Kap. 16) verdient haben.

3.5 Zusammenfassung

Mit ganz wenig Programmierarbeit haben wir jetzt eine erste Frontend-Komponente mit zwei Views erstellt und eine Basis für die weitere Entwicklung geschaffen.



Sie haben gelernt

- wie im MVC-Entwurfsmuster Model View und Controller zusammenarbeiten,
- was es heißt, die Basisklassen `JModel`, `JView` und `JController` zu erweitern,
- wichtige Klassen/Methoden einzusetzen, zum Beispiel
  - `JFactory`, um eine Referenz zu erhalten,
  - `JHtml`, für die Aufbereitung von html-Ausgaben,
  - `JRoute`, für die Aufbereitung von Links,
  - `JDatabase`, für den Datenbank-Nullwert,
  - `JInput`, für die Abfrage der Benutzereingabe,
- den Input-Bereich der Applikation anzusprechen,
- eine Datenbankabfrage zu erstellen und das Ergebnis anzufordern.

Ebenso schnell schreiben Sie im nächsten Abschnitt Ihr erstes Modul.

Stichwortverzeichnis

Axel Tüting
Christiane Maier-Stadtherr
René Serradeil

Joomla!-Extensions entwickeln

Eigene Komponenten, Module und Plugins programmieren

Manchmal muss es eine eigene Erweiterung sein! Und Erweiterungen für Joomla! zu entwickeln, ist gar nicht so schwer. Dieses Buch zeigt, wie Sie eine maßgeschneiderte Lösung für Ihren Bedarf entwickeln. Die Autoren beschreiben Schritt für Schritt das Vorgehen vom Entwurf bis zur fertigen eigenen Komponente für Joomla! 2.5.

▶ Komponenten, Module und Plugins

Es gibt unterschiedliche Möglichkeiten, Erweiterungen in Joomla! einzubringen. Neben einer größeren Komponente werden Ihnen hier auch Beispiele für die Erstellung zusätzlicher Module oder Plugins gezeigt.

▶ Die Joomla-API

Die Programmierschnittstelle von Joomla! ist sehr umfangreich. Mit der Einführung des Frameworks und später der Joomla! Platform wurde zwar einiges vereinfacht und vereinheitlicht, aber trotzdem ist es schwer, hier einen Überblick zu erlangen. Damit Sie sich in der Fülle der Klassen und Methoden zurechtfinden, gibt es in diesem Buch eine Einführung in die API der Joomla!-Platform 11.4 im Joomla!-CMS 2.5.

▶ Vom Entwurf bis zum Installer

Zuerst steht eine Idee, was eine Erweiterung können soll, dann aber muss diese umgesetzt werden. Am Beispiel einer Komponente für ein Verleihsystem, das mit unterschiedlichsten Funktionalitäten aufwartet, wird dieser Prozess in allen Schritten durchgespielt.

Auf www.buch.cd

Der komplette Quellcode des Buches.

Besuchen Sie unsere Website
www.franzis.de

Aus dem Inhalt:

Über die Autoren:

Axel Tüting ist Fachinformatiker, Anwendungsentwicklung. 2008 gründete er die Firma time4mambo für Programmierung, Schulung und Webentwicklung mit eindeutigem Fokus auf Joomla! bzw. seinen Vorgänger Mambo.

Christiane Maier-Stadtherr war viele Jahre in der Softwareentwicklung tätig, bis sie sich 2010 als Web-Entwicklerin in München selbständig machte. Ihr Hauptinteresse ist die Barrierefreiheit des Web, ihre Leidenschaft sind Schulungen.

René Serradeil aus Karlsruhe ist seit 1996 als freiberuflicher Web-Worker tätig. Für Joomla! 1.0 ersann er das deutsche Backend „JooDe“ und war später maßgeblich am Aufbau von docs.joomla.org beteiligt. Im Netz ist er mit WebMechanic.biz unter der Haube tätig und bietet Trainings für HTML, CSS, JavaScript und Adobe Photoshop an.

30,– EUR [D]
ISBN 978-3-645-60134-4

FRANZIS