

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.

Alle Angaben in diesem Buch wurden vom Autor mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. Der Verlag und der Autor sehen sich deshalb gezwungen, darauf hinzuweisen, dass sie weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernehmen können. Für die Mitteilung etwaiger Fehler sind Verlag und Autor jederzeit dankbar. Internetadressen oder Versionsnummern stellen den bei Redaktionsschluss verfügbaren Informationsstand dar. Verlag und Autor übernehmen keinerlei Verantwortung oder Haftung für Veränderungen, die sich aus nicht von ihnen zu vertretenden Umständen ergeben. Evtl. beigefügte oder zum Download angebotene Dateien und Informationen dienen ausschließlich der nicht gewerblichen Nutzung. Eine gewerbliche Nutzung ist nur mit Zustimmung des Lizenzinhabers möglich.

© 2012 Franzis Verlag GmbH, 85540 Haar bei München

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien. Das Erstellen und Verbreiten von Kopien auf Papier, auf Datenträgern oder im Internet, insbesondere als PDF, ist nur mit ausdrücklicher Genehmigung des Verlags gestattet und wird widrigenfalls strafrechtlich verfolgt.

Die meisten Produktbezeichnungen von Hard- und Software sowie Firmennamen und Firmenlogos, die in diesem Werk genannt werden, sind in der Regel gleichzeitig auch eingetragene Warenzeichen und sollten als solche betrachtet werden. Der Verlag folgt bei den Produktbezeichnungen im Wesentlichen den Schreibweisen der Hersteller.

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 anfangen, 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 offiziellen 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 anfangen, 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

1	Bevor es losgeht	17
1.1	Die GNU General Public License (GPL)	17
1.1.1	Was ist eigentlich GNU GPL?	17
1.1.2	Open Source versus GPL	19
1.1.3	Was bedeutet das für uns?	19
1.2	Arbeitsumgebung	20
1.2.1	Lokale Joomla-Installation	20
1.2.2	Der geeignete Arbeitsplatz	20
1.2.3	Der Quelltext-Editor	20
1.2.4	Integrierte Entwicklungsumgebung	21
1.2.5	Debugger	22
1.3	Git und Github	22
2	Joomla-Grundwortschatz	23
2.1	Internetjargon	23
2.2	CMS – Backend und Frontend	24
2.3	Zwei Hüte, ein Kopf: Platform vs. Framework	25
2.4	Finger weg vom Core	26
2.5	Erweiterungen in Joomla	27
2.5.1	Komponenten	28
2.5.2	Module	28
2.5.3	Plugins	29
2.6	Die Datenbank	29
2.7	Das MVC-Entwurfsmuster	31
3	Erste Ausgabe im Frontend	35
3.1	Der Ausgangspunkt	35
3.2	Vorbereitung	36
3.2.1	Die Komponente MyThings einschmuggeln	37
3.2.2	Die Datenbanktabelle anlegen	38
3.2.3	Beispieldaten einfügen (optional)	40
3.2.4	Die berühmte Datei index.html	41

3.3	Programmieren – die erste Ausgabe	41
3.3.1	Der Einstiegspunkt der Komponente	41
3.3.2	Erweiterung der Klasse JController	43
3.3.3	Erweiterung der Klasse JModel	44
3.3.4	Erweiterung der Klasse JView	46
3.3.5	Die Listenansicht	48
3.3.6	Einen Menütyp anlegen	50
3.4	Zweite View: Detailansicht	51
3.4.1	Link zur Detailansicht einfügen	51
3.4.2	Das Model MyThing	52
3.4.3	View für die Detailansicht erstellen	54
3.5	Zusammenfassung	56
4	Unser erstes Modul	57
4.1	Model: helper.php	58
4.1.1	Hintergründiges zu \$query und SQL	60
4.2	Controller: mod_mythingsstats.php	61
4.3	Layout: default.php	62
4.3.1	Hintergrundwissen: Warum dieser Aufbau?	63
4.4	Installer: mod_mythingsstats.xml	64
5	Das erste Plugin	65
5.1	Einfach nur einen Titel	66
6	Sprachen	69
6.1	Eleganter: mit Sprachdateien	69
6.2	Eins, viele oder nichts	72
6.3	Und noch mehr Sprache: *.sys.ini	72
6.3.1	Gleiches Schlüsselwort – unterschiedliche Texte	74
6.4	JText im Überblick	76
6.5	Kleine Anmerkung zu den Sprachen	77
7	Standards, Regeln, Konventionen	79
7.1	Allgemeine Empfehlungen zum Code	79
7.1.1	PHP 5.2.x	80
7.1.2	PHP-Tags	80
7.1.3	Textkodierung UTF-8 ohne BOM	81
7.1.4	Code einrücken, Zeilen umbrechen, Leerzeilen verwenden	81
7.1.5	Kontrollblöcke klammern	85

7.1.6	Sprechende Namen vergeben	87
7.1.7	Schreibweisen von Bezeichnern und Namen in PHP	88
7.1.8	Joomla-Sprache ist britisches Englisch	88
7.1.9	You can say you to me	89
7.1.10	Reservierte Wörter in Klassennamen	89
7.1.11	Dokumentation durch Kommentare	91
7.2	Namenskonzepte in Joomla	95
7.2.1	Klassen, damals und heute	95
7.2.2	Frontend, Backend, »joomla« und »cms«	96
7.2.3	Schnelle Wege zur Klasse	96
8	Objekte & Co.	101
8.1	Objekte sind ein Abbild der Wirklichkeit	101
8.2	Abstrakte Klassen	104
8.3	Statische Klassen	105
8.4	Und das Ganze mit PHP	106
8.4.1	Der Konstruktor	106
8.4.2	... und der Destruktor	106
8.4.3	Kapselung	107
8.4.4	Abstrakte Klassen in PHP	108
8.4.5	Statische Klassen in PHP	108
8.4.6	Mehrfachvererbung	109
8.4.7	Wer ist \$this?	110
8.4.8	Das __call-Center	111
8.4.9	Singleton-Entwurfsmuster	112
8.5	Abschließende Bemerkungen	113
9	Die Joomla-API – eine Art Einführung	115
9.1	Rundgang um die Pakete der Plattform	115
9.2	Objektfabrik: JFactory	116
9.2.1	Fabrikmodel oder getInstance()?	117
9.3	Keine Referenz	118
9.3.1	Der Urschleim: base	119
9.3.2	Anwendungen: application	120
9.3.3	Ausgabe: document	130
9.3.4	HTML-Fragmente: html	134
9.3.5	Formulare: form	134
9.3.6	Validierung: filter	134
9.3.7	Texte: string	135
9.3.8	Sprachen: language	136

