



Fachhochschule Köln
Cologne University of Applied Sciences

Fachhochschule Köln
Fakultät für Informatik und Ingenieurwissenschaften

W I - P R O J E K T

Wordpress Plugin-Entwicklung

Die Entwicklung eines Plugins zur Mentorensuche für das Programm

Mentoring4Excellence

Vorgelegt an der
Fachhochschule Köln,
Campus Gummersbach
im Studiengang
Wirtschaftsinformatik

Ausgearbeitet von:
AMLING, TIMO - 1107
TISSEN, ANATOL - 1107

Projektbetreuung: Frau Prof. Dr. Heide Faeskorn-Woyke
Fachbetreuung: Herr B.Sc. Ludger Schönfeld

Gummersbach, im März 2013

Inhaltsverzeichnis

1	Einleitung	1
1.1	Zielsetzung	1
1.2	Motivation	1
1.3	Inhalt	1
1.3.1	Kapitelübersicht	1
1.3.2	Aufbau	2
2	Vorbereitung	3
2.1	Was ist Wordpress?	3
2.2	Was ist ein Plugin?	3
2.3	Benötigte Werkzeuge	4
2.3.1	Webserver	4
2.3.2	Wordpress	4
2.3.3	Editor	5
2.3.4	FTP Client	5
2.4	Programmierstandards	5
3	Aufbau des Plugins	7
3.1	Allgemeines	7
3.2	Aktivierung des Plugins	7
3.3	Information Header	8
3.4	Benutzerdefinierte Funktionen	9
3.4.1	Das erste eigene Plugin	9
3.4.2	Debug-Modus	10
3.5	Hooks	11
3.5.1	Referenzieren mittels <code>add_action()</code> und <code>add_filter()</code>	12
3.5.2	Hook-Anwendungsbeispiele	12
4	Menüerstellung	16
4.1	Wozu dient ein Menü?	16
4.2	Erstellen von Top-Level-Menüs	16
4.2.1	Beispiel Top-Level-Menüs aus Mentoren-Suche	17
4.3	Erstellen von Submenüs	18
4.3.1	Beispiel Untermenü aus Mentoren-Suche	19
4.4	Integration in bestehende Menüs	21
4.4.1	Beispiel Integration von Submenüs	22
4.5	Ausblick	24
5	Shortcodes	25
5.1	Was ist ein Shortcode?	25
5.2	Shortcodes verwenden	25

5.3	Shortcode-API	27
6	Datenbankzugriff über Plugin-API	29
6.1	Die Datenbankschnittstelle WPDB	29
6.1.1	Syntax und grundsätzliche Beispiele	29
6.1.2	Beispiele aus dem Plugin Mentoren-Suche	33
6.2	Verhindern von SQL Injections	36
7	Formular	39
7.1	Was ist ein Formular?	39
7.1.1	POST-Methode	39
7.1.2	GET-Methode	39
7.2	Wordpress-Formulare	40
7.3	Beispiel Formular aus Mentoren-Suche	40
7.3.1	Beispiel für PHP-Formular	40
7.3.2	Beispiel für Wordpress-Formular	41
8	Internationalisierung und Lokalisierung	45
8.1	Einführung	45
8.2	Lokalisierung	46
8.3	Laden einer Textdomäne	48
8.4	Anlegen der Übersetzung	49
8.5	Konfiguration der Wordpress-Sprache	53
9	Pluginverwendung in Wordpress	55
9.1	Unterscheidung der Installationstypen	55
9.2	Installation	55
9.2.1	Intallationsroutinen	56
9.3	Deinstallation	59
9.3.1	Deinstallieren	61
10	Zusammenfassung und Ausblick	64

Abbildungsverzeichnis

1	Pluginübersicht im Pluginmenü	8
2	Erfolgreiche Meldung der Aktivierung	8
3	Printbefehl-Ausgabe	10
4	Printbefehl mit aktivierten Debugger	11
5	Ausgabe Top-Level-Menü-Funktion	18
6	Ausgabe Submenü-Funktion	21
7	Übersicht Admin-Bereich von Wordpress	21
8	Eigenes Untermenü in vorhandene Menüstrukturen eingebunden	23
9	Shortcode in Seite einbinden	25
10	Ergebnis des eingebundenen Shortcodes	26
11	Beispielformular mittels PHP	41
12	Beispielformular mittels do_action	42
13	Katalogoptionen - Reiter „Projektinfo“	50
14	Katalogoptionen - Reiter „Pfade“	50
15	Katalogoptionen - Reiter „Schlüsselwörter“	51
16	Katalogoptionen - Dialog ”Speichern unter“	51
17	Katalogoptionen - Meldung ”Übersicht aktualisieren“	52
18	Poedit - Übersetzungen erstellen	52
19	Deutsche Spracheinstellung (de_DE) [Standard]	54
20	Englische Spracheinstellung (en_US)	54

Tabellenverzeichnis

1	Vergleich Action- und Filter-Hooks	12
---	--	----

Abkürzungsverzeichnis

API Application Programming Interface

CMS Content Management System

CSS Cascading Style Sheets

FTP File Transfer Protocol

GPL General Public Licence

HTML HyperText Markup Language

PHP PHP: Hypertext Preprocessor

POT Portable Object Template

SQL Structured Query Language

URI Uniform Ressource Identifier

URL Uniform Resource Locator

XAMPP Cross-plattform, Apache, MySQL, PHP, Perl

Quellcodeverzeichnis

1	Information Header	8
2	Erste Printausgabe	9
3	Verbesserte Printausgabe	10
4	Debug-Modus aktivieren	10
5	Beispiele für <code>add_action()</code>	12
6	Beispiele für eigene <code>add_action()</code>	13
7	Beispiele für <code>add_filter()</code>	13
8	Funktion <i>hello_dolly</i> mit Input-Variable	14
9	Beispiele für <code>add_option()</code>	15
10	Syntax der Top-Level-Menü - Funktion	16
11	Beispiel Mentoren-Suche Top-Level-Menü	17
12	Syntax der Submenü - Funktion	18
13	Beispiel Mentoren-Suche Submenü	19
14	Syntax zum integrieren von Menüs in bereits vorhandene	22
15	Beispiel hinzufügen eines Untermenüs zu vorhandenen Menüs	23
16	Shortcode hinzufügen	26
17	Gekürzte Beispielfunktion <code>showSearchArea</code>	26
18	Allgemeine Syntax	29
19	Selektion einer Variablen	30
20	Selektion einer Zeile	31
21	Selektion von Spalten	31
22	Insert von Spalten	32
23	Update von Spalten	33
24	Beispiel <code>SELECT</code> von Zeilen aus <code>mentoren-plugin</code>	34
25	Beispiel <code>INSERT</code> von Spalten aus <code>mentoren-plugin</code>	34
26	Beispiel <code>DROP</code> -Anweisung aus <code>mentoren-plugin</code>	35
27	Beispiel für SQL-Injection-Verhinderung	36
28	Verhindern einer SQL-Injection aus <code>mentoren-suche</code>	37
29	Beispiel Formular in Wordpress	40
30	Beispiel Formular in Wordpress	42
31	Ausgeführte Action der <code>do_action</code> -Anweisung	43
32	Aufgerufene Funktion der Action	43
33	Beispiel für die Verwendung von <code>_e()</code>	46
34	Beispiel für die Verwendung von <code>__()</code>	47
35	Beispiel für die korrekte Einbettung von Variablen mit der <code>sprintf()</code> -Funktion	47
36	Laden einer Textdomäne	48
37	<code>wp-config.php</code> – Die in PHP definierte Konstante <code>WPLANG</code>	53
38	Beispiel für einen Activation-Hook	56
39	Beispiel für einen Activation-Hook	57
40	Prüfung der Mindestversion in Wordpress	57
41	Prüfen der Wordpressversion	58

42	Menüerstellung im Adminbereich	58
43	Beispiel dbDelta	59
44	Beispiel Syntax Deactivation-Hook	60
45	Deactivation-Hook aus Mentorensuche	60
46	Funktion Plugin Uninstall	60
47	Beispiel Uninstall.php-Datei	61
48	Syntax Uninstall-Hook	62
49	Beispiel Uninstall-Hook	62

1 Einleitung

Dieses Tutorial soll anhand von Erklärungen und Beispielen eine Übersicht über die Pluginentwicklung für ein Wordpresssystem geben.

Dabei wird anhand von verschiedener Fachliteratur auf einzelne Themengebiete eingegangen und diese dann im Detail behandelt.

Die zentrale Fragestellung lautet dabei: Wie entwickel ich ein Plugin in Wordpress. Dabei wird sich am Beispiel an dem Projekt "mentoren-suche", welches für das Programm *Mentoring4Excellence* programmiert wurde, orientiert.

1.1 Zielsetzung

Ziel dieser Arbeit ist dem Leser eine Übersicht über die Techniken, die für die Pluginentwicklung benötigt werden, zu geben.

Dabei lässt sich aufgrund der beschränkten Zeit nicht auf jedes Detail genauestens eingehen. Kenntnisse in den Programmiersprachen PHP, HTML und CSS sind Voraussetzung, um die einzelnen Anweisungen zu verstehen.

1.2 Motivation

Das Tutorial wurde im Rahmen des WI-Projektes von den beiden Studenten Timo Amling und Anatol Tissen geschrieben, welche zuvor auch das dazugehörige Plugin entwickelt und dokumentiert haben.

Daher haben die Autoren die ersten Praxiserfahrungen mit dem Umgang professioneller Pluginentwicklung kennengelernt und möchten mit diesem Tutorial ihre Erfahrungen und Wissen einem breiteren Spektrum an Informatiker, Studenten und Informatikinteressierte weitergeben. Anbei soll an dieser Stelle auch erwähnt werden, dass das Kapitel 8 von Herrn Ludger Schönfeld geschrieben wurde. Wir danken ihn an dieser Stelle vorab für seine konstruktive Mitarbeit.

1.3 Inhalt

In diesem Unterkapitel wird neben einer Übersicht über die einzelnen Kapitelinhalte auch der Aufbau dieses Tutorials begründet.

1.3.1 Kapitelübersicht

Nachdem in diesem ersten Kapitel Einleitung unter anderem die Motivation dieses Tutorials geklärt wurde, wird im zweiten Kapitel Vorbereitung eine Übersicht über Wordpress behandelt. Dabei wird sich mit dem Begriff des Plugins, sowie die benötigten Entwicklungswerkzeuge und Standards für die Programmierung eingegangen.

Anschließend wird in Kapitel Aufbau des Plugins die grundsätzliche Struktur eines jeden Plugins, was Funktionen sind und wie diese geschrieben werden, beschrieben.

Danach kommt das 4te Kapitel, Shortcodes. Dieses Kapitel widmet sich den kleinen Codeschnipseln, welche an beliebiger Stelle im Seiteninhalt eingebaut werden können. Dabei wird sich auch der Shortcode-API angesprochen.

Im Kapitel Datenbankzugriff über Plugin-API werden alle technischen Aspekte, die benötigt werden um mit einer Datenbank über ein Plugin zu kommunizieren beschrieben.

Anschließend wird im Kapitel Pluginverwendung in Wordpress die zwei wesentlichen Aspekte eines Plugins aus Anwendersicht besprochen: Die Installation und Deinstallation eines Plugins. In Kapitel 7 wird sich zunächst dem Begriff des Formulars gewidmet, anschließend wird beschrieben, wie Formulare erstellt und verwendet werden.

Nachdem die rudimentären technischen Aspekte in den vorherigen Kapiteln erläutert wurden, widmet sich das Kapitel Internationalisierung und Lokalisierung der Sprachanpassung für Plugins. Nach einigen Begrifflichkeiten, wird anschließend gezeigt, wie Übersetzungen erstellt und auf den neusten Stand gehalten werden. Natürlich spielt auch das Laden einer solchen Textdomain in diesem Kapitel eine Rolle.

Das letzte Kapitel Zusammenfassung und Ausblick beschreibt zusammenfassend die einzelnen oben genannten Schritte der Pluginentwicklung.

1.3.2 Aufbau

In diesem kleinen Abschnitt geht es um die grundsätzliche Frage nach dem strukturellen Aufbau dieses Tutorials.

Wie sich erkennen lässt, behandeln die ersten drei Kapitel inhaltlich die ersten Schritte in der Plugin-Entwicklung. Dabei wird immer spezifischer von der Klärung des Begriffs *Plugin* über entsprechende Werkzeuge und Normen bis zur Aktivierung und den ersten Schritten zum eigenen Plugin beschrieben.

Die Kapitel vier bis acht dienen der tieferen und komplexeren Entwicklung. Dort werden spezielle Techniken angesprochen. Angefangen mit dem Verwenden von Shortcodes um das Plugin auf eine Wordpress-Seite einzubinden, wird anschließend Aspekte der Datenbankkommunikation besprochen. Auch Sicherheitsaspekte wie beispielsweise bestimmte Angriffstechniken auf die Datenbank angehen, werden hier beachtet.

Weiter geht es dann mit Formularen in Kapitel sieben. Neben einigen grundsätzlichen Methoden und Begriffen, wird anschließend gezeigt, wie Formulare in Wordpress erstellt werden. Dies wird dann zum Abschluss des Kapitels mit Beispielen aus der Mentoren-Suche abgerundet.

Im achten Kapitel - der Internationalisierung und Lokalisation - geht es um die Sprachanpassung eines Wordpress-Plugins. Dabei werden nach einer kurzen Einführung in das Thema einige Techniken gezeigt, sich Übersetzungen erstellen lassen und diese im einzelnen konfiguriert werden. wie.

Im neunten Kapitel werden dann die Installation und Deinstallation aus Anwendersicht beschrieben, um das Tutorial abzuschließen und aus der Entwicklersicht wieder auf die Anwendersicht zu kommen.

Zum Schluss werden dann im Kapitel neun die wichtigsten Fakten aufgelistet und dient so als Rückblick auf den Inhalt. Weiterhin wird auch hier ein Ausblick gegeben. Dies rundet dann das Tutorium sinnvoll ab.