9.3.9	Systemumgebung: environment	137
9.3.10	Helferlein: utilities	138
9.3.11	Sitzungsverwaltung: session	139
9.3.12	Mutter aller Parameter: registry	141
9.3.13	Datenbanken: database	142
9.3.14	Benutzer: user	148
9.3.15	Zugriffsrechte: access	150
9.3.16	Plugins und Ereignisse: plugin & event	153
9.3.17	Protokolle: log	154
9.3.18	Postversand: mail	155
9.3.19	Dateisystem: filesystem	157
9.3.20	Netzwerken über FTP & LDAP: client	158
9.3.21	Code-Archiv: Github	159
9.3.22	Optimierung: cache	159
9.3.23	Installieren und aktualisieren: installer & updater	162
9.3.24	Bilder manipulieren: image	163
9.4	Extrawurst CMS	163
9.4.1	Hab dich! in: cms/captcha	163
9.4.2	Klick mich! Schnellstart-Symbole in: cms/html	164
9.4.3	Und der ganze Rest	165
9.5	Konstante Pfade zum richtigen Ziel	165
9.6	Sammelbecken JHtml	170
9.6.1	Verwaltung	171
9.6.2	Datum, Kalender und Ressourcen	173
9.6.3	Formularelemente	175
9.6.4	HTML-Widgets und Behaviors	176
9.6.5	Toolbar	177
9.6.6	Mögen Sie Mootools ... nicht?	179
9.7	Die Joomla-API unterm Strich	180
9.7.1	Es bleibt alles anders	180
9.8	Einsatz missbilligt: deprecated	181
9.8.1	Schritte bei der Migration von 1.5er Erweiterungen	182
10	Unsere Komponente: Backend	187
10.1	Das Einstiegs-Skript	188
10.2	Der Chef-Controller	189
10.3	Die Verzeichnisstruktur	189
10.4	Die Controller	191
10.4.1	Controller für die Listenansicht	192
10.4.2	Controller für die Formularansicht	192

10.5	Die Views	192
10.5.1	Die Toolbar	192
10.5.2	Die Klasse JView	193
10.5.3	Die Listenansicht: View MyThings	193
10.5.4	Das Default-Layout	195
10.5.5	Die Formularansicht: View MyThing	198
10.5.6	Das Layout edit	200
10.6	JTable – die Schnittstelle zur Datenbank	202
10.7	Die Models	203
10.7.1	Das Model MyThings	203
10.7.2	Die Formulardefinition	204
10.7.3	Das Model MyThing	205
10.8	Noch mal Controller	207
10.8.1	Der Controller der Listenansicht: MyThings	207
10.8.2	Der allgemeine Controller	208
10.9	Das Geheimnis der drei Controller	209
10.10	Die Sprachdateien	211
10.11	Aus Fehlern lernen	212
10.12	Backend, die erste – geschafft!	213
11	Formulare cleverer: JForm	215
11.1	XML-Dateien für HTML-Formulare	215
11.1.1	Container, Strukturen, Daten und Dateien	215
11.1.2	Fields und Attribute. Viele davon	217
11.1.3	Datenfilter	219
11.1.4	Kleine Feldstudie	220
12	Fehlerbehandlung	225
12.1	Mit Ausnahmen umgehen	226
12.1.1	Mit Ausnahmen üben	229
12.1.2	Die Ausnahme zur Regel machen	231
12.2	Ausnahmen in MyThings	231
13	Alles bleibt anders – Exkurs Refactoring	233
13.1	Refactoring im Backend	234
13.2	Refactoring im Frontend – Hausaufgabe!	235
13.3	Schrotflinten-Chirurgie und Nebenwirkungen	235

14	Filter – Sortierung – Pagination	237
14.1	Die Listenansicht wird erwachsen	238
14.1.1	Neue Eigenschaften für die View	238
14.1.2	Die Ausgabe anpassen	239
14.2	Das Model und sein state	243
15	Kategorien, User und JForms	247
15.1	Komponenten-Submenü einschleusen	247
15.2	Normalisierung der Datenbank	249
15.2.1	Mini-Exkurs Normalisierung	249
15.2.2	Tabelle #__mythings ändern	250
15.2.3	MyThingsTableMyThings anpassen	251
15.3	Das Model anpassen: categories und user	251
15.4	Die Listenansicht anpassen	255
15.4.1	Das Submenü der Listenansicht	255
15.4.2	Listenansicht mit Kategoriefilter	257
15.5	Formularansicht und JForms	259
15.5.1	Man nehme ... fieldsets und fields	259
15.5.2	... garniere sie mit Eigenschaften	260
15.5.3	... zeige sie als Formular	263
15.5.4	Layout-Variante mit Schleife	265
15.6	Erweiterung der Sprachdateien	265
15.7	Frontend anpassen	266
16	Kosmetik fürs Frontend	267
16.1	Layout-Overrides erstellen	267
16.2	Alternatives Layout erstellen	268
16.3	Ressourcen einbauen: JDocument	271
16.3.1	Gegen den Inline-<script>	272
16.3.2	Probieren Sie mal Heredocs	273
16.4	Browser-Ressourcen in ./media	275
16.4.1	Medien in der manifest.xml	275
16.5	Trickreiche Ausführung	275
17	Plugins – Arbeiten im Untergrund	277
17.1	Klammern ersetzen im Beitrag	277
17.2	Dynamischer Titel per Parameterübergabe	280
17.3	Triggern von Ereignissen	282

17.4	System-Plugins	285
17.4.1	Plugin MyThings Systemtools: Verleihstatus	286
17.5	Bemerkenswerte Ereignisse	290
17.5.1	onContentBeforeSave und onContentAfterSave	290
17.5.2	onContentPrepareData und onContentPrepareForm	291
18	Module: Daten immer anders	293
18.1	Eines für alle	293
18.1.1	Das einfachste vorweg: Erweiterte Optionen	295
18.1.2	Eigene Parameter in den Basisoptionen	295
18.1.3	Alles unter Kontrolle	300
18.1.4	Ausgabe der Arrays	300
18.2	Und weiter...	301
19	Die Komponente wird konfigurierbar	303
19.1	Globale Einstellungen der Komponente	304
19.1.1	Toolbar: Das Icon »Optionen«	304
19.1.2	Die config.xml	305
19.1.3	Verwendung im Frontend	307
19.2	Einstellungen zum Menüpunkt	307
19.3	Eine Extrawurst für jedes Ding	308
19.3.1	Parameter definieren	309
19.3.2	Formularansicht erweitern	309
19.3.3	Datenbank und jTable	310
19.4	Feuer frei! Einsatz im Frontend	312
19.4.1	Wer gewinnt? Parameter zusammenführen	312
19.4.2	Menü-Parameter für die Listenansicht	313
19.4.3	Die Detailansicht MyThing	314
19.5	Zusammenfassung	315
19.5.1	Kurz und bündig – Parameterdefinitionen	316
19.6	Sprachdateien	317
20	Wer darf was? Zugriffsrechte	319
20.1	Globale Konfiguration der Berechtigungen	319
20.2	Der Dreh- und Angelpunkt: Die access.xml	320
20.3	Asset – Das Objekt der Begierde	323
20.3.1	Die Tabelle #__assets	323
20.3.2	Tabelle #__mythings erweitern	324
20.3.3	jTable erweitern, um assets zu generieren	325

20.4	Der User, das bekannte Wesen	327
20.5	Wächter in Aktion – Zugriffsrechte prüfen	327
20.5.1	Zugang zur Komponente einschränken	327
20.5.2	Zugriffsregeln für einzelne Datensätze	330
20.6	Zusammenfassung	331
21	Formular im Frontend	333
21.1	Das Model	333
21.2	Die View	336
21.3	Der Controller	337
21.4	Weniger ist mehr	339
21.5	Sprachschlüssel	339
22	Routing und SEF	341
22.1	BuildRoute	342
22.1.1	Alias, Slug und der Rest	344
22.1.2	Die Itemid sticht	344
22.2	ParseRoute	345
23	Installer	349
23.1	Das XML-Grundgerüst	349
23.1.1	!DOCTYPE	352
23.1.2	manifest.xml	352
23.1.3	method: install oder upgrade	353
23.2	Die Container im Überblick	353
23.3	Übersicht der einzelnen Standardtypen	354
23.3.1	Komponente	354
23.3.2	Modul	358
23.3.3	Plugin	359
23.3.4	Language	360
23.3.5	Template	363
23.3.6	Library	364
23.3.7	File	365
23.3.8	Package	367
23.4	install-, uninstall-, update-Skripte	369
23.4.1	SQL-Skripte	370
23.4.2	PHP-Skripte	373
23.5	Media	375

23.6	Update-Server	376
23.6.1	type="extension"	376
23.6.2	type="collection"	378
23.6.3	Anmerkungen zum Update-Server	380
Anhang A – Nützliche Links		381
A.1	Entwicklungsumgebungen und -hilfen	381
A.1.1	NetBeans	381
A.1.2	PHPEdit	381
A.1.3	EasyCreator	381
A.1.4	PHPUnit	381
A.1.5	Versionskontrolle und gemeinsame Projekte	382
A.2	GNU/GPL	382
A.3	PHP	382
A.4	SQL	382
A.5	HTML, CSS und Co.	382
A.6	Programmierung allgemein	383
A.7	Joomla	383
Anhang B – Events (Plugins)		385
B.1	System (Anwendung)	385
B.2	Authentifizierung, Login, Logout	385
B.3	Content (allgemein)	386
B.4	Erweiterungen (Backend)	387
B.5	Installation	387
B.6	Formulare und Models (JForm, JModel)	387
B.7	Komponente: Beiträge (com_content)	388
B.8	Komponente: Kontakte (com_contact)	389
B.9	Komponente: Benutzer (com_users)	390
B.10	Komponente: Suche	390
B.11	Editor	391
B.12	Captcha	392
Anhang C – Entwicklungsumgebung		393
C.1	PHPEdit – Axels Editor	394
C.1.1	Moderne Oberfläche	394
C.1.2	Alles Einstellungssache	396
C.1.3	Benutzung von Frameworks	398
C.1.4	Debug	399

C.1.5	Datenbankabfragen	399
C.1.6	Fazit	401
C.2	NetBeans – Christianes Werkzeugkasten	402
C.2.1	NetBeans installieren	402
C.2.2	XDebug installieren	403
C.3	NetBeans verwenden	404
C.3.1	Projekt einrichten	405
C.3.2	Klassen und Methoden	406
C.3.3	PHP-Editor	407
C.3.4	Debuggen	408
Stichwortverzeichnis		409

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 Begriffe⁵ 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 mit- samt 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 *Internetseite* und *Website* 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 semantischen⁷, 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! Plattform 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 Plattform 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 Plattform 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 Plattform-Applikationen haben wird, muss sich erst noch zeigen.

Die Joomla!-Plattform 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 *Plattform* 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/Plattform 11.4 noch als Bestandteil der Plattform identifizieren *würden*, kann im CMS 3.x/Plattform 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!-Plattform (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 Plattform 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! Plattform*.