2 Vorbereitung

In diesem Kapitel werden grundlegende Begriffe für den Umgang mit einem CMS wie Wordpress erklärt und erläutert. Darauf aufbauend wird in die Definition eines Plugins gegangen. Zum Abschluss werden anschließend die wichtigsten Werkzeuge zur Pluginentwicklung vorgestellt.

2.1 Was ist Wordpress?

Laut Bondari und Griffiths¹ lässt sich Wordpress als CMS beschreiben, welches hauptsächlich für Blogs genutzt wird. Dieses Projekt besteht seit 2003 und hat sich seitdem als eines der weit verbreiteten Blogging-Software entwickelt.

Als CMS lässt sich ein System beschreiben, mit welchem sich Daten auf einer Website dynamisch verwalten lassen. Drei wichtige Punkte sind hierbei zu beachten, um von einem vollwertigen CMS zu sprechen. Dies ist zum einen die Tatsache, dass Benutzer nicht tief in die Programmierung eingreifen müssen, um die Website im Aufbau zu verändern.²

Im Wikipedia-Eintrag für Wordpress³ finden sich die anderen beiden Punkte. Diese lauten zum einen, dass Benutzerrollen und deren Rechte verwaltet werden können. Es gibt also eine Möglichkeit, verschiedenen Benutzer, unterschiedliche Rechte zuzuteilen. Gleichzeitig ist es aber auch möglich, dass mehrere Benutzer gleichzeitig Inhalte bearbeiten und anpassen können.

Der letzte Punkt handelt von der Erweiterbarkeit eines CMS, nämlich die Möglichkeit, externe Plugins in das System einzubinden. All die aufgezählten Punkte werden von Wordpress erfüllt, sodass sich von einem vollwertigen CMS sprechen lässt.

Die Software ist laut wordpress.org⁴ unter der GPL lizenziert. Dies bedeutet, dass Wordpress frei von jedem verwendet werden darf, sei es im privaten oder kommerziellen Bereich. Dabei wird die Software und der Quellcode kostenfrei zur Verfügung gestellt und kann von jedem verwendet und weiterentwickelt werden.

Von Hause aus bietet Wordpress nur eine begrenzte Funktionalitäten, welche jedoch mit Plugins erweiterbar sind. Diese werden oft von engagierten Programmierern aus der Wordpressgemeinschaft programmiert, um wünschenswerte Funktionalitäten einzubauen und so die Software zu erweitern.⁵

2.2 Was ist ein Plugin?

Plugins sind Erweiterungen für Wordpress. Sie dienen dazu, das System mit bestimmten Funktionalitäten zu erweitern und liegen im Plugin-Verzeichnis `/wp-content/plugins/`. Wordpress unterscheidet nicht wie andere CMS in bestimmte Module oder Addons. Für Wordpress ist jeder Quellcode, der das System erweitert ein Plugin.

Um diese Plugins zu programmieren, bietet Wordpress eine eigene Schnittstelle für Programmierer an: das sogenannte API. Hiermit können Programmierer mit dem System über bestimmte Funktionen und Variablen kommunizieren. Der größte Teil der Funktionen ist prozedural in

¹Vgl. BONDARI/GRIFFITHS (2011), Seite 7.

²Vgl. HETZEL (2012), Seite 24.

³Vgl. WIKIPEDIA.DE (abgerufen am 16.08.2012).

⁴Vgl. MULLENWEG (abgerufen am 13.08.2012).

⁵Vgl. HETZEL (2012), Seite 22.

PHP, HTML und teils in CSS geschrieben. Da es sich um sehr viele Funktionen handelt, muss bei der eigenen Programmierung darauf geachtet werden, dass Namen für Funktionen nicht doppelt vorkommen. Dies könnte zu Namenskollisionen mit anderen Funktionen führen. Diese Thematik wird tiefer im Unterkapitel Programmierstandards behandelt.

Zum Schluss dieser Einführung wird noch der Begriff *eventorientiert* erklärt und erläutert. Dies ist die Architektur, nach der Plugins in Wordpress arbeiten⁶.

Kurz gesagt bedeutet dieser Begriff, dass über sogenannte *hooks*, Kontakt zur Wordpress-Schnittstelle aufgebaut werden kann. Dabei wird in einem bestimmten Bereich der Seite entweder eine neue Funktion eingebaut (dann ist es ein *action_hook*) oder bestimmter Inhalt gefiltert (dann ist es ein *filter_hook*).⁷

Hooks sind also ein Platz auf einer Seite, an der sich frei übersetzt Quellcode "aufhängen" lässt. Die eventorientierte Architektur bietet also die Möglichkeit, erst bei einem Klick auf ein bestimmtes Objekt, einen hook auszulösen. In HTML wäre ein hook die Funktion hinter dem Button "Button anklicken"⁸. Tiefer wird auf die eventorientierte Architektur im Kapitel 3.4 eingegangen.

Soviel zu dem Begriff des Plugins. Im 2.3 geht es nun um die Entwicklungsvorbereitung. Dabei werden verschiedene Umgebungen und Programme vorgestellt.

2.3 Benötigte Werkzeuge

Hier werden nun die Werkzeuge vorgestellt, welche für die Entwicklung und dem eigentlichen Umgang mit Wordpress benötigt werden. Hierzu wird auf der einen Seite Wordpress, aber natürlich auch eine Umgebung für die Programmierung, zum Testen und Hochladen benötigt.

2.3.1 Webserver

Der Webserver muss laut offiziellen Vorgaben⁹ von Wordpress mindestens PHP 5.2 und MySQL 4.1.2 SQL für die von uns verwendete Wordpressversion 3.2 unterstützen.

Wir haben dafür eine lokale Installation mittels XAMPP verwendet. XAMPP kann unter <http://www.apachefriends.org/en/xampp.html> heruntergeladen werden.

2.3.2 Wordpress

Wordpress bietet unter <http://wordpress.org/download> die aktuellste Version seines Softwarepakets an. Dieses ist gepackt und muss anschließend nur noch auf den Webserver im *htdocs*-Ordner entpackt werden. Anschließend wird die URL <http://localhost/wordpress/> oder <http://deine-domain/wordpress/> aufgerufen und die entsprechenden Datenbank-Details eingegeben werden.¹⁰

Da die Pluginprogrammierung hier im Mittelpunkt steht, wird nicht weiter auf dieses Thema eingegangen. Für unser Plugin ist es erforderlich eine Version über 3.0 zu nehmen, da es ansonsten zu Fehlern kommen kann.

⁶Vgl. BONDARI/GRIFFITHS (2011), Seite 9 - 10.

⁷Vgl. HETZEL (2012), Seite 274.

⁸Vgl. BONDARI/GRIFFITHS (2011), Seite 10.

⁹Vgl. BONDARI/GRIFFITHS (2011), Seite 11.

¹⁰Vgl. HETZEL (2012), Seite 48.

2.3.3 Editor

Als Editor zum programmieren der einzelnen Funktionen ist kein spezieller erforderlich. Allerdings sollte darauf geachtet werden, dass dieser entsprechende Anforderungen entspricht¹¹:

1. Syntax-Highlighting
2. Anzeige von Zeilennummern
3. Anzeige von zusammengehörigen Klammern

Bei der Mentoren-Suche wurde für Windows das Programm Notepad++ verwendet, welches sich unter <http://notepad-plus-plus.org/download> herunterladen lässt und frei verfügbar ist. Für Mac-Systeme kann das Programm *textwrangler* verwendet werden.. Dies ist ein proprietäres Programm, welches kostenlos unter folgenden Link <http://www.barebones.com/products/TextWrangler/> heruntergeladen werden kann.

2.3.4 FTP Client

Um vom lokalen Rechner Dateien auf den Server zu transportieren, wird ein FTP-Client benötigt. Dafür kann unter Windows das Programm Filezilla verwendet werden. Es handelt sich hierbei um ein unter der GPL lizenzierten Programm.

Für Mac OS X kann das Programm Cyberduck verwendet werden. Auch hierbei handelt es sich um GPL-lizenziertes Programm.¹²

2.4 Programmierstandards

Standards für Programmierer sind deshalb sehr wichtig, weil damit auch andere Programmierer den logischen Aufbau des Plugins schnell verstehen können - auch wenn diese manchmal nicht mit den eigenen Ideen zufrieden sind. Die Hauptpunkt, um den eigenen Quellcode möglichst gut zu organisieren und strukturieren, werden an dieser Stelle nach Bondari und Griffiths¹³ aufgezählt:

1. Aufgaben werden in Prozeduren eingeteilt

- Für jede Aufgabe gibt es genau eine Prozedur, welche aber auch andere Prozeduren aufrufen kann. Eine Regel hierbei ist, dass nicht mehr als drei Variablen der Prozedur übergeben werden sollen.

2. Es werden Klassen verwendet

- Je mehr programmiert wird, desto mehr wird in die objektorientierte Programmierung eingestiegen, da mit diesem Programmierparadigma Vererbung, Erweiterungen und Klassen besser programmiert werden können. Eine prozedurale Programmierung mit Klassen ist auch möglich. Wir haben uns aufgrund von nur drei Entwicklern und diesem vergleichbar kleinen Projekt für die prozedurale Programmierung entschieden.

¹¹Vgl. BONDARI/GRIFFITHS (2011), Seite 12-13.

¹²Vgl. BONDARI/GRIFFITHS (2011), Seite 14.

¹³Vgl. BONDARI/GRIFFITHS (2011), Seite 15 - 18.

3. Verwendung von beschreibenden Variablen und Funktionen

- Für PHP gibt es keine großen Anforderungen an eine Variable, was den Namen angeht. Allerdings sollte immer darauf geachtet werden, dass Variablen den entsprechenden Kontext, möglichst kurz und auf Englisch wiedergeben sollten. Wenn der Quelltext mit Variablen bestückt ist, die sich nicht von selbst erklären, fällt es sehr schwer die Logik des Programms zu verstehen.

4. Verwendung von modularer Programmierung

- Indem modulare Programmierung verwendet wird, können einzelne Klassen beispielsweise für andere Projekte verwendet werden, da diese übersichtlich abgespeichert werden.

In dem Projekt der Mentoren-Suche wurde versucht, sich an diese Standards zu halten. Deshalb soll in diesem Tutorial nicht davon abgewichen werden.

3 Aufbau des Plugins

In diesem Kapitel wird der grundsätzliche Aufbau eines Plugins besprochen. Dabei werden neben allgemeinen Details, wie die Aktivierung eines Plugins, auch einzelne Programmierdetails angesprochen. Inbegriffen sind hierbei der erste Abschnitt eines jeden Plugins: der Information Header, sowie die einzelnen Funktionen und wie diese referenziert werden.

3.1 Allgemeines

Ein paar Details zur Programmierung von Plugins wurden bereits im Unterkapitel 2.2 angesprochen - beispielsweise eine Einleitung zu hooks.

Nach Hetzel¹⁴ gibt es jedoch einige technische Details, mit welchen sich der Programmierer beschäftigen muss, bevor die eigentliche Programmierung beginnen kann. Zunächst ein Detail, welches auch in Was ist ein Plugin? angesprochen wurde, jedoch von außerordentlicher Wichtigkeit ist: Plugins werden generell unter Wordpress in dem Plugin-Ordner `/wp-content/plugins/`. Dies hat den wichtigen Grund, dass Wordpress nur Plugins erkennt, welche auch in diesem Ordner kopiert worden sind. Weiterhin bietet es sich an, für jedes Plugin einen eigenen Unterordner zu erstellen, um so die Übersichtlichkeit bei dem Einsatz mehrerer Plugins zu gewährleisten. Auch sollte die Hauptdatei des Plugins immer den Namen des Plugins haben und eine Namenskonvention, welche nicht mit anderen Plugins in Konflikt kommt, eingehalten werden.

An dieser Stelle möchten wir vorstellen, wie die Details für das Plugin Mentoren-Suche aussehen:

1. Als Hauptordner unter dem Pluginverzeichnis `/wp-content/plugins` haben wir *mentoren-suche* gewählt
2. Die Hauptdatei unseres Plugins lautet *mentoren-suche.php*

Soviel zu den Details für die Vorbereitung der Programmierung. Im nächsten Abschnitt wird erläutert, wie das Plugin aktiviert wird.

3.2 Aktivierung des Plugins

Insgesamt gibt es aus Anwendersicht drei Schritte¹⁵, um ein Plugin zu aktivieren:

1. Als erster Schritt wird zum Wordpress Dashboard navigiert (`http://localhost/wp-admin/` oder `http://deine-domain/wp-admin/`)
2. Anschließend wird die Option *Plugins* ausgewählt
3. Als letzter Schritt wird auf *Aktivieren* des entsprechenden Plugins (*mentoren-suche*) geklickt

Soviel einmal zur Theorie. Der nächste Abschnitt leitet in die Praxis der Aktivierung. Zur besseren Übersichtlichkeit und für einen ersten Eindruck des fertigen Plugins, ist in Abbildung 1 das Pluginmenü mit den entsprechenden Informationen aus dem im nächsten Unterkapitel Information Header abgebildet.

Nach einem Klick auf *Aktivieren* ist das Plugin aktiviert und es erscheint die in Abbildung 2

¹⁴Vgl. HETZEL (2012), Seite 274.

¹⁵Vgl. BONDARI/GRIFFITHS (2011), Seite 29.

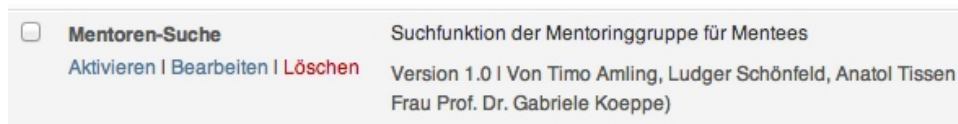


Abbildung 1: Pluginübersicht im Pluginmenü



Abbildung 2: Erfolgreiche Meldung der Aktivierung

abgebildete Meldung. Während der Aktivierung werden verschiedene Funktionen ausgeführt. Beispielsweise wird im Backend-Menü in der Menüstruktur angelegt oder es wird geprüft, ob die benutzte Wordpressversion mit dem Plugin kompatibel ist. Weiterhin steht auch nun der Shortcode für das Plugin bereit und die Übersetzungen der einzelnen Textelemente werden angezeigt. Wie dies alles zu programmieren ist, wird in den nächsten Kapiteln besprochen. Dabei wird als erstes auf den Informationen Header, anschließend auf einzelne Funktionen und abschließend die Hooks besprochen. Soviel erst einmal zur Aktivierung und kleinen technischen Hintergrundabläufen.

3.3 Information Header

In Wordpress ist der *Information Header* in der Pluginentwicklung immer der erste Teil des Programmes. Er wird dafür benötigt, um im Pluginmenü erforderliche Informationen über den Autoren oder eine Beschreibung des Plugins darzustellen - wie bereits im Unterkapitel Aktivierung des Plugins dargestellt ist. Ohne diesen Header wird kein Plugin funktionieren, auch wenn die Programmierung bis in kleinste Detail abgeschlossen ist.¹⁶

Wie ein solcher aussehen könnte, ist beispielhaft in Listing 1 dargestellt. Dabei handelt es sich immer um eine Angabe, welche in Kommentaren geschrieben wird (`/*`).

```
1 <?php
2 /*
3 Plugin Name: Mentoren-Suche
4 Description: Suchfunktion der Mentoringgruppe für Mentees
5 Version: 1.0
6 Author: Timo Amling, Ludger Schönfeld, Anatol Tissen im Auftrag von
    Mentoring4Excellence (FH Köln, Campus Gummersbach –
    Programmleitung: Frau Prof. Dr. Gabriele Koeppe)
7 – im August 2012
8 License: GPL 3
9 */
```

¹⁶Vgl. BONDARI/GRIFFITHS (2011), Seite 30.


```
10 ...  
11 ?>
```

Listing 1: Information Header

Wie sich erkennen lässt, umfasst der Header insgesamt fünf Informationsobjekte. Zu den hier vorgestellten kann optional auch noch die Plugin-URL - also falls das Plugin auf einer Webseite im Internet angeboten wird - angegeben werden.¹⁷ Da es sich bei den Lesern dieses Tutorials größtenteils um Personen handelt, welche mit Informatik beruflich oder privat zu tun haben, wird hier nicht näher auf die einzelnen Objekte eingegangen.

Im nächsten Kapitel werden erste Funktionen erzeugt, diese eingebunden und auf Fehler aufmerksam gemacht.

3.4 Benutzerdefinierte Funktionen

In diesem Kapitel geht es darum, die ersten benutzerdefinierten Funktionen zu schreiben und auf mögliche Fehlerursachen aufmerksam zu machen. Dabei wird davon ausgegangen, dass eine korrekte php-Konfiguration vorhanden und der entsprechende Debug-Modus aktiviert ist. Wir möchten an dieser Stelle nur auf das Wordpress-System eingehen und nicht auf externe Konfigurationsmöglichkeiten, da dies den Umfang dieses Tutorials sprengen würde. Weitere Informationen sind jedoch unter <http://php.net/manual/en/debugger.php> zu finden.

3.4.1 Das erste eigene Plugin

Als unsere erste Funktion soll eine normale Textausgabe mit folgendem Inhalt geschrieben werden *Ich bin eine Textausgabe*. Dafür eignet sich der Befehl *print*. Dieser ist bereits in Listing 4 eingefügt worden und kann anschließend im Pluginmenü von Wordpress aktiviert werden. Benutzerdefinierte Funktionen werden in Wordpress dafür benötigt, um Logik des Plugins zu programmieren (näheres dazu findet sich im Kapitel 3.2).¹⁸

```
1 <?php  
2 /*  
3 Plugin Name: Mentoren-Suche – Printausgabe  
4 Description: Ein erster Versuch  
5 Version: 0.1  
6 //...WEITERE ANGABEN VON INFORMATION-HEADER...  
7 */  
8 print "Ich bin eine Textausgabe";  
9 ?>
```

Listing 2: Erste Printausgabe

Nach der Aktivierung dieses Plugins erscheint im oberen Bereich der Plugin- und jeder anderen Seite der Back- und Frontendseite der entsprechende Text (vgl. Abbildung 3).

¹⁷Vgl. BONDARI/GRIFFITHS (2011), Seite 31.

¹⁸Vgl. BONDARI/GRIFFITHS (2011), Seite 35-36.

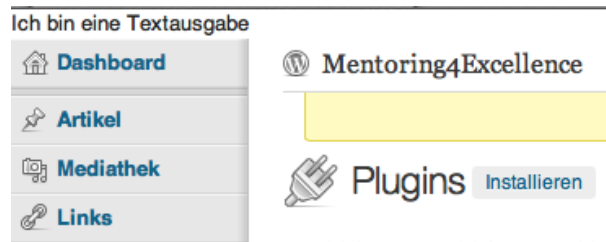


Abbildung 3: Printbefehl-Ausgabe

Dieses Verhalten ist darauf zurückzuführen, dass das Plugin nach dem aktivieren direkt geladen wird und so auf jede Seite vorhanden ist.

Dies lässt sich durch das Hinzufügen von Funktionen unterbinden, welches in Listing 3, Zeile 8 - 10 dargestellt ist.

```
1 <?php
2 /*
3 Plugin Name: Mentoren-Suche – Printausgabe 2
4 Description: Ein zweiter Versuch
5 Version: 0.2
6 //...WEITERE ANGABEN VON INFORMATION-HEADER...
7 */
8 function printausgabe() {
9     print "Ich bin eine Textausgabe";
10 }
11 ?>
```

Listing 3: Verbesserte Printausgabe

Das nun verwendete Plugin bewirkt, dass das Printstatement so isoliert wird, dass es erst bei unmittelbaren Aufruf verwendet wird.

Zwar ist nach dem Aktivieren des Plugins das Textfeld verschwunden, allerdings taucht es dieses auch nicht an anderer Stelle auf. Dafür werden sogenannte Hooks (siehe Abschnitt 3.5) verwendet.¹⁹

Bevor diese allerdings angesprochen werden und nach diesem kleinen Exkurs auch mit dem Mentoring-Plugin weitergearbeitet wird, wird im nächsten Kapitel 3.4.2 noch die Aktivierung des Debug-Modus kurz erläutert.

3.4.2 Debug-Modus

Es ist ratsam bei der Pluginentwicklung im sogenannten-Debug-Modus von Wordpress zu arbeiten, um so Fehlermeldungen zu sehen und diese zu beheben. Dieser Modus lässt sich in der Hauptkonfigurationsdatei *wp-config.php* einstellen. Dafür muss im folgenden Listing der Wert in Zeile 9 von *false* (ausgeschaltet) auf *true* (eingeschaltet) geändert werden.²⁰

```
1 ...
2 /**
```

¹⁹Vgl. BONDARI/GRIFFITHS (2011), Seite 38-39.

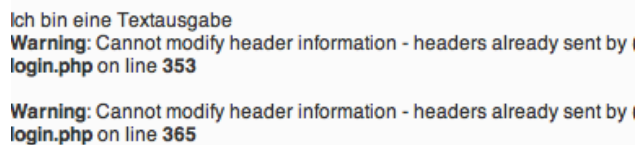
²⁰Vgl. BONDARI/GRIFFITHS (2011), Seite 23.

```
3  * For developers: WordPress debugging mode.
4  *
5  * Change this to true to enable the display of notices during
    development.
6  * It is strongly recommended that plugin and theme developers use
    WP_DEBUG
7  * in their development environments.
8  */
9  define( 'WP_DEBUG', true );
10 ...
```

Listing 4: Debug-Modus aktivieren

Nachdem der Debug-Modus aktiviert wurde, kann als Kontrolle nochmal die Funktion aus Listing 4 aktiviert werden. Anschließend wird zwar wieder im oberen Bereich die entsprechende Textausgabe ausgegeben, allerdings werden darunter einige Fehlermeldungen und Warnungen angezeigt: Der Debugger ist also aktiviert (siehe Abbildung 4).

Wie dieser Fehler behoben wird ist an dieser Stelle schon aus Listing 3 bekannt: Nämlich die Printausgabe als Funktion schreiben.



```
Ich bin eine Textausgabe
Warning: Cannot modify header information - headers already sent by
login.php on line 353

Warning: Cannot modify header information - headers already sent by
login.php on line 365
```

Abbildung 4: Printbefehl mit aktivierten Debugger

Nachdem die Vorbereitungen abgeschlossen und die ersten Funktionen geschrieben wurde, wird nun das Referenzieren von Hooks über Filter und Actions angesprochen.

3.5 Hooks

In diesem Kapitel wird sich mit sogenannten *Hooks* beschäftigt. Diese sind dafür verantwortlich, wann ein Plugin ausgeführt ist. Genauer gesagt handelt es sich bei einem Hook um eine Art Eventhandler: Wenn ein bestimmtes Ereignis auftritt, wird eine gewisse Funktion ausgeführt. Ein solches Ereignis könnte beispielsweise vorliegen, wenn ein bestimmtes Menü angeklickt wird oder ein Suchfeld aktiviert wird.

Dabei sollte noch erwähnt werden, dass sich diese Eventhandler in zwei Komponenten unterteilen lässt: Action- und Filter-Hooks.²¹

Wie diese genau funktionieren und allgemein aufgebaut sind, wird in den nächsten Abschnitten erläutert.

²¹Vgl. BONDARI/GRIFFITHS (2011), Seite 39.

3.5.1 Referenzieren mittels `add_action()` und `add_filter()`

Bevor die eigentlichen Anwendungsbeispiele kommen, soll an dieser Stelle zuvor die allgemeine Syntax von *action*- und *filter*-Hooks erläutert werden: Anhand von Tabelle 1 lässt sich erkennen,

Hook-Art	Syntax
Action	<code>add_action('Event','Funktion')</code>
Filter	<code>add_filter('Event','Funktion')</code>

Tabelle 1: Vergleich Action- und Filter-Hooks

dass die beiden Hook-Arten genau gleich aufgebaut sind. Wie immer steckt hier der Teufel im Detail - genauer gesagt: Wie Wordpress mit diese beide Arten umgeht.

Wenn eine Funktion zu einem Action-Hook verbunden wird, ignoriert die Funktion Eingabeparameter wie beispielsweise übergebene Variablen. Auch wenn die Funktion einen Rückgabewert besitzt, wird auch dieser Wert ignoriert.²²

Dies ist der Punkt, an dem der Unterschied zum Filter-Hook klar wird: Dieser akzeptiert Eingabeparameter und Werte und kann diese verändert - oder eben gefiltert - zurückgeben.

Dies geschieht noch bevor dieser Parameter dem Benutzer auf dem Bildschirm angezeigt oder in die Wordpress-Datenbank geschrieben wird²³.

Im Grunde genommen haben die beiden Funktionen eines gemeinsam: Beide binden eine Event an eine Funktion, gehen nur jeweils anders damit um.²⁴

Eine überschaubare Funktionsreferenz von Action und Filter-Hooks lässt sich unter http://codex.wordpress.org/Plugin_API finden.

3.5.2 Hook-Anwendungsbeispiele

An dieser Stelle werden nun ein paar Beispiele aus dem Mentoren-Plugin genommen und weitestgehend erläutert. Anschließend wird noch eine weitere Funktion vorgestellt: Der `add_option`-Hook.

3.5.2.1 Vordefinierte Actions

Wordpress bietet von Hause aus schon vordefinierte Actions, welche ab Wordpressversion 2.1 vorhanden und verwendbar sind. Da die Anzahl dieser vordefinierten Actions ziemlich umfangreich ist, wird an dieser Stelle nur auf ein Action verwiesen, welches vergleichbar oft in dem Mentorenplugin vorkam: Die Action `add_action`.²⁵

```
1 <?php
2 ...
3 add_action( 'admin_menu', 'register_Mentees_menu' );
4 ...
5 <?>
```

Listing 5: Beispiele für `add_action()`

²²Vgl. BONDARI/GRIFFITHS (2011), Seite 40.

²³Vgl. MULLENWEG (abgerufen am 22.10.12).

²⁴Vgl. BONDARI/GRIFFITHS (2011), Seite 40.

²⁵Vgl. MULLENWEG (abgerufen am 22.10.12).

An dieser Stelle wird auf die in Listing 6 vorgestellten Funktionen genauer eingegangen.

Allgemein lässt sich formulieren, dass die Action *admin_menu* dafür verwendet wird, um ein zusätzliches Untermenü und bestimmte Menü-Optionen im Administrator-Bereich anzulegen.²⁶ Dies bedeutet, dass beim Aktivieren des Plugins verschiedene Actions ausgeführt werden. Beispielsweise wird eine Funktion *register_Mentees_menu* aufgerufen, welche für die Erstellung des Submenüs zur Verwaltung des Plugins geschrieben wurde.

Weitere Informationen werden zu diesem Thema im Kapitel 7 angesprochen.

3.5.2.2 Eigene Actions

Natürlich ist es auch möglich eigene Actions zu schreiben und diese einer Menüstruktur zuzuordnen. Im Listing 6 sind solche Beispiele aufgelistet.

```
1 <?php
2 ...
3 //wenn check erfolgreich , werden DB-Tabellen angelegt
4 add_action( 'hk_mentees_db_create' , 'mentees_db_create' );
5 add_action( 'hk_mentor_db_create' , 'mentor_db_create' );
6 ...
7 <?
```

Listing 6: Beispiele für eigene `add_action()`

Falls also ein Mentor oder Mentee angelegt werden soll, werden die entsprechenden Daten wie Vor- oder Nachname in ein Formular eingetragen. Anschließend wird mit einem Button die Eingabe bestätigt. Genau das ist der Zeitpunkt, an dem die Action eintritt: Es muss eine Datenbank angelegt werden. Dies geschieht über die beiden Funktionen *mentor_db_create* und *mentees_db_create*.