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 »Plattform« 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-Plattform 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 trotz kreativer Zweckentfremdung nicht ausreichen und unter den Abertausenden existierender Erweiterungen⁸ 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 WAMP¹⁰ oder XAMPP¹¹ installiert. Bei Linux-Distributionen ist MySQL ebenso wie der Apache-Webserver i. d. R. bereits enthalten und mit hoher Wahrscheinlichkeit läuft auch Ihr Webespace »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.

Tip: Für Interessierte ein paar Stichworte für die Suchmaschine ihres Vertrauens: *Datenbanknormalisierung* 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 Voraussetzungen¹² 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.

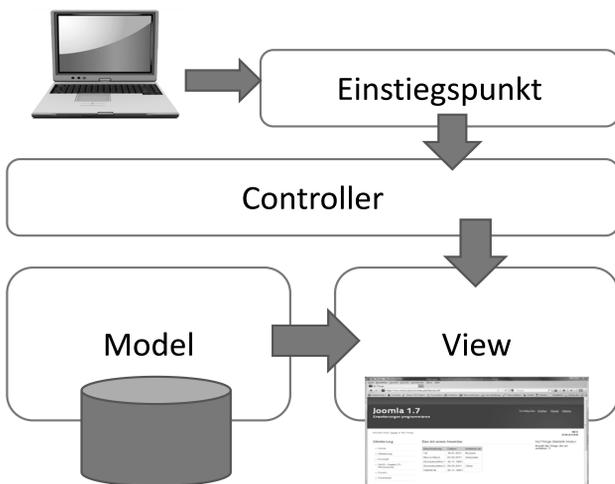


Bild 2.1: Model-View-Controller

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 Datenmodel 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 *administrator/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-Plattform. 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 Skripten 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 Komponentename 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 Streichholzsachtel 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.

DAEMON	
Science Fiction, spannend bis zur letzten Zeile	
Zustand:	wie neu
Wert:	16,50 €
Gewicht:	
Ist gerade verliehen.	

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.

```

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

```

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

Symbole

#__assets 150
 #__extensions 306
 #__menus 308
 \$_REQUEST 122
 \$! 278
 \$app 120
 \$cachable 159
 \$context 67
 \$key 99
 \$limitstart 67
 \$mainframe 120
 \$namespace 139
 \$param->get 296
 \$parent 374
 \$query 59, 343
 \$segments 343, 345, 346
 \$this 110
 \$type 374
 \$urlparams 159
 %d 70
 &\$article 67
 &\$params 67
 .ini 69
 __call() 111, 227
 __clone() 112
 __construct() 106
 __destruct() 106
 __FILE__ 61
 __get() 107
 __set() 107, 227
 <field>-Element 260, 320
 <fieldset>-Element 319
 <jdoc/>-Elemente 131
 <script>-Elemente 272

A

ableiten 103
 abstract 108

Abstrakte Klasse 104, 108
 access 150
 Access Control List 319
 Access Rules 152
 access.actions 175
 access.assetgrouplist 175
 access.assetgroups 175
 access.level 175
 access.usergroup 175
 access.usergroups 175
 access.xml 330, 331
 ACL 116, 152, 319
 addEntry() 255, 256
 addIncludePath() 171, 202
 admin 356
 Admin 24
 Administrator 24, 118
 Administratoren 151
 Administratorfunktionen 150
 Adminlisten 171
 advanced 294
 AJAX 132
 Aktualisieren 162
 Alias 297, 301, 344
 Alternatives Layout 63, 295
 Apache-Webserver 29
 Apfelkuchen 59
 API 115
 API-Referenz 118
 application 120
 Application 116
 Application Programming
 Interface 115
 application.php 121
 Applikation 120, 121
 archive 158
 array_merge 312
 Arrays 141
 article 67

ASC 298
 Asset 323
 asset_id 324
 Assets 152
 Aufrufreihenfolge 407
 Aufrufverfolgung 393
 Ausnahmefehler 225
 Ausnahmen 226, 229, 231
 Auswahlliste 175
 Authentifizierung 150
 authorise() 327, 328
 autocomplete-Attribut 219

B

Backend 24, 96, 118, 127, 167,
 187
 Backend-Zugang 150
 BadFunctionCallException 225
 BadMethodCallException 225
 basic 293
 Basisklassen 127
 Bearbeiten 150
 behavior.caption 176
 behavior.colorpicker 176
 behavior.core 176
 behavior.formvalidation 176
 behavior.framework 176
 behavior.highlighter 176
 behavior.keepalive 176
 behavior.modal 177
 behavior.multiselect 177
 behavior.noframes 177
 behavior.switcher 177
 behavior.tooltip 260
 behavior.tree 177
 behavior.uploader 177
 Behaviors 98, 176
 Beispieldaten 40
 Benutzer 148