Wie mit solchen Datenbankobjekten interagiert wird, wird genau in Kapitel 6 besprochen.

3.5.2.3 Vordefinierte Filter

Da in dem aktuellen Plugin der Mentoren-Suche keine Filter zum Einsatz kommen, aber trotzdem für ein solches Tutorial von Bedeutung sind, wird an dieser Stelle auf Fachliteratur zurückgegriffen und entsprechend erläutert.

Wie schon erläutert, handelt es sich bei Filter-Hooks um Funktionen, damit Wordpress bestimmte Text oder weitere Typen modifizieren kann.

Nach Bondari und Griffiths²⁷ soll das Ziel dieses kleinen Beispiel-Plugins nur verdeutlichen, wie Actions im Zusammenhang mit Filtern funktionieren. Dafür wird eine Kopie des Standard-Plugins *hello dolly von Wordpress* verwendet und verändert.

Die dazugehörige Plugindatei liegt im Pluginverzeichnis von Wordpress und lautet *hello.php*

```
1 <?php
2 ...
3 //VORHER
4 add_filter( 'admin_footer' , 'hello_dolly' );
5
```

²⁶Vgl. MULLENWEG (abgerufen am 23.10.12b).

²⁷Vgl. BONDARI/GRIFFITHS (2011), Seite 42 - 43.

```
6 //NACHER
7 add_filter( 'the_content', 'hello_dolly' );
8 ...
9 <?
```

Listing 7: Beispiele für add_filter()

Wie in Listing 7 dargestellt, wird als Vorbereitung das Event von *admin_footer* auf *the_content* geändert, da hier die bisherigen Posts verändert werden sollen.

Dafür wird in der Datei *hello.php* zusätzlich auch die Funktion *hello_dolly()* verändert.

Dies ist in Listing 8 genauer dargestellt.

```
1 <?php
2 ...
3 //VORHER
4 function hello_dolly() {
5     $chosen = hello_dolly_get_lyric();
6     echo "<p_id='dolly'>$chosen</p>";
7 }
8
9 //NACHER
10 function hello_dolly($input) {
11     $chosen = hello_dolly_get_lyric();
12     return $input . "<p_id='dolly'>$chosen</p>";
13 }
14 ...
15 <?
```

Listing 8: Funktion *hello_dolly* mit Input-Variable

Was hier nun passiert, lässt sich wie folgt beschreiben: Die Variable *\$input* beschreibt den Inhalt eines jeden Posts. Dieser wird anschließend in die Filter-Funktion übertragen, modifiziert und anschließend per *return* zurückgegeben.

Anstatt die kompletten Inhalt eines Posts zu überschreiben, wird stattdessen ein beliebiger Text unterhalb des Posts eingebunden. Damit wird also schlussendlich der Inhalt verändert und dargestellt - genau das, was eine Filter-Funktion machen sollte.

Unterm Strich lässt sich formulieren, dass eine Funktion, die bei einer Action aktiviert wird, keine Eingabe- und Ausgabewerte akzeptiert.

Eine Action, die über eine Filter-Funktion aufgerufen wird, erlaubt die oben genannten Einschränkungen. Dies ist zugleich der Unterschied zwischen diesen beiden Event-Typen.²⁸ Weitere Informationen finden sich unter den folgenden Internetseiten zu finden:

1. http://codex.wordpress.org/Function_Reference
2. http://codex.wordpress.org/Action_Reference

3.5.2.4 Funktion add_option

Zum Schluss dieses Kapitels wird sich noch mit dem *add_option*-Hook beschäftigt. Dieser ist

²⁸Vgl. BONDARI/GRIFFITHS (2011), Seite 43.

mehrmals im Plugin zum Einsatz gekommen und ist syntaktisch auch anders aufgebaut als ein *filter* oder *action*-Hook.

Laut Mullenweg²⁹ wird eine `add_option`-Funktion dafür verwendet, um Optionen in die Wordpress-Datenbank zu schreiben. Als erstes wird dabei geprüft, ob eine Funktion mit demselben Namen schon vorhanden ist. In diesem Fall wird ein *false* zurückgegeben.

Wenn anschließend der Name keiner der geschützten Optionen überschreiben soll, wird die Option erstellt. Dies ist der allgemeine Ablauf, der sich hinter *add_option* befindet. Die allgemeine Syntax sowie ein Anwendungsbeispiel lässt sich aus Listing 9 ablesen.

```
1 <?php
2 ...
3 // Allgemeine Syntax
4 add_option( 'Option', 'Value', 'Deprecated', 'autoload' )
5 // Beispiel
6 add_option( 'ms_frontend_searchform_url', '', '', yes );
7
8 <?
```

Listing 9: Beispiele für `add_option()`

In dem in Listing 9 dargestellten Beispielen muss ergänzt werden, dass die Werte *Value* (für Option-Name reserviert), *deprecated* (nur für Wordpressversion 2.3) und *autoload* (ob die Funktion automatisch geladen werden soll mit Standardwert *yes* für ja) optional sind.

Im konkreten Anwendungsbeispiel wird also die Option *ms_frontend_searchform_url* aufgerufen und in die Datenbank eingetragen. Diese Option ist dafür verantwortlich, dass im Frontendbereich an der Stelle, an dem der Shortcode eingebunden wird (ausführlich im nächsten Kapitel), eine Suchmaske erscheint und ausgeführt werden kann.

Wie diese aussieht, wird im nächsten Kapitel näher besprochen.

Soviel zu den Hooks. Im nächsten Kapitel 5 wird die Thematik Shortcodes behandelt.

²⁹Vgl. MULLENWEG (abgerufen am 23.10.12a), Artikel Function Reference/add option.

4 Menüerstellung

Nachdem im vorigen Kapitel der allgemeine Aufbau eines Plugins und der Begriff des *Hooks* erklärt und mit Beispielen aus der Mentoren-Suche erläutert wurde, geht es in diesem Kapitel um das Erstellen einer Menüstruktur im Backend von Wordpress.

Dabei wird zuerst auf den Begriff eines Menüs eingegangen (vgl. Abschnitt 4.1), danach wird erläutert wie Menüs erstellt werden (vgl. Abschnitt 4.2 und 4.3). Diese beiden Abschnitte werden dann mit Beispielen aus der Mentoren-Suche jeweils abgerundet (vgl. Abschnitt 4.2.1) und Abschnitt 4.3.1). Zum Schluss kommt dann noch ein Teil, der sich mit der Integration von Menüs in bestehende vom Wordpress-Backend beschäftigt (siehe Abschnitt 4.4).

4.1 Wozu dient ein Menü?

Ein Menü dient zur Verwaltung des Plugins und ist im Backend zu finden. Dabei können beispielsweise verschiedene Optionen eingestellt werden können. Dabei kann Wordpress mit zwei verschiedenen Arten von Menüs umgehen. Dieses ist zum einen das Top-Level-Menü (siehe Abschnitt 4.2) und das Submenü (vgl. Abschnitt 4.3).³⁰

4.2 Erstellen von Top-Level-Menüs

Ein Top-Level-Menü - oder auch Obermenü - ist laut Williams/Richard/Tadlock³¹ die oberste Einheit eines Menüs. Beispielsweise wäre das Menü *Einstellungen* im Wordpress-Admin-Bereich ein Top-Level-Menü. Die Autoren raten, für jedes Plugin ein solches Menü zu erstellen, welche mehrere Einstellungsmöglichkeiten bietet. Für die Erstellung eines Top-Level-Menüs wird die Funktion `add_menu_page` verwendet³²:

```
1 <?php
2 ...
3     add_menu_page( $page_title, $menu_title, $capability, $menu_slug,
4                   $function, $icon_url, $position );
5 ?>
```

Listing 10: Syntax der Top-Level-Menü - Funktion

Dabei sind die Parameter wie folgt zu verstehen:

1. **\$page_title**

- Dies ist der Titel der Seite und verpflichtend (vom Typ String)

2. **\$menu_title**

- Hierbei handelt es sich um den Menünamen (als String), welchem im Dashboard angezeigt werden soll. Auch dieser ist verpflichtend.

3. **\$capability**

³⁰Vgl. WILLIAMS/RICHARD/TADLOCK (2011), Seite 59.

³¹Vgl. WILLIAMS/RICHARD/TADLOCK (2011), Seite 60 - 61.

³²Vgl. MULLENWEG (abgerufen am 16.02.13a).

- Dies ist die Beschreibung für die Fähigkeit an Rechten, um auf das Menü zugreifen zu können. Auch hierbei handelt es sich um einen String. Rechte sind hierbei als Rollen zu betrachten, mit denen ein Benutzer etwas im Blog ändern darf oder nicht. Dies ist von dem jeweiligen Recht abhängig. Weitere Informationen finden sich unter <http://faq.wpde.org/welche-benutzerrolle-bietet-welche-rechte>.

4. `$menu_slug`

- Dies ist ein Bedienungsparameter, um die Einstellungsseite auf der Menüseite anzuzeigen. Es könnte also auch eine PHP-Datei sein. Dieser Parameter ist erforderlich.

5. `$function`

- Ist der erste optionale Parameter, welcher eine Funktion angeben kann, die auf der aufzurufenden Seite ausgeführt werden soll.

6. `$icon_url`

- Die Icon-URL ist auch ein optionaler Parameter, mit dem der Programmierer die Möglichkeit hat, ein Menüicon einzubinden. Beispielsweise könnte dies bei einem professionellen Plugin durchaus Sinn machen (beispielsweise das Firmenlogo).

7. `$position`

- Dieser optionale Parameter, der als Integer-Wert angegeben werden muss, beschreibt die Position, an der das Menü erscheinen soll. Je nach Zahl, wird das Menü dann an entsprechender Stelle eingeblendet. Wenn keine Zahl angegeben wurde, wird das Menü automatisch am Ende des Menüs angezeigt.
- Beispiele für Menüplatzierungen
 - a) 2 - Dashboard
 - b) 20 - Seiten
 - c) 65 - Plugins
 - d) 70 - Benutzer

Wie in der oben genannten Aufzählung beschrieben, gibt es verschiedene optionale und verpflichtende Parameter. Um diese in der Praxis zu sehen, wird nun in Unterabschnitt 4.2.1 ein Beispiel zu dieser Funktion vorgestellt und erläutert.

4.2.1 Beispiel Top-Level-Menüs aus Mentoren-Suche

Nachdem die Theorie über das Erstellen von Top-Level-Menüs im Abschnitt 4.2 besprochen wurde, wird nun anhand von einem Beispiel dieses Wissen praxisnah aufgezeigt. Dabei handelt es sich um das Obermenü des Plugins Mentoren-Suche (siehe Listing 11).

```
1 <?php
2 ...
3 function register_Mentees_menu() {
```

```

4   add_menu_page(__( 'Mentoren_Suche', 'mentoren-suche' ), __( 'Mentoren_Suche
      ', 'mentoren-suche' ), 'unfiltered_html', 'mentees', 'ms_row_edit' );
5 }
6 ...
7 ?>

```

Listing 11: Beispiel Mentoren-Suche Top-Level-Menü

Die Funktion `register_Mentees_menu()` hat die Aufgabe, den Menüpunkt *Mentoren Suche* als Top-Level-Menü in das Backend von Wordpress einzubinden. Dazu wird die Rechteklasse *mentees* verwendet, welche eigens in Blog der Mentorinnen und Mentoren erstellt wurde - also gewissermaßen eine selbst definierte Benutzerrechtsklasse. Anschließend wird die Funktion `ms_row_edit` aufgerufen, mittels welcher die Datensätze von Mentorinnen und Mentoren bearbeitet und gelöscht werden kann. Wie dies schlussendlich aussieht, ist in Abbildung 5 abgebildet.



Abbildung 5: Ausgabe Top-Level-Menü-Funktion

4.3 Erstellen von Submenüs

Da bereits im Abschnitt 4.2 beschrieben wurde, wie Top-Level-Menüs zu erstellen sind, wird nun auf die Submenüs - oder auch Untermenüs - eingegangen.

Beispielsweise, wäre laut Williams/Richard/Tadlock³³ das Menü *Allgemein* im Backend von Wordpress das Submenü, während *Einstellungen* das Top-Level-Menü.

Um ein solches Untermenü zu erstellen, wird die Funktion `add_submenu_page` verwendet³⁴ :

```

1  <?php
2  ...
3  add_submenu_page( $parent_slug, $page_title, $menu_title, $capability,
      $menu_slug, $function );
4  ...

```

³³Vgl. WILLIAMS/RICHARD/TADLOCK (2011), 61.

³⁴Vgl. MULLENWEG (abgerufen am 16.02.13b).

5 ?>

Listing 12: Syntax der Submenü - Funktion

Dabei sind die Parameter wie folgt zu verstehen:

1. **\$parent_slug**

- Hierbei handelt es sich um einen nicht-optionalen Parameter, welcher den Namen des Obermenüs beschreibt. Falls ein Menü geschrieben werden soll, welches keinen Menü als Unterpunkt auftaucht, kann der Wert *NULL* oder *options.php* gesetzt werden.

2. **\$page_title**

- Der Seitentitel beschreibt den Text, der auf der Seite als Titel nach dem anklicken des Submenüs angezeigt wird. Dieser ist in Form eines Strings zu verwenden und nicht optional.

3. **\$menu_title**

- Der Menütitel beschreibt den Titel, welcher in dem Untermenü-Button angezeigt wird.

4. **\$capability**

- Die \$capability beschreibt die Rechtezuteilung, damit User diese sehen oder nicht sehen können: Je nachdem, wie die Rechtevergabe aussieht. Diese Angabe ist nicht optional. Dabei kann zwischen verschiedenen Hierarchien von Rechten unterschieden werden. Diese fangen an oberster Stelle mit dem Super-Administrator und Administrator an, und hören bei dem Abbonennten auf. Weitere Informationen finden sich unter http://codex.wordpress.org/Roles_and_Capabilities.

5. **\$menu_slug**

- Dies ist Parameter, um das Menü eindeutig anzusprechen. Dies ist erforderlich, um das Elternmenü nicht mehrmals zu erstellen. Diese Angabe ist erforderlich.

6. **\$function**

- Hierbei handelt es sich um einen String in Form einer Funktion, welcher als Ausgabe für die angeklickte Seite dient. Dieser ist optional.

Die Theorie wird nun anhand eines Beispiels im Abschnitt 4.3.1 erläutert.

4.3.1 Beispiel Untermenü aus Mentoren-Suche

Die bereits genannten Parameter sollen nun anhand von einem Beispiel aus der Mentoren-Suche veranschaulicht werden.

```
1 <?php
2 ...
3 function me_dataset_import() {
```

```
4     add_submenu_page( 'mentees', __('Datensatz□eintragen', 'mentoren-suche')
      , __('Datensatz□eintragen', 'mentoren-suche'), 'unfiltered_html', '
        row-import', 'ms_row_new');
5 }
6 ...
7 ?>
```

Listing 13: Beispiel Mentoren-Suche Submenü

Die Funktion *me_dataset_import* ist dafür da, den Untermenüpunkt *Datensatz eintragen* in das Menü *Mentoren Suche* einzubinden. Anschließend wird dann die Funktion *row_import*, welche dann die Funktion *ms_row_new* aufruft. Diese ist zum Eintragen von Datensätzen vorhanden. Abschließend ist dieses Beispielenü in der Abbildung 6 dargestellt.

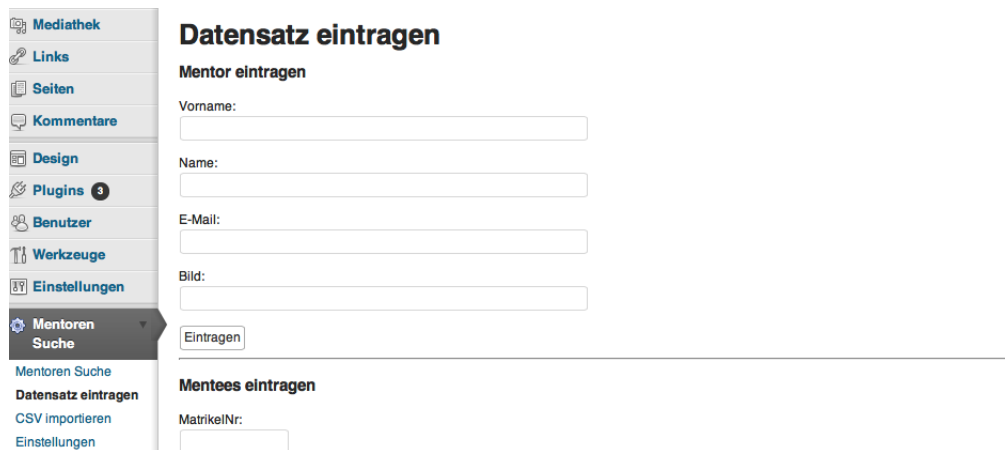


Abbildung 6: Ausgabe Submenü-Funktion

Im nächsten Kapitel geht es nicht um das neue anlegen von Top- oder Submenüs, sondern um die Integration von Menüs in bereits bestehende Standard-Menüs von Wordpress.

4.4 Integration in bestehende Menüs

Nachdem bereits erwähnt wurde, wie Top-Level- und Submenüs erstellt werden, kommt nun zum Schluss die Integration von Einstellungsseiten in eine vorhandene Menüstruktur.

Wenn ein Plugin einen überschaubaren Umfang hat, besitzt es meistens auch nicht mehr als eine Einstellungsseite. Nun könnte ein Programmierer auch für diese Seite ein einzelnes Top-Level- und/oder Submenü erstellen. Alternativ dazu gibt es laut Williams/Richard/Tadlock³⁵ die Möglichkeit, bereits in vorhandenen Menüstrukturen, Untermenüs zu erstellen. Dies könnte je nach Kontext des Plugins beispielsweise unter Design oder Einstellungen eingebaut werden. Weitere Beispiele sind der Abbildung 7 gezeigt.



Abbildung 7: Übersicht Admin-Bereich von Wordpress

³⁵Vgl. WILLIAMS/RICHARD/TADLOCK (2011), Seite 62.

Es gibt unter Wordpress sehr viele verschiedene Möglichkeiten, um die Standardmenüs mit eigenen zu erweitern. An dieser Stelle wird die Funktion `add_options_page` verwendet. Die allgemeine Syntax sieht folgendermaßen aus³⁶:

```
1 <?php
2 add_options_page( $page_title , $menu_title , $capability , $menu_slug ,
    $function );
3 ?>
```

Listing 14: Syntax zum integrieren von Menüs in bereits vorhandene

Im weiteren werden nun die einzelnen Parameter laut Wordpress Codex erläutert:

1. **\$page_title**

- Hierbei handelt es sich um einen Parameter vom Typ String und beschreibt den Titel der Seite, die beim auswählen des Menüs geöffnet wird. Diese Angabe nicht erforderlich.

2. **\$menu_title**

- Auch dieser Parameter ist nicht optional und beschreibt den dargestellten Text für das Menü - also die Menübezeichnung.

3. **\$capability**

- Der zweite Parameter beschreibt die Mindestfähigkeit an Rechten, um das Untermenü anklicken zu können. Auch hierbei handelt es sich um eine Rechte-Hierarchie, welche genauer unter der folgenden Adresse tiefer beschrieben wird: http://codex.wordpress.org/Roles_and_Capabilities.

4. **\$menu_slug**

- Hierbei handelt es sich um einen eindeutigen Indentifizierer, um das Untermenü eindeutig zu beschreiben. Auch dieser Parameter ist nicht optional.

5. **\$function**

- Hingegen den anderen Parametern, ist dieser Parameter optional und beschreibt eine Funktion, die aufgerufen wird, um den Inhalt der aufgerufenen Seite darzustellen.

Soweit an dieser Stelle zur Theorie. Im nächsten Unterabschnitt wird dann ein Beispiel gezeigt und erläutert.

4.4.1 Beispiel Integration von Submenüs

In diesem Beispiel, welches in Listing 15 dargestellt ist, wird unter den Einstellungen im Dashboard von Wordpress ein weiteres Untermenü eingefügt.³⁷

³⁶Vgl. MULLENWEG (abgerufen am 17.02.13).

³⁷Vgl. WILLIAMS/RICHARD/TADLOCK (2011), Seite 62 - 63.

```
1 <?php
2 ...
3 add_action( 'admin_menu', 'boj_menuexample_create_menu' );
4
5 function boj_menuexample_create_menu() {
6     //create a submenu under Settings
7     add_options_page( 'My Plugin Settings Page', 'Menu Example Settings',
8         'manage_options', __FILE__, 'boj_menuexample_settings_page' );
9 }
10 <?>
```

Listing 15: Beispiel hinzufügen eines Untermenüs zu vorhandenen Menüs

In diesem Beispiel wird ein Untermenü mit dem Namen *My Plugin Settings Page* im Einstellungsmenü dargestellt (siehe Zeile 7). Anschließend wird der Seitentitel auch auf *My Plugin Settings Page* gesetzt und die Mindestfähigkeit an Rechten auf `manage_options` gesetzt (was soviel bedeutet, dass der Benutzer die Rechte haben muss, um die Einstellungen zu verändern / auf dieses Menü zugreifen zu können). Abschließend wird dann eine Funktion genannt, welche bei Aufruf des Menüs ausgeführt werden muss.

Schlussendlich sieht dies dann folgendermaßen aus (siehe Abbildung 8):

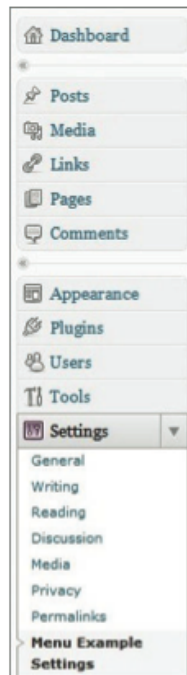


Abbildung 8: Eigenes Untermenü in vorhandene Menüstrukturen eingebunden

Weitere Möglichkeiten, um Submenüs in Wordpress einzubinden sind beispielsweise³⁸

1. `add_dashboard_page` - Fügt ein Untermenü zum Dashboard hinzu
2. `add_theme_page` - Fügt ein Untermenü zum Thememenü hinzu

³⁸Vgl. WILLIAMS/RICHARD/TADLOCK (2011), Seite 63.

3. `add_plugins_page` - Fügt ein Untermenü zum Pluginmenü hinzu

Weitere sind unter der folgenden Adresse zu finden: http://codex.wordpress.org/Administration_Menus#Admin_Menu_Functions

Die Autoren Williams/Richard/Tadlock³⁹ raten an dieser Stelle dazu, wenn ein Plugin nur eine einzelne Einstellungsseite hat, diese als Untermenü in ein vorhandenes einzubinden. Wenn allerdings mehr als eines (wie beispielsweise bei der Mentoren-Suche der Fall), sollte ein eigenes erstellt werden.

4.5 Ausblick

Insgesamt gibt es mehrere Möglichkeiten - je nach Bedarf - ein Menü neu anzulegen oder zu integrieren.

Es sollte allerdings darauf geachtet werden, dass bei einer möglichen Veröffentlichung des Plugins auch in die Dokumentation gehört, wo die Plugineinstellungen gefunden werden können. Dies ist vor allem von großen Interesse, wenn neue Menüs in bestehende integriert werden.⁴⁰ Weitere Informationen zu diesem Thema finden sich unter:

1. http://codex.wordpress.org/Function_Reference
2. http://codex.wordpress.org/Administration_Menus
3. http://codex.wordpress.org/User_Levels

³⁹Vgl. WILLIAMS/RICHARD/TADLOCK (2011), Seite 63.

⁴⁰Vgl. WILLIAMS/RICHARD/TADLOCK (2011), Seite 60 - 61.

5 Shortcodes

In diesem Kapitel wird sich den Thema *Shortcode* gewidmet. Dabei wird nach einer kleinen Definition, die Benutzung und der Einsatz in einer Wordpressumgebung erläutert.

5.1 Was ist ein Shortcode?

Um Shortcodes zu verwenden, wird nun erläutert, um was es sich hierbei handelt. Nach Bondari und Griffiths⁴¹ lässt sich ein Shortcode als Makro definieren, welches in einem Artikel oder Seite hinzugefügt werden kann. Damit lassen sich also bestimmte Funktionen zu einer Seite hinzufügen.

Die offizielle technische Dokumentation von Wordpress⁴² bringt des weiteren noch hervor, dass ein Shortcode von einem Pluginentwickler dem eigentlichen Benutzer dabei hilft, auf einer bestimmten Seite - also im Frontend von Wordpress - das Plugin einzubinden.

5.2 Shortcodes verwenden

In Abbildung 9 ist der Shortcode für das Mentoren-Plugin abgebildet, welcher im Inhaltsbereich einer Seite / Artikel das Frontend der Mentorensuche einbindet - und zwar genau an der Stelle, an der er eingebunden wird.

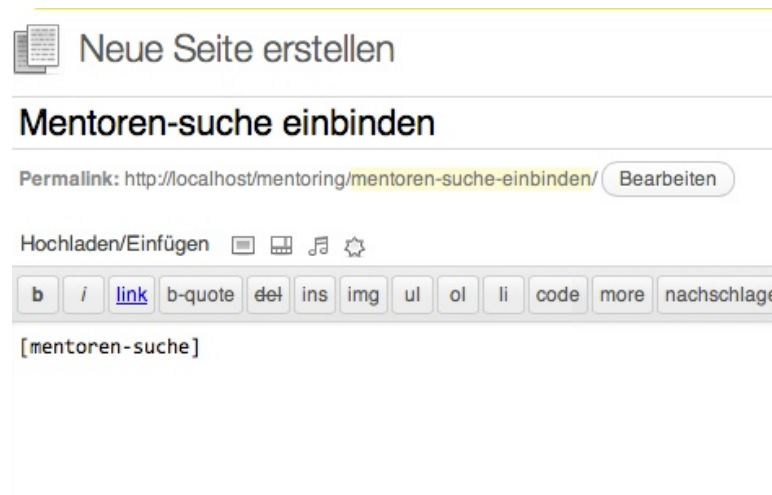


Abbildung 9: Shortcode in Seite einbinden

Im Anschluss lässt sich die neue Seite (Abbildung 10) im Browser betrachten:

⁴¹Vgl. BONDARI/GRIFFITHS (2011), Seite 177.

⁴²Vgl. MULLENWEG (abgerufen am 12.08.2012).

Mentoren-suche einbinden

**Um Ihren/Ihre Mentor/in zu finden, geben Sie bitte
und Ihren Nachnamen in die entsprechenden Felder.**

Matrikelnummer:

Ihr Nachname:

Abbildung 10: Ergebnis des eingebundenen Shortcodes

Wie ein solches Frontend erstellt wird, wird näher im Kapitel Formularerstellung (siehe Kapitel 7) erläutert. Hier geht es erstmal um die Einbindung solcher Funktionen.