- Benutzergruppe 327
 - Benutzer-ID 148
 - Benutzernamen 148
 - Benutzerverwaltung 148
 - Berechtigungen 319
 - Berechtigungs-Panel 323
 - Bezeichner 88, 128
 - Bibliotheken 27
 - Bildbearbeitung 163
 - Bilder manipulieren 163
 - Binärdaten 132
 - bind() 324
 - Blätter-Funktion 234, 237
 - Blättern 243
 - BOM 81
 - Breadcrumbs 160
 - Brotrumen 160
 - Browserweiche 272
 - Bugfixes 354
 - BuildRoute 342
- C**
- cache 159
 - Cache 116, 160, 166
 - Komponentenausgabe 159
 - Modulausgabe 160
 - Pluginausgabe 162
 - cachemode 161
 - calendar() 173
 - CamelCaps 44
 - captcha 163
 - categories 298
 - chaining 122
 - charset 370
 - Checkbox 196
 - checkToken() 140
 - class-Attribut 218
 - client 358, 362
 - CMS 23, 24, 96, 163, 180
 - CMS-Datenbank 117
 - Code 79, 393
 - Kommentare 42
 - code completion 407
 - Code Injection 245
 - code smells 233
 - Code-Archiv 159
 - Code-Editor 21
 - Codeervollständigung 401
 - collection 377, 378
 - com_config 305
 - com_content 67
 - com_content.article 67
 - com_finder 133
 - com_search 133
 - Community Builder 372
 - config 216, 281, 293
 - Config 116
 - config.xml 305, 319
 - configuration.php 29, 166
 - Container-Elemente 215
 - Content 67
 - Content Management System 23
 - Content-Plugins 66
 - context 67
 - Controller 31, 32, 36, 42, 57, 189
 - Copyleft 17
 - Copy-Paste-Programmierung 236
 - Core 26
 - Core Hack 26
 - CSS 267, 271, 276, 383
 - CSS4you 383
 - CSS-Klasse 71
- D**
- database 142
 - Database 116
 - date 173
 - Date 116
 - Dateibeschreibung 91
 - Dateien archivieren 158
 - Dateisystem 157
 - Daten formatieren 144
 - Daten laden 144
 - Datenbank 29, 59, 60
 - Normalisierung 249
 - Datenbanken 142
 - Datenbanksystem 60
 - Datenbasis 31, 44
 - Datenfilter 219
 - Datum 173
 - DBO 116
 - Debuggen 22, 393, 399
 - schrittweise 393
 - Debugger 22
 - Debug-Modus 181
 - de-DE 69
 - def() 119, 120
 - default.php 61, 62, 268, 276
 - default.xml 307
 - default-Attribut 218
 - defines.php 166, 167
 - Definitionsliste 265
 - deprecated 181
 - deprecated.php 181
 - Deprecation Log 181
 - DESC 298
 - description 281
 - description-Attribut 218
 - Destruktor 106
 - dispatcher 283
 - Dispatcher 154
 - DOCTYPE 352
 - Document 116
 - Dokument 130
 - Domain 23
 - DomainException 225
 - Dr. Web 383
 - driver 370
 - DRY-Prinzip 328
 - DS 61
 - DTD 352
 - Dummy-Installationsskript 37
- E**
- EasyCreator 381
 - echo 71
 - Eclipse 393, 394
 - Editor 116, 260
 - Editor-Plugins 66
 - Eigene bearbeiten 150
 - Eigenschaften 93, 102
 - Eingabefelder 171
 - Eingabeformular 216
 - Eingabekontrolle 121
 - Einstiegsskript 41, 188
 - Einzahl/Mehrzahl 190
 - Einzelansicht 268
 - en-EN 69

Entwicklung 393
 Entwicklungsprojekt 22
 Entwicklungsumgebung 22
 Entwurfsmuster 31, 112, 383
 environment 137
 Ereignis 29, 65
 Ereignisroutinen 153
 Ereignisse 153
 error 133
 error.php 133, 154
 Erstellen 150
 Erweiterungen 27, 43
 überprüfen 38
 Erweiterungsverwaltung 38
 escape 197
 event 153
 event handler 153
 Events 153
 Exception-Objekte 225
 Exceptions 225, 226, 231
 execute() 341
 executeComponent() 130
 extension 350, 354, 377
 extensions.manifest_cache 353
 eZ Publishing 398

F

Factory Pattern 116
 Farbauswahl 223
 fatal error
 Call to private... 112
 feed 132
 Feedparser 116
 Feeds 132
 Fehlerbehandlung 225
 Fehlercode 226
 Fehlercodes 230
 Fehlermeldung 226
 Fehlerseiten 133
 Fehlertext 226
 field 293
 field-Elemente 215
 fields 215, 216, 281, 293
 Attribute 217
 fieldset 260, 281, 293, 306
 fieldsets 215, 216
 files 356

files() 157
 filesystem 157, 158
 filter 134
 Filter 237, 257
 filter fields 237
 filter-Attribut 219
 Filterregeln 219
 folder 361
 folders() 157
 form 134, 216
 Form 199
 form field 304
 forms.php 219
 Formular 333
 Formularanfragen 347
 Formularansicht 127, 189, 190,
 192
 Formulare 134, 171, 215
 Formularelemente 171, 175
 Framework 25, 96
 Frameworks 394
 from 297
 Frontend 24, 40, 50, 96, 118,
 127, 167
 Frontend-Zugang 150
 FTP 158
 FTP-Befehle 158
 FTP-Client 158
 function display(\$tpl = null)
 283
 Funktionen 58
 Funktionsaufrufe 82
 Funktionsdefinitionen 82

G

General Public License 17
 get 282
 get() 119
 getAcl() 116
 getApplication() 116
 getCache() 116
 getClientInfo() 126
 getComponent() 129
 getComponentName() 126
 getConfig() 116
 getDate() 116
 getDbo() 59, 116

getDocument() 116
 getEditor() 116
 getFeedParser() 116
 getForm() 205
 getFormToken() 140
 getInstance() 43, 112, 113, 117,
 283
 getItems() 47
 getLanguage() 116
 getLayoutPath 61
 getListFooter() 242
 getMailer() 116
 get-Methode 116, 281
 getParams() 129
 getPath() 126
 getPlugin() 153
 getProperties() 119
 getQuery 59, 296
 getSession() 116
 getStream() 116
 getToken() 140
 getURI() 116
 getUser() 116
 getUserStateFromRequest()
 253
 Gewährleistungsanspruch 18
 Git 22, 182, 382
 Github 22, 159, 182
 globale Konstante 42
 GNU General Public License 17
 GNU/GPL 18, 19, 382
 GNU-Projekt 17
 GPL 17
 Grafikdatei 174
 group 285

H

Haltepunkt 393, 407, 408
 hash key 159
 hasToken() 140
 Hauptverzeichnis 24
 Helferlein 138
 helper.php 58, 61
 Helperklasse 236, 255, 328
 Heredocs 273
 hidden-Attribut 218
 Highlighting 407