Schauen wir uns also an dieser Stelle genauer an, wie ein Shortcode erstellt ist. Die allgemeine Syntax ist in Listing 16 dargestellt. Der erste Teil beschäftigt sich mit der allgemeinen Syntax, anschließend ist ein Beispiel dargestellt. Dabei sollte diese Funktion in der Hauptdatei des Plugins vorhanden sein. Also beispielsweise *mentoren-suche.php*.

```
1 <?php
2 ...
3 // Allgemeine Syntax fuer einen Shortcode
4 add_shortcode( 'Shortcodename', 'AufzurufendeFunktion' );
5 ...
6
7 // Beispieldefinition des Shortcodes
8 require_once( 'frontend.php' ); //frontend.php muss vorhanden sein
9 add_shortcode( 'mentoren-suche', 'showSearchArea' );
10 ...
11 ?>
```

Listing 16: Shortcode hinzufügen

Festzuhalten ist, dass Shortcodes sich ähnlich zu Filtern verhalten: Es werden Parameter akzeptiert und ein Ergebnis wiedergegeben. Dabei hat die *add_shortcode*-Funktion zwei Parameter: Einmal den Shortcodename, welcher dann in einem Post eingebunden werden kann (siehe *mentoren-suche* und die aufzurufende Funktion *showSearchArea*).⁴³

an dieser Stelle soll auch ein kleiner Blick auf die Funktion *showSearchArea* geworfen werden, um einen ersten Eindruck der Funktion zu bekommen. Diese ist (gekürzt) in Listing 17 dargestellt.

```
1 /**
2  * Zeigt die Suchmaske fuer das Frontend an.
3  */
4 function showSearchArea()
5 {
6     // Variablendefinitionen
7 }
```

⁴³Vgl. MULLENWEG (abgerufen am 12.08.2012).

```
8     if( isset( $send_mentorenSearch ) )
9     {
10         ...
11         // Ausgabe des Ergebnisses
12         // Fehlermeldung: Eingabedaten sollten vervollstaedigt werden
13         ...
14     }
15     else{
16         ...
17         // Eingabe der Matrikelnummer und des Nachnamens
18         ...
19     }
20 }
```

Listing 17: Gekürzte Beispielfunktion showSearchArea

Diese Funktion ist in der Datei *frontend.php* vorhanden und dient dazu, ein Formular mit einer Suchfunktion darzustellen. Dabei wird geprüft, ob ein Ergebnis vorliegt - also falls der entsprechende Datensatz gefunden wurde. Falls keiner gefunden wurde, wird eine entsprechende Fehlermeldung ausgegeben. Zusätzlich erfolgt bei einer leeren Eingabe eine Meldung, dass Matrikelnummer und Nachname eingeben werden müssen und ein Link zur Suchformular verweist. Weitere Informationen für Shortcodes finden sich unter <http://codex.wordpress.org/Shortcode>. Im nächsten Kapitel geht es um die Wordpress Shortcode-API.

5.3 Shortcode-API

Die Shortcode-API wurde mit Wordpress 2.5 eingeführt und beinhaltet Funktionen, um spezielle Makros zum erstellen von Shortcodes in Posts zu verwenden.

An dieser Stelle werden ausgewählte Funktionen vorgestellt und erläutert⁴⁴:

1. function add_shortcode(\$tag, \$func)

- Diese Funktion wurde bereits im Beispiel-Listing 16 verwendet und dient dazu, neue Shortcodefunktionen zu registrieren.
- Dabei ist *\$tag* der Shortcodename, welcher der Benutzer in einem Post verwenden kann
- Der zweite Parameter *\$func* dient dazu, eine bestimmte Funktion auszuführen, wenn der Shortcode aktiviert wird.

2. function remove_shortcode(\$tag)

- Im Allgemeinen lässt sich hiermit ein Shortcode von einer Seite entfernen und es wird nur der Inhalt angezeigt, der auch ohne Shortcode vorhanden ist.
- Der Parameter *\$tag* entspricht hierbei dem Shortcodenamen, der von Benutzer verwendet wird (Parameter 1 in *add_shortcode*)

⁴⁴Vgl. MULLENWEG (abgerufen am 12.08.2012).

- Beispielsweise lässt sich diese Funktion verwenden, wenn auf einer Seite ein oder mehrere Shortcodes eingebunden sind und eine davon temporär oder für immer deaktiviert werden soll, ohne das entsprechende Plugin zu deinstallieren.

3. **function remove_all_shortcodes()**

- Diese Funktion deaktiviert im Gegensatz zu *function remove_shortcode(\$tag)* alle Shortcodes auf einer Seite.

4. **function do_shortcode(\$content)**

- Die Funktion `do_shortcode($content)` eignet sich dazu, in einem Post andere Funktionen zusätzlich einzubinden. Beispielsweise könnte dies ein Kontaktformular sein.

Weitere Funktionen sind auf der offiziellen Shortcode-API seite von Wordpress zu finden (siehe http://codex.wordpress.org/Shortcode_API#Function_reference)

6 Datenbankzugriff über Plugin-API

In diesem Kapitel geht es um die Interaktion zwischen Datenbank und Plugin. Es wird allgemein auf die entsprechende Datenbankschnittstelle sowie einigen Beispielen zur Manipulation und Ausgabe von SQL-Anweisungen eingegangen. Weiterhin wird darauf eingegangen, wie Angriffe auf die Datenbank verhindert werden können (vgl. Abschnitt 6.2).

6.1 Die Datenbankschnittstelle WPDB

Wordpress bietet eine Schnittstelle an, mit der Datenbanken angesprochen werden können. Diese heißt *wpdb* - eine Abkürzung für *Wordpress Database* - und basiert auf *ezSQL*.⁴⁵

Laut Vincent⁴⁶ ist *ezSQL* eine OpenSource Datenbankklasse, welche auf PHP basiert und dazu dient, zwischen einem PHP-Skript und einer Wordpress-Datenbank zu interagieren.

Grundvoraussetzung ist eine globales Objekt, über welche dann auf die Datenbank zugegriffen werden kann. Diese nennt sich *\$wpdb* und soll verhindern, dass Funktionen aus der WPDB-Klasse direkt aufgerufen werden können. Dies kann zu Kompatibilitätsproblemen führen und wird von Wordpress nicht unterstützt.

Bevor sich den Möglichkeiten dieses Objektes zugewendet wird, werden noch zwei wichtige Informationen über die WPDB-Schnittstelle aufgezeigt. Zum einen kann diese Schnittstelle nur mit der Wordpress-Datenbank kommunizieren, zum anderen ist es manchmal auf notwendig mit anderen Datenbanken zu interagieren.

Wenn mit einer anderen oder mehreren Datenbanken interagieren werden soll, ist das Plugin *Hyperdb* hilfreich. Dieses ist unter <http://wordpress.org/extend/plugins/hyperdb/> zu finden.⁴⁷

In diesem Tutorial wird mit der Standard-Datenbank von Wordpress gearbeitet und somit muss kein zusätzliches Plugin verwendet werden.

6.1.1 Syntax und grundsätzliche Beispiele

Nachdem die Theorie soweit besprochen wurde, geht es nun um den Syntaxaufbau und die ersten Beispiele.

Die allgemeine Syntax einer Abfrage an die Datenbank ist in Listing 18 dargestellt.

```
1 <?php
2 ...
3 $wpdb->query ( 'query' );
4 ...
5 <?>
```

Listing 18: Allgemeine Syntax

Wie sich erkennen lässt, ist der erste Teil einer jeden Datenbankinteraktion immer das *\$wpdb*-Objekt. Gefolgt wird das Objekt von einem *->*. Hierbei handelt es sich um einem Operator, welcher eine Methode aufruft. Dieser ist vergleichbar mit dem Punktoperator im objektorientierten

⁴⁵Vgl. MULLENWEG (abgerufen am 16.08.2012), Abschnitt Interfacing With the Database.

⁴⁶Vgl. VINCENT (abgerufen am 22.08.12).

⁴⁷Vgl. MULLENWEG (abgerufen am 16.08.2012), Abschnitt Notes On Use.

Java. Anschließend wird die Funktion *query* aufgerufen. Gefolgt von einem Datenbankbefehl, der in diesem Beispiel in Klammern und Hochkommata steht. Insgesamt sendet diese Anfrage einen Integer zurück, welcher beispielsweise die Anzahl der selektierten Zeilen zurückgibt. Wenn die Funktion einen Fehler enthält, wird automatisch ein *FALSE*, also eine *NULL* zurückgegeben.⁴⁸ Soviel erstmal zum ersten Eindruck. In den nächsten Unterkapiteln geht es um einzelne SQL-Anweisungen und wie diese ausgeführt werden können. Anschließend wird im Kapitel 6.1.2 die Theorie mit Praxisbeispielen aus dem Plugin Mentoren-Suche abgerundet.

6.1.1.1 SELECT von Variablen

Als erstes weiterführendes Beispiel, sollen bestimmte Variablen mittels einer SELECT-Anweisung aus der Datenbank ausgelesen werden.

Dafür wird die sogenannte *get_var*-Funktion verwendet. Diese kann eine einzelne Variable aus der Datenbank zurückgeben. Sie eignet sich also perfekt für unsere oben erwähnte Anforderung. Wenn kein Ergebnis vorhanden ist, wird auch hier eine *NULL* zurückgegeben.

Kommen wir an dieser Stelle nun zu dem Syntaxaufbau, welches in Listing 19 dargestellt ist.

```
1 <?php
2 ...
3 $wpdb->get_var( 'query', column_offset, row_offset );
4 ...
5 <?
```

Listing 19: Selektion einer Variablen

Dieser Syntaxaufbau aus Listing 19 ist wie folgt zu beschreiben:

1. **query**

- Bei *query* handelt es sich um einen String, welcher ausgeführt werden soll.

2. **column_offset**

- Ein Integerwert, welcher die gewünschte Spalte auswählt. Hierbei stellt der Wert 0 die erste Zeile dar und ist auch gleichzeitig der Initialisierungswert.

3. **row_offset**

- Hierbei handelt es sich um einem String oder integer, welcher die gewünschte Zeile angibt. Auch hier ist der Initialisierungswert 0 und stellt die erste Zeile dar.

Die Informationen wurden aus der offiziellen Wordpress-API genommen und entsprechend aufbereitet.⁴⁹ Im nächsten Abschnitt geht es um das Selektieren von Zeilen.

6.1.1.2 SELECT von Zeilen

Um eine Zeile in einer SELECT-Abfrage einzubinden, wird die Funktion *get_row* verwendet. Diese kann eine Zeile als einzelnes Objekt oder als Array wiedergeben. Wenn mehr als eine Zeile zurückgegeben wird, wird nur die erste ausgewählte Zeile ausgegeben - die anderen allerdings

⁴⁸Vgl. MULLENWEG (abgerufen am 16.08.2012), Abschnitt Run Any Query on the Database.

⁴⁹Vgl. MULLENWEG (abgerufen am 16.08.2012), SELECT a Variable.

im Zwischenspeicher für den späteren Gebrauch gespeichert. Falls kein Ergebnis gefunden wird, wird auch hier eine *NULL* zurückgegeben wird.⁵⁰ Um diese Theorie zu verdeutlichen, wird nun die Syntax erläutert.

```
1 <?php
2 ...
3 $wpdb->get_row( 'query' , output_type , row_offset );
4 ...
5 ?>
```

Listing 20: Selektion einer Zeile

Gehen wir nun auf die einzelnen Syntaxelemente ein:

1. **query**

- Auch hier handelt es sich wieder um einen String, der ausgeführt werden soll.

2. **output_type**

- Hier gibt es drei vordefinierte Konstanten, von denen jeweils eine eingesetzt werden darf.
 - a) Objekt: Ergebnis wird als Objekt wiedergegeben
 - b) `Array_A`: Ein assoziatives (also mit nichtnumerischen Schlüsseln) wird zurückgegeben
 - c) `Array_N`: Ein numerisches Array wird zurückgegeben (also mit numerischen Schlüsseln)

3. **row_offset**

- Bezeichnet die gewünschte Zeile. Auch hier ist die Initialisierungswert *NULL* und stellt zugleich die erste Zeile dar.

Im nächsten Abschnitt wird das selektieren von Spalten besprochen.

6.1.1.3 SELECT von Spalten

Um Spalten zu selektieren, gibt es die Funktion `get_col`, welche ein Array ein Array der ausgewählten Spalten zurückgibt. Bei einem nicht vorhandenem Ergebnis leeres Array zurückgegeben.⁵¹ Die Syntax lautet folgendermaßen:

```
1 <?php
2 ...
3 $wpdb->get_col( 'query' , column_offset );
4 ...
5 ?>
```

Listing 21: Selektion von Spalten

Es handelt sich hierbei um eine relativ kleine Syntax, welche nun im Detail besprochen wird:

⁵⁰Vgl. MULLENWEG (abgerufen am 16.08.2012), SELECT a Row.

⁵¹Vgl. MULLENWEG (abgerufen am 16.08.2012), SELECT a Column.

1. **query**

- So wie bei den oben genannten Formen, handelt es sich hier auch um den auszuführenden String.

2. **column_offset**

- Der `column_offset` bezeichnet die ausgewählte Spalte in Form eines Integers und wird mit `NULL` initialisiert.

Soviel zu den Select-Befehlen. Nun wird auf zwei weitere Abfragen eingegangen; nämlich `INSERT` von Spalten und `UPDATE` von Spalten.

6.1.1.4 **INSERT von Spalten**

Für das Einfügen von Spalten bieten sich in der `wpdb`-Umgebung die Funktion `insert` an. Diese besteht aus drei Parametern und wird nun anhand der Syntax erläutert⁵² :

```
1 <?php
2 ...
3 $wpdb->insert( $table , $data , $format );
4 ...
5 ?>
```

Listing 22: Insert von Spalten

Soviel zum allgemeinen Syntaxaufbau. Die einzelnen Elemente bedeuten dabei folgendes:

1. **\$table**

- Der Name der Tabelle, in die Daten eingefügt werden sollen. Dieser erste Parameter wird in Form des Typs *String* angegeben.

2. **\$data**

- Beschreibt die konkreten Daten, die eingefügt werden sollen. Dabei handelt es sich um ein Objekt vom Typ *array*

3. **\$format**

- Dies ist eine optionale Angabe, die für das Format aus `$data` steht. Das bedeutet, wenn in `$data` nur Strings stehen, kann hier `%s` gesetzt werden und alle Inhalte werden als Strings behandelt. Hiermit lässt sich der genaue Inhalt von `$data` angeben.

6.1.1.5 **UPDATE von Spalten**

Der letzte der SQL-Anweisungen, der hier vorgestellt wird, ist der Update-Befehl⁵³. Dieser eignet sich dafür, eine Zeile zu verändern. Dabei gibt der Befehl ein *false* bei einem aufgetretenen Fehler zurück.

Die allgemeine Syntax lässt sich wie folgt beschreiben:

⁵²Vgl. MULLENWEG (abgerufen am 16.08.2012), `INSERT rows`.

⁵³Vgl. MULLENWEG (abgerufen am 16.08.2012), `UPDATE rows`.


```
1 <?php
2 ...
3 $wpdb->update( $table , $data , $where , $format = null , $where_format =
    null );
4 ...
5 ?>
```

Listing 23: Update von Spalten

1. **\$table**

- Auch hier ist mit dieser Anweisung die Tabelle gemeint, auf die sich bezogen wird.

2. **\$data**

- Auch hier handelt es sich um die Daten, die verändert werden sollen.

3. **\$where**

- Bei der where-Angabe handelt es sich um eine spezielle Anweisung, was in einer Spalte oder Zeile geändert werden soll. Hier ist es auch möglich, mehrere Angaben mit dem logischen AND zu verknüpfen.

4. **\$format**

- format beschreibt die Art der ausgewählten Daten - genauer gesagt um welchen Datentypen es sich handelt. Bei einem String würde ein %s angegeben werden und alle Inhalte wie Strings behandelt. Diese Angabe ist optional.

5. **\$where_format**

- Auch hierbei handelt es sich um eine optionale Angabe, welche beschreibt, welche Art von Format für den Update-Befehl ausgewählt werden soll. Auch diese Angabe ist optional und für einen String für %s dienen und alle anderen Datentypen ignoriert werden.

Weitere Informationen befinden sich unter der Adresse http://codex.wordpress.org/Class_Reference/wpdb. Hier sind auch einige Beispiele vorhanden. Konkrete Beispiele aus dem Mentoren-Plugin finden sich hingegen im nächsten Kapitel 6.1.2.

6.1.2 Beispiele aus dem Plugin Mentoren-Suche

Nachdem in Kapitel 6.1.1 einige Befehle angesprochen wurden, werden diese nun anhand von Praxisbeispielen aus dem Plugin *mentoren-suche* vorgestellt und erläutert. Dabei wird nun so vorgegangen, dass zuerst ein Beispiel genannt, welches im Anschluss erläutert wird.

6.1.2.1 Beispiel SELECT von Zeilen

Eine normale SELECT-Anfrage stellt die einfachste aller Anfragen dar. Dieser soll einen bestimmten Inhalt aus der Datenbank auslesen. Dabei kommt der SELECT-Befehl zum Einsatz (siehe dazu 6.1.1.2).

In dem in Listing 24 dargestellten Auszug aus der Funktion `ms_showDeleteDataForm()`, welche für das Anzeigen eines Datensatzes für das Löschen-Formulars (näheres zu Formulare in Kapitel 7) programmiert wurde, besteht aus 2 Teilen.

```
1 <?php
2 ...
3 function ms_showDeleteDataForm()
4 {
5
6     $sql_mentorID="SELECT┐mentees┐mid┐".
7         "FROM┐".MENTEE┐TABLE.
8         "┐WHERE┐mentees┐matrikelnr┐='".
9         $delete_mentee_id." '";
10    $rs_mentorID=$wpdb->get_row($sql_mentorID);
11 }
12 ...
13 <?
```

Listing 24: Beispiel SELECT von Zeilen aus mentoren-plugin

Der erste Teil definiert einen Befehl zum selektieren eines bestimmten Datensatzes, der zweite in Zeile 10 dargestellte Befehl gibt dann das Ergebnis wieder. Dieses kann dann mit der Variable `$rs_mentorID` weiter verwendet werden.

Der Befehl wurde entsprechend verschachtelt, um potenzielle SQL-Injections zu vermeiden. Näheres dazu unter Abschnitt 6.2

6.1.2.2 Beispiel SELECT von Spalten

Im Mentoren-Plugin wurden keine Abfragen zum selektieren von Spalten verwendet. Deshalb wird an dieser Stelle auf das Beispiel aus dem offiziellen Wordpress-Codex verwiesen. Dieses ist unter

1. http://codex.wordpress.org/Class_Reference/wpdb#SELECT_a_Column

zu finden.

6.1.2.3 Beispiel INSERT von Spalten

Der INSERT-Befehl dient dem Einfügen von Spalten und besteht insgesamt aus 3 Parametern. Diese wurde bereits näher in Abschnitt 6.1.1.4 erläutert.

```
1 <?php
2 ...
3 function ms_m_data_insert() {
4
5     global $wpdb;
6
7     if(!empty($_POST['ms_m_name']) && // ...)
8     {
9         $name = $_POST['ms_m_name'];
```

```
10     $vorname = $_POST[ 'ms_m_vorname' ];
11     // ...
12
13     $rows_affected = $wpdb->insert( MENTOR_TABLE, array( 'mentoren_name' =>
        $name, 'mentoren_vorname' => $vorname, ... ) );
14
15     } else {
16         echo " // ... Error ";
17     }
18     ...
19     <?
```

Listing 25: Beispiel INSERT von Spalten aus mentoren-plugin

Der in Listing 25 dargestellte Quellcode ist ein Teil der Funktion *ms_m_data_insert()*, welche dazu dient nach einem Submit-Button, die einzelnen Mentoren in die Mentoren-Tabelle der Datenbank zu speichern. Es handelt sich hierbei nur um ein gekürztes Beispiel, um sich auf den wesentlichen Inhalt zu konzentrieren.

Im Grunde geht es darum, dass bei der Eingabe des Namen und Vornamen des Mentors in der Eingabemaske, dieser in die Datenbank eingetragen wird.

Dafür wird in Zeile 13 die INSERT-Methode verwendet. Die betreffende Tabelle heißt MENTOR_TABLE und die konkreten Daten werden mittels eines Arrays eingetragen.

Zur Vollständigkeit wird an dieser Stelle angemerkt, dass falls die Eingabe des Mentors nicht korrekt ist, ein entsprechender Fehler (siehe Zeile 16) ausgegeben wird.

6.1.2.4 Beispiel DROP von Spalten

Da keine Update-Anweisung im Plugin Mentoren-Suche verwendet wurde, wird an dieser Stelle auf die offizielle Wordpress-Dokumentation verwiesen (siehe dazu http://codex.wordpress.org/Class_Reference/wpdb#UPDATE_rows)

Stattdessen wird aber kurz der Befehl zum Löschen von Tabellen vorgestellt: Der DROP-Befehl⁵⁴.

Die in Listing 26 verwendete Funktion dient dazu, das Plugin komplett zu löschen. Dabei wird - neben anderen kleineren Anweisungen - auch die Tabelle gelöscht. So wird nicht nur nach dem Deaktivieren, das Plugin aus dem Front- und Backend, sondern auch aus der Datenbank gelöscht.

```
1 <?php
2 ...
3 function pluginUninstall() {
4     global $wpdb;
5
6     //Tabellen loeschen:
7     $sql = "DROP TABLE ".MENTOR_TABLE;
8
9     $wpdb->query( $sql );
```

⁵⁴Vgl. GARDNER (abgerufen am 02.12.12), How to Get Your WordPress Plugin To DROP TABLE From The Database.

```
10
11     $sql = "DROP TABLE ".MENTEE_TABLE;
12     $wpdb->query( $sql );
13 }
14 ...
15 ?>
```

Listing 26: Beispiel DROP-Anweisung aus mentoren-plugin

Was in diesem Ausschnitt nun passiert ist folgendes: Nachdem ein bereits erzeugtes `wpdb`-Objekt zugegriffen wurde, wird in der Zeile 7 die Anweisung `DROP_TABLE` ausgeführt. Die allgemeine Syntax lautet:

- "DROP TABLE ".TABELLENNAME;

Diese bewirkt, dass die Tabelle `MENTEE_TABLE` gelöscht wird. In Zeile 12 wird die Anweisung dann schlussendlich ausgeführt. Soviel zu den Beispielen für die SQL-Anweisungen im Zusammenhang mit dem `$wpdb`-Objekt von Wordpress.

Weitere Informationen werden unter den folgenden Internetadressen gefunden:

1. http://codex.wordpress.org/Class_Reference/wpdb

Im nächsten Kapitel werden noch in 6.2 das Verhindern von SQL-Injections angesprochen.

6.2 Verhindern von SQL Injections

In diesem Abschnitt wird erläutert, wie sich SQL-Injections in der Pluginentwicklung verhindern lassen. Dies ist das Thema dieses letzten Abschnitts des Kapitels 6.

SQL Injections bezeichnen einen Angriff auf eine SQL-Datenbank, indem SQL-Anfragen so manipuliert werden, dass Sie vertrauliche Daten ausspähen können. Es handelt sich also hierbei um eine Sicherheitslücke, welche aber mit bestimmten Anweisungen verhindert werden können.⁵⁵

Diese werden an dieser Stelle vorgestellt. Eine einfache und effektive Lösung, um solche Angriffe zu verhindern, ist es, SQL-Abfragen (zu Deutsch *SQL-Abfragen*) mit der Prepare-Anweisung zu versehen. Diese Anweisung bereinigt und entgeht der eigentlichen SQL-Anweisung.

Praktisch betrachtet, soll an dieser Stelle in Beispiel im Listing 27 aufgezeigt werden.

```
1 <?php
2
3 //1. Abfrage ohne prepare()-Statement
4 SELECT 'post_title'
5 FROM $wpdb->posts
6 WHERE 'post_author' = 1
7
8 //2. Abfrage mit prepare()-Statement in Datei 1
9 $sql = "SELECT 'post_title'
10 FROM $wpdb->posts
11 WHERE 'post_author' = %d";
12
```

⁵⁵Vgl. WIKIPEDIA.DE (abgerufen am 01.12.12), SQL-Injection.

```
13 //Datei 2 mit prepare()
14 <?php
15     $id = 1;
16     $safe_sql = $wpdb->prepare($sql,$id);
17     $posts = $wpdb->getresults ($safe_sql);
18 ?>
19
20 //EOF
21 ?>
```

Listing 27: Beispiel für SQL-Injection-Verhinderung

Dieses Beispiel gibt die Titel aller Autoren aus, welche die ID 1 haben. Die Abfrage wird dabei auf die Tabelle *posts* bezogen.

Nun zur Erläuterung:

1. Im ersten Beispiel ohne dem prepare-Statement, wird die SQL-Anweisung sozusagen im Klartext geschrieben: Es ist möglich, die Autoren-ID mitzulesen und entsprechend zu manipulieren.
2. Im zweiten Beispiel wird anders verfahren. Die eigentliche SQL-Anweisung wurde mit Platzhaltern ausgestattet (siehe Zeile 11: %d).

Im zweiten Teil dieses Beispiels wird von einer anderen PHP-Datei auf die Anweisung zugegriffen. Dabei wird zuerst eine Variable vom Namen %id angelegt und mit dem Wert 1 initialisiert.

Anschließend wird mittels des *prepare*-Statements die Anweisung in Zeile 9 mittels *\$sql* und *\$id* aufgerufen.

Weiterhin wird dann mittels *getresults* das Ergebnis der *\$safe_sql*-Anfrage in *posts* gespeichert und kann weiterverarbeitet werden.

Der Vorteil an dem Prepare-Statement ist also, dass nur Vorlagen an Abfragen geschrieben werden, aber mittels entsprechender Variablen dynamisch verwendet werden können und so eine SQL-Injection nicht mehr so einfach möglich ist.⁵⁶

Wie dies genauer im Mentoren-Plugin aussieht, ist in Listing 28 dargestellt.

```
1 <?php
2
3     global $wpdb;
4
5     $sql="SELECT _ms.mentees_matrikelnr, " // ...
6         FROM " .MENTEE_TABLE. " / ...
7         WHERE ms.mentees_name=%s AND " . "ms.mentees_matrikelnr=%s ";
8
9     _$_result=$wpdb->get_row(_$wpdb->prepare($sql,$name,$matrikelNr));
```

Listing 28: Verhindern einer SQL-Injection aus mentoren-suche

⁵⁶Vgl. WILLIAMS/RICHARD/TADLOCK (2011), Seite 157 - 158.

Bei diesem Beispiel ist zu beachten, dass die *prepare*-Anweisung in der *get_row*-Abfrage verschachtelt ist. Trotzdem wird auch hier das *prepare*-Statement ausgeführt und kann anschließend über *\$result* weiterverwendet werden.

Weitere Informationen finden sich unter:

1. http://codex.wordpress.org/Class_Reference/wpdb#Protect_Queries_Against_SQL_Injection_Attacks

7 Formular

Dieses Kapitel beschäftigt sich mit der Verwendung von Formularen in Wordpress. Dazu wird neben dem Begriff eines Formulars auch auf Beispiele aus der Mentoren-Suche zurückgegriffen.

7.1 Was ist ein Formular?

Um überhaupt Formulare in Wordpress entwickeln zu können, soll an dieser Stelle zuerst eine Definition erwähnt werden.

Laut SelfPHP⁵⁷ handelt es sich bei einem Formular um eine Maske, in der ein Benutzer Daten eingeben kann. Diese werden dann von einem Server verarbeitet und wenn erforderlich zurückgegeben. Dabei wird das Formular mittels dem HTML-Tag *form* realisiert. Dabei kommen die beiden HTML-Methoden POST (vgl. Abschnitt 7.1.1) und GET (vgl. Abschnitt 7.1.2) zum Einsatz. Diese werden nun erläutert.

7.1.1 POST-Methode

Einer der beiden Methoden⁵⁸, um mit Formularen zu arbeiten, ist die POST-Methode. Diese wird dann eingesetzt, wenn ein Client eine Anforderung an einen Server sendet, dass weitere Daten übertragen werden sollen. Dabei sollte beachtet werden, dass die gesendeten Daten Teil von der vom Server festgelegten URI ist.