- Hilfe
 - kontextsensitiv 393
- htaccess 167
- HTML 382
- HTML-Dokument 48, 131
- HTML-Formulare 140, 215
- HTML-Fragmente 134
- HTML-Paket 170
- HTML-Widgets 176
- I**
- Icon 192, 357
- icon-16-component 357
- IDE 22, 393
- if-Abfrage 85
- iframe 174
- image 163, 174
- images-Verzeichnis 170
- implementieren 105
- implode 283
- importPlugin 283
- importPlugin() 153
- index.html 41
- index.php 24, 131, 275, 276
- INI 141
- Inline-Kommentare 94
- input 201
- Input-Feld 281
- install 371, 372, 374
- install.sql 372
- Installationsskript 39
- installer 162
- Installer 24, 247
- installfile 373
- Installieren 162
- instanzieren 103
- int 297
- int_array 219
- Internet 24
- Internet Explorer 272
- Internetjargon 23
- Internetpräsenz 24
- InvalidArgumentException 225
- isEnabled() 129
- isRegistered() 171
- Item 199
- Itemid 344
- J**
- JAccess 150, 152
- JAccessRule 152
- JAccessRules 152
- JAdministrator 121
- JApplication 120
- JApplicationHelper 126
- JArchive 158
- JArrayHelper 138
- Java Development Kit 402
- JavaScript 223, 267, 271, 276
- JavaScript-Bibliothek 276
- JavaSE 402
- JBrowser 137
- JCaption 179
- JCategories 121
- JComponentHelper 129, 307, 316
- JController 42, 43, 44, 127, 191
- JControllerAdmin 191, 192
- JControllerForm 191, 192
- JDatabase 142, 143
- JDatabaseException 142
- JDatabaseQuery 45, 143, 145
- JDate 138
- JDispatcher 283
- JDK 402
- JDocument 130, 271
- JDocumentFeed 132
- JDocumentHTML 131
- JDocumentRaw 133
- JError 225
- JEvent 153
- JEXEC 42
- JFactory 59, 116, 117, 296, 299
- JFactory::getUser() 327
- JFeedEnclosure 132
- JFeedItem 132
- JFile 157, 158
- JFilterInput 134
- JFilterOutput 135
- JFolder 157
- jform 338
- JForm 134, 215, 259
- JFormField 134, 259
- JForms-Konzept 204
- JFTP 158
- JHtml 98, 170, 258
 - Helperklassen 172
 - Verwaltung 171
- JHtmlGrid 197, 242
- jimport() 42, 97
- JInput 121, 123, 124
- JInstallComponent 374
- JInstallModule 374
- JLanguage 136, 137
- JLoader 97
- JMail 156
- JMailHelper 156
- JMenu 121
- JMenuAdministrator 121
- JMenuSite 121
- JModel 44, 127
- JModelAdmin 205
- JModelForm 203
- JModelList 45, 47, 203
- JModule 130
- JModuleHelper 61, 130
- JObject 119
- John Doe 364
- Join 250
- JOIN 297
- Joomla 383
- Joomla! CMS 26
- Joomla! Platform 25, 26
- Joomla-API 115, 384
- Joomla-Document 130
- JPATH 165
- JPATH_BASE 166
- JPATH_CACHE 166
- JPATH_COMPONENT 167
- JPATH_COMPONENT_ADMINISTRATOR 167
- JPATH_COMPONENT_SITE 167
- JPATH_CONFIGURATION 166
- JPATH_INSTALLATION 166
- JPATH_LIBRARIES 166
- JPATH_MANIFESTS 166
- JPATH_PLATFORM 166
- JPATH_ROOT 157, 166
- JPATH_THEMES 166
- JPathway 121

- JPathwaySite 121
 - JPlugin 67, 153
 - JpluginHelper 283
 - JPluginHelper 153, 283
 - JRegistry 141, 307, 312
 - JRequest 122, 124, 137
 - JResponse 137
 - JRoute 52, 342
 - JRouter 121, 125
 - JRouterAdministrator 121
 - JRouterSite 121, 125
 - JSite 121
 - JSON 132, 141, 352
 - JSON-Format 306
 - JString 135
 - JStringNormalise 135
 - JSubMenuHelper 255, 256
 - JTable 146, 202, 311
 - bind 311
 - JText 70, 76, 194
 - JToolBarHelper 121, 193, 194
 - JURI 137
 - JUser 148, 152, 327
 - JUserHelper 152
 - JUser-Objekt 150
 - JUtility 138
 - JView 45, 127, 193
- K**
- Kalender 173, 223, 260
 - Kapselung 107
 - Kategorie-Filter 253
 - Kategorien 247
 - Kategorienliste 299
 - Klammern 84
 - Klasse 96, 102
 - Klassen 88, 93, 95, 97
 - Klassenmethode 99
 - Klassennamen 89
 - Kommentare 91, 234
 - Inline-Kommentare 94
 - Komponente 27, 28, 36
 - Einstiegsskript 41
 - Frontend 36
 - Name 45
 - Steuerung 42
 - Komponentenausgabe cachen 159
 - Komponenten-Events 276
 - Komponenten-Layout 276
 - Komponentenname 41
 - Komponentenparameter 305
 - Konfigurationspanel 306
 - Konfigurationsparameter 216
 - Konfigurieren 150
 - Konfigurierungs-Icon 323
 - Konkrete Klasse 104, 108
 - Konstantennamen 88
 - Konstruktor 106, 203, 244
 - Kontrollkästchen 221
 - Kontrollstrukturen 85
 - Konventionen 79
- L**
- label 201
 - Label-Attribute 217
 - Laden von Ressourcen 171
 - Länderkennung 70
 - language 69, 70, 136
 - Language 116
 - languages 360
 - Laufzeit 393
 - layout 295
 - Layout 46, 267, 268, 276
 - Layout-Overrides 267
 - LDAP 158
 - LengthException 225
 - libraries 364
 - libraryname 365
 - Lightweight Directory Access Protocol 158
 - LIMIT 298
 - limitstart 67
 - link 174
 - Listen 171
 - Listenansicht 127, 189, 190, 192, 341
 - listFolderTree() 157
 - Literale 273
 - Lizenz 17
 - Lizenzbedingungen 17
 - loadAssoc() 144
 - loadAssocList() 145, 299
 - loadColumn() 145
 - loadNextObject() 145
 - loadNextRow() 145
 - loadObject() 144
 - loadObjectList() 144, 145
 - loadResult() 60, 145, 299
 - loadRow() 145
 - loadRowList() 145
 - log 154
 - LogicException 225
 - Löschen 150
- M**
- mail 155
 - Mailer 116
 - Manager 151
 - manifest.xml 275, 352, 354, 371
 - Martin Fowler 233
 - media 375
 - media-Element 275
 - media-Verzeichnis 170, 177, 275
 - Mehrfachansicht 341
 - Mehrfachvererbung 109
 - Mehrsprachigkeit 70, 165
 - Menü 160
 - menu.php 121
 - Menüpunkt 307
 - Menütyp 50, 307
 - Metadatei 362
 - Metadaten 171
 - method 353
 - Methoden 58, 88, 93, 102
 - get 47
 - methods.php 76
 - Microsoft 394
 - Migration 182
 - MIME-Typ 133
 - Mini-Templates 273
 - mod 59
 - mod_articles_latest 160
 - mod_breadcrumbs 160
 - mod_menu 160
 - Model 31, 36, 44, 57, 58
 - Model-Klasse 129
 - Model-View-Controller 31, 32, 41

- Modulausgabe cachen 160
 - Module 27, 28, 293
 - Basisoptionen 295
 - Erweiterte Optionen 295
 - moduleCache() 161
 - moduleclass_sfx 295
 - modulelayout 295
 - Modulklassensuffix 295
 - Modultyp 358
 - Mootools 171, 177, 179
 - multilang 165
 - multiple-Attribut 218
 - MVC 31, 57, 62, 112, 383
 - MVC-Entwurfsmuster 36
 - MVC-Klassen 127, 128
 - MySQL 29, 30, 61, 382
 - MySQL-Datenbank 44
 - MySQL-Server 29
- N**
- name-Attribut 216, 217, 218
 - Namen 88
 - Namenskonventionen 45
 - Namenskonzepte 95, 127
 - Nested Sets 147
 - NetBeans 21, 402
 - Editor 407
 - Installation 402
 - Projekt 405
 - Start 404
 - Syntaxprüfung 408
 - Neueste Artikel 160
 - new 112
 - News 132
 - Nikolai Plath 381
 - Normalisierung 250
- O**
- object.php 119
 - Objekte 88, 101, 141
 - Objekteigenschaften 142
 - Objektorientierte
 - Programmierung 101, 103, 383
 - Objektverkettung 122
 - oddball-Lösung 236
 - öffentliche Ressourcen 170
 - Office 2010 394
 - Offline-Zugang 150
 - onAfterDispatch 276
 - onAfterRoute 276
 - onchange-Attribut 218
 - onContentBeforeDisplay 276
 - onContentPrepare 67, 278, 282
 - onMythingsBeforeDisplay 283
 - onMythingsDisplay 276, 285
 - OOP 101, 113
 - Open Source 19
 - opensearch 133
 - option 294
 - Optionen 304, 305, 319, 323
 - optiongroup 175
 - Optionselement 175
 - order 298
 - ORDER BY 298
 - OutOfBoundsException 225
 - OutOfRangeException 225
 - OverflowException 225
 - Override 63
- P**
- Paamayim Nekudotayim 109
 - packages 368
 - Pakete 115
 - Parameter 303
 - Parameterverwurstungs-
maschine 303
 - params 67, 281, 282, 293, 306
 - params->get 282
 - Parse 393
 - ParseRoute 345
 - pathway.php 121
 - Pfadangaben 165
 - Pfade 97
 - PHP 80, 106
 - php.ini 403
 - PHPEdit 21, 381, 394
 - PHP-Editoren 381
 - PHP-Klassen 115
 - phpMyAdmin 39
 - PHP-Tag 80
 - Kurzform 80
 - PHPUnit 381
 - Plattform 25, 96, 180
 - platform.php 166
 - Plattform 25
 - plg 67
 - plugin 153
 - Plugin 359
 - Pluginausgabe cachen 162
 - Plugin-Klasse 67
 - Plugins 27, 29, 65, 153, 277
 - Plugintyp 67
 - plural 72
 - populateState() 244
 - position 364
 - Positionsangaben 363
 - POST 140
 - postflight 374
 - Postversand 155
 - Prado 398
 - Präfix 128
 - Präsentation der Daten 31
 - preferences 304
 - preflight 374
 - preg_replace 278
 - Primärschlüssel 39
 - printf 70, 71
 - private 109, 112
 - Programmblöcke 81
 - Programmdateien 170
 - Programmierkonventionen 79, 233
 - Programmierschnittstelle 115
 - Programm Quelltext 79
 - Programmsteuerung 31
 - Projektverwaltung 395
 - Protokolle 154
 - public 120
 - Punktnotation 193
- Q**
- Quelltext 79
 - queries 371
 - query 59, 371
 - Query 246
 - Query-Objekt 145
 - Quick-Icons 164
 - quote() 245

R

Radiobox 175
 Radio-Buttons 221
 raw 132, 219
 Raw-Dokumente 133
 readonly-Attribut 218
 Redirect 133
 Refactoring 233, 235
 Regeln 79
 regex 278
 register() 172
 registry 141
 Registry-Objekte 141
 Reguläre Ausdrücke 278
 renderComponent() 129
 require_once 61, 300
 Reservierte Wörter 89
 Ressourcen 173
 Rohdaten 132
 Root 24
 Root Asset 152
 Root-Verzeichnis 24
 Router 341, 342
 Joomla!-Application-Router 342
 Komponenten-Router 342
 router.php 121, 341
 Routing 341
 rules 219

S

safehtml 219
 schema 165
 schemapath 372
 Schlüsselnamen 125, 142
 Schlüsselwert 159
 Schlüsselwort 99
 Schnellstart-Symbole 164
 Schnittstellen 115
 Schrotflinten-Chirurgie 235
 script 174
 Script-Elemente 171
 scriptfile 373
 search engine friendly URL 125
 search engine optimization 125
 section 330
 SEF 125, 341

select 297
 select.booleanlist 175
 select.genericlist 175
 select.groupedlist 175
 select.integerlist 175
 select.option 175
 select.options 175
 select.radiolist 175
 Selectlist 260
 Select-Liste 258
 self 112
 SelfHTML 382
 SEO 125
 separation of concerns 191
 serialize() 141
 server_utc 219
 Session 116, 139, 237
 set() 120
 setFormatOptions() 172
 setQuery 298
 shotgun surgery 235
 Sicherheit 245
 Sicherheitsmaßnahme 41, 198
 Singleton 112, 383
 Singleton-Entwurfsmuster 116
 Singleton-Objekt 117
 site 356
 Site 24
 Sitzungsverwaltung 139
 size-Attribut 218
 Slider 263, 264
 Sliderpanel 263
 sliders.end 176
 sliders.panel 176
 sliders.start 176
 Slug 344
 Sortieren 237
 Sortierfeld 245
 Sortierrichtung 242, 245
 Sortierung 234, 238
 Sourceforge 382
 Sprachdateien 69, 70, 136, 265, 317
 Sprachen 136
 Sprachpaket 136
 Sprachpakete 27
 Sprachschlüssel 70, 331

Sprechende Namen 87
 sprintf 71
 sql 373
 SQL 30, 61, 142, 143, 382
 SQL Injections 382
 SQL-Abfrage 59
 SQL-Schemata 165
 Stammverzeichnis 24
 Standardlayout 268
 Standardregeln 150
 Standards 79
 Standardsprache 137
 state 237, 258
 static 108
 Statische Klassen 105, 108
 Status ändern 150
 Statusinformationen 245
 Steuerung 42
 Stream 116
 string 135
 stringURLSafe 135
 Structured Query Language 30, 142
 Style-Elemente 171
 stylesheet 174
 stylesheets 267
 Sub-Klassen 98
 submenu 357
 Submenü 247, 248, 256
 submit-Button 242
 Suchergebnisse 133
 Suchfilter 243
 Suchmaschinenfreundliche URL 125, 341
 Suchmaschinenoptimierung 125
 Such-Plugin 133
 Super Admin 150
 Super Users 151
 Symphony 398
 Syntaxprüfung 408
 sys.ini 72
 System-Plugins 66, 276
 Systemumgebung 137

T

Tabellen 146

- Tabellenansicht 268, 341
 - Tabellenpräfix 29
 - tabs.end 176
 - tabs.panel 176
 - tabs.start 176
 - task 193, 210
 - tel 219
 - Templates 27, 166, 276
 - Ternäre Kurzschreibweise 86
 - Texte 135
 - Textkodierung 81
 - Titel 344
 - tmpl 57, 62
 - token 338
 - Token 140, 198
 - Toolbar 177, 192, 193, 255, 319
 - toolbar.php 121
 - tooltip 173
 - Tooltip 173, 196, 260
 - Topliste 296
 - triggern 283
 - try-catch 339
 - type
 - description 260
 - input 260
 - type-Attribut 218
 - type-Attribute 217
- U**
- UML 104
 - Umleitung 133
 - Unicode Transformation
 - Format 20
 - Unicode-Format 81
- U**
- Unified Modeling Language 104
 - Uniform Resource Locator 23
 - uninstall 371, 374
 - uninstall.sql 373
 - uninstallfile 373
 - unregister() 172
 - unserialize() 141
 - unset 219
 - Unsortierte Liste 265
 - update 374, 378
 - Update 372
 - updater 162
 - Update-Server 349, 376
 - upgrade 353
 - URI 116
 - url 219
 - URL 23, 341, 343, 347
 - URL-Design 342
 - user 148
 - User 116, 247
 - user_utc 219
 - UTF 383
 - UTF-8 81, 350, 370
 - UTF-8-BOM 383
 - utilities 138
 - utility.php 138
- V**
- Validierung 134
 - Variablennamen 88, 236
 - Vererbung 109
 - version 354
 - Versionskontrolle 22
 - Versionskontrollsystem 182
 - Versionsverwaltungssystem 22
 - View 31, 36, 44, 46, 50, 57, 90
 - View-Klasse 129
 - Virtuelle Methoden 104
 - Visual Studio 394
- W**
- W3C 383
 - Waterproof 394
 - Web 24
 - Webauftritt 24
 - Web-Magazin 383
 - Webseite 23
 - Website 24
 - Werkzeugleiste 192
 - where 298
 - Widgets 98, 170, 173
 - WikiLeaks 19
 - Wikipedia 383
 - wohlgeformt 352
- X**
- XDebug 228, 399, 403
 - XML 132
- Z**
- Zielcontroller 193
 - Zugriffsberechtigung 319
 - Zugriffsrechte 150, 152
 - Zugriffsregeln 320, 323, 330, 332
 - Zwischenspeicher 159

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:

- Joomla-Grundwortschatz
- Erste Ausgabe im Frontend
- Unser erstes Modul
- Das erste Plugin
- Sprachen
- Standards, Regeln, Konventionen
- Objekte & Co
- Die Joomla-API – Eine Einführung
- Unsere Komponente: Backend
- Formulare cleverer: JForm
- Fehlerbehandlung
- Alles bleibt anders – Exkurs Refactoring
- Filter – Sortierung – Pagination
- Kategorien, User und JForms
- CSS – Kosmetik fürs Frontend
- Plugins – Arbeiten im Untergrund
- Module: Daten immer anders
- Die Komponente wird konfigurierbar
- Wer darf was? Zugriffsrechte
- Formular im Frontend
- Routing und SEF
- Installer
- Events (Plugins)
- Entwicklungsumgebungen

Über die Autoren:

Axel Tüting ist Fachinformatiker, Anwendungs-entwicklung. 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