Der Zweck von POST liegt darin, Formulardaten an den Server zu übertragen.

7.1.2 GET-Methode

Hingegen zur POST-Methode ist die GET-Methode⁵⁹ dafür da, alle möglichen Informationen mittels der Ergebnis-URI zu identifizieren. Eine komplette URI besteht demnach aus 4 Bestandteilen:

1. Die URI
2. Die URL
3. Einem Fragezeichen, welches als Trennzeichen verwendet wird
4. Den gesendeten Daten

Insgesamt lässt sich feststellen, dass POST-Methode die gesendeten Daten im Body (adressiert durch Header) sendet, während die GET-Methode diese als Teil der URL betrachtet und mit anhängt.

Abschließend lässt sich formulieren, dass für das Plugin Mentoren-Suche die Post-Methode verwendet wurde, da es sich um vertrauenswürdige Daten handelt und diese durch GET manipulierbar wären - genauer gesagt durch die Manipulation der URL. Aus diesem Grunde wird in den folgenden Beispielen auch nur die POST- und Wordpress-Methode angesprochen. Der zweiten wird sich im nächsten Abschnitten gewidmet.

⁵⁷Vgl. SELFPHP.INFO (abgerufen am 22.02.13).

⁵⁸Vgl. SELFPHP.INFO (abgerufen am 22.02.13).

⁵⁹Vgl. SELFPHP.INFO (abgerufen am 22.02.13).

7.2 Wordpress-Formulare

Auch Wordpress bietet eine Methode an, um mit Formularen zu arbeiten. Es handelt sich hierbei um eine vordefinierte Action von Wordpress, um beispielsweise Mentoren mit Vor- und Nachname sowie einem Bild und E-Mail-Adresse einzutragen.⁶⁰

Die genauere Syntax ist in der folgenden Aufzählung dargestellt.

1. do_action

- Führt einen Hook aus, der mittels `add_action` (vgl. Abschnitt 3.5.1) erstellt wurde.
- Die Syntax ist folgendermaßen definiert
 - `do_action($tag, $ arg);`
 - * `$tag` beschreibt des auszuführenden Hook und muss angegeben werden.
 - * `$arg` Beschreibt einen optionalen Parameter, welcher eine Liste von anzugebenen Argumenten für den Hook darstellt.

Nachdem die beiden grundlegenden Techniken besprochen wurden, werden im nächsten Kapitel die Theorie mit der Praxis angereichert und Beispiele erläutert.

7.3 Beispiel Formular aus Mentoren-Suche

An dieser Stelle werden nun Beispiele für Formulare aus der Mentoren-Suche gezeigt und erläutert.

Dabei wird zuerst ein Beispiel für ein PHP-Formular und abschließend die Wordpress-Methode vorgestellt.

7.3.1 Beispiel für PHP-Formular

In diesem Unterkapitel geht es um ein PHP-Beispiel, welches in Listing 30 dargestellt ist.

```

1 function ms_getOptionsPage()
2 {
3     echo "<h2>".__( 'Frontend-Seiten: ', 'mentoren-suche' )."</h2>";
4     echo "<p>".__( 'ANLEITUNG_ZUR_BENUTZUNG. ', 'mentoren-suche' )."</p>";
5     echo "<form action=\"\"\" method=\"post\">
6     \<table>
7     \<tr>
8     \<td colspan=2>Suchformular
9     \</tr>
10    \</table>
11
12    \<tr>
13    \<td colspan=2><input type=\"submit\"
14    \<name=\"ms_options_save\"
15    \<value=\"\".__( 'Speichern ', 'mentoren-suche' ).\">
16    \</td>

```

⁶⁰Vgl. MULLENWEG (abgerufen am 12.02.13).


```

17 <td></td></tr>
18
19 </table>
20 </form>" ;
21
22 if ( isset ( $_POST[ 'ms_options_save' ] ) )
23 {
24     ms_options_update ( ) ;
25     echo "<h2 class=\"ms_info_message\">".
26         __( 'Bitte aktualisieren Sie die Browseransicht (z.B.
27             mit F5) , um die veränderten Daten zu
28             sehen.' , 'mentoren-suche' ) . "</h2>" ;
29 }
30 }
31 }

```

Listing 29: Beispiel Formular in Wordpress

Dieses Beispielformular aus Listing 30 ist in verkürzter Art und Weise dargestellt, würde aber insgesamt wie in Abbildung 11 aussehen.

Frontend-Seiten:

Geben Sie in die Textfelder den (absoluten) Pfad zur Seite und eine Beschreibung (Linktext) an. - Es erscheint ein Link mit der angegebenen Beschreibung, auf der entsprechenden Seite.

Suchformular:	URL: <input type="text"/>	Linktext: <input type="text"/>
Mentoren-Details:	URL: <input type="text"/>	Linktext: <input type="text"/>

Hier können Sie die Kontaktdaten der Person einpflegen, die es bei Problemen zu kontaktieren gilt.

Fehler-Seite:	Name: <input type="text"/>	E-Mail: <input type="text"/>
----------------------	----------------------------	------------------------------

Abbildung 11: Beispielformular mittels PHP

Der Zweck dieses Formulars ist es, Einstellungen für das Plugin vorzunehmen. Dies geschieht in Form von Textfeldern, welche entsprechend ausgefüllt werden müssen.

Erkennen lassen sich in Zeile 5 und Zeile 22 - 29 die POST-Methode.

Nachdem die einzelnen Daten in die dafür vorgesehenen Felder eingetragen und der Speicher-Button aktiviert (*ms_options_save*) wurde, wird in Zeile 22 die Funktion *ms_options_update* aufgerufen. Der Zweck hinter dieser Funktion liegt darin, die Einstellungen für die Daten des geänderten Mentoren oder Mentees zu ändern.

So würde ein Beispiel für eine Post-Methode in der Mentoren-Suche aussehen. Weitere Informationen zu dem Thema finden sich unter:

- <http://www.selfphp.info/praxisbuch/index.php>
- http://www.phpbox.de/php_tutorials/formularversenden1.php

7.3.2 Beispiel für Wordpress-Formular

Nachdem ein Beispiel mit der POST-Methode gezeigt und erläutert wurde, wird nun die Wordpress-Variante behandelt.

Wie schon angesprochen, wird bei dieser Art die `do_action`-Anweisung verwendet (vgl. 7.2). Diese ist beispielhaft in Listing 30 dargestellt.

```
1 function ms_row_new() {
2
3     global $wpdb;
4     global $title;
5     echo "<h1>". $title . "</h1>";
6
7     echo "<div id=\"ms_row_new\">
8         <div class=\"register-form\">
9             <div class=\"title\">
10                 <h3>.( 'Mentor eintragen ', 'mentoren-suche ' ) . "</h3>
11             </div>
12             <form
13                 // Formular
14
15             do_action( 'ms_insert_mentor ' );
16
17             echo " <input type=\"submit\" value=\"Eintragen\" id=\"register\"
18                 />
19             </form>
20         </div>
21     </div>";
22
23     ...
24     ?>
```

Listing 30: Beispiel Formular in Wordpress

Die in Listing 30 dargestellte Funktion hat den Zweck, die Oberfläche für "Datensatz eintragen" funktional darzustellen. Dies ist in Abbildung 12 bildhaft dargestellt.



Mentor eintragen

Vorname:

Name:

E-Mail:

Bild:

Abbildung 12: Beispielformular mittels `do_action`

Was sich direkt erkennen lässt, sind die beiden Form-Tags in den Zeilen 12 und 19. Dazwischen befindet sich zum einen das Formular, zum anderen aber auch ein Button (*Eintragen*) zum Bestätigen der Eingabe und in Zeile 15 die `do_action`-Anweisung.

Was hier nun passiert ist einfach zu erklären: Mittels der *do_action*-Anweisung wird die Action *ms_insert_mentor* aufgerufen. Diese wiederum ruft in der Hauptdatei *mentoren_suche.php* die Action *ms_m_data_insert* auf (siehe Listing 31).

```
1 <?php
2 ...
3 add_action( 'ms_insert_mentor', 'ms_m_data_insert' );
4 ...
5 <?
```

Listing 31: Ausgeführte Action der *do_action*-Anweisung

Diese wiederum ist dafür verantwortlich, nach dem Bestätigen des Submit-Buttons aus der Zeile 17 des Listing 30, die angegebenen Mentoren in die Mentorentabelle der Datenbank einzutragen. Vollständigerweise ist diese Funktion in Listing 32 dargestellt und wird anschließend kurz erläutert.

```
1 function ms_m_data_insert() {
2
3     global $wpdb;
4
5     if ( isset( $_POST[ 'ms_m_name' ] ) and isset( $_POST[ 'ms_m_vorname' ] ) and
6         isset( $_POST[ 'ms_m_email' ] ) ) {
7
8         if ( !empty( $_POST[ 'ms_m_name' ] ) && !empty( $_POST[ 'ms_m_vorname' ] ) && !
9             empty( $_POST[ 'ms_m_email' ] ) ) {
10
11             {
12                 $name = $_POST[ 'ms_m_name' ];
13                 $vorname = $_POST[ 'ms_m_vorname' ];
14                 $email = $_POST[ 'ms_m_email' ];
15                 $bild = $_POST[ 'ms_m_bild' ];
16
17                 $rows_affected = $wpdb->insert( MENTOR_TABLE, array( 'mentoren_name' =>
18                     $name, 'mentoren_vorname' => $vorname, 'mentoren_email' => $email, '
19                     mentoren_bild' => $bild, ) );
20
21             } else {
22                 ... //Fehlermeldung
23             }
24
25         }
26     }
27 }
```

Listing 32: Aufgerufene Funktion der Action

Insgesamt besteht die Funktion aus zwei IF-Abfragen. Dabei soll die erste (Zeile 5) laut PHP.net⁶¹ prüfen, ob eine Variable existiert und nicht NULL ist. Falls dies der Fall sein sollte, wird die nächste IF-Abfrage (Zeile 7) prüfen, ob die Inhalte des Formulars befüllt sind - spricht nicht leer sind. Anschließend wird, wenn die Bedingung erfüllt ist, die entsprechenden Daten des Formulars

⁶¹Vgl. GROUP (abgerufen am 15.02.13).

in Variablen übertragen (Zeile 9 - 12) und in Zeile 14 dann in die Datenbank eingetragen. Bei dem nicht Zutreffen einer Bedingung wird automatisch die Funktion beendet.

Nachdem die beiden Versionen von Formularen vorgestellt wurden, kann sich formulieren lassen, dass die PHP-Version zwar um einiges kürzer ist als die von Wordpress. Beide sind allerdings funktional gesehen gleichwertig zu betrachten.

Damit soll das Kapitel Formulare beendet sein und dem nächsten Kapitel zugewandt. Dieses handelt von der Internationalisierung und Lokalisation von Wordpress-Plugins.

8 Internationalisierung und Lokalisierung

In diesem Kapitel wird die Internationalisierung und Lokalisierung von Wordpressplugins beschrieben und anhand von Beispielen erläutert. Dazu wird erst eine Einleitung in Abschnitt 8.1 gegeben, die die Begrifflichkeiten und den Sinn dieser Technik vorstellt. Im Abschnitt 8.2 wird beschrieben, wie die Lokalisierung (innerhalb des Programmcodes) erfolgt. Danach erfolgt die Beschreibung im Abschnitt 8.3, wie eine Textdomäne geladen werden kann. Der Abschnitt 8.4 beschreibt, wie die eigentlichen Texte (Zeichenketten) übersetzt werden. Abschließend wird im Abschnitt 8.5 kurz gezeigt, wie die Sprache des Wordpress-Systems konfiguriert werden kann.

8.1 Einführung

Bei der Internationalisierung und Lokalisierung handelt es sich um ein allgemeines Konzept in der Programmierung.

Nach Shirah⁶² handelt es sich bei der Internationalisierung (engl. internationalization) um einen „Prozess der Erstellung einer Software für den Einsatz im multinationalen Kontext“ (eigene Übersetzung). Neben der Unterstützung von unterschiedlichen Sprachen, ist die Software auch in der Lage, mit unterschiedlichen Datums-, Zeit-, Währungsformaten umzugehen. Dabei ist bedeutsam, dass dies ohne Eingriff in den Programmcode erfolgt. Unter Praktikern ist für den Begriff „Internationalization“ die Abkürzung „I18N“ (steht für: die 18 Buchstaben zwischen dem „I“ und dem „N“ des Wortes „Internationalization“) gebräuchlich.

Die Lokalisierung (abgekürzt häufig: „L10N“, nach ähnlicher Semantik wie bei dem Begriff „Internationalisierung“) ist als Bestandteil der Internationalisierung zu verstehen. Darunter ist die Vorbereitung des Programms für den Umgang mit unterschiedlichen Sprachen, Kulturen sowie Ländern gemeint. Shirah beschreibt sehr anschaulich, wie die Lokalisierung praktisch zu verstehen ist. Zur Verhinderung einer Verfälschung der Aussage, wird die entsprechende, englische Passage aus dem Originaltext zitiert: „Usually, though, true localization is achieved by core code that accesses locale, location, political, or other specific components and modules, along with translating text as appropriate for the audience“.

Die Lokalisierung kann demnach als eine Aktivität innerhalb der Internationalisierung verstanden werden: Hier geht es um die Übersetzung der vorbereitenden Daten in einem Nutzungskontext. Ein Benutzer greift mittels eines Browsers auf eine Seite zu und die Wordpress-Umgebung verwendet direkt die richtige Sprache für die Oberflächenelemente. Dabei ist zu beachten, dass die Lokalisierung immer lokal stattfindet. Dies bedeutet, dass die Spracheinstellungen in der Hauptkonfiguration von Wordpress (`wp-config.php`) eine wichtige Rolle spielen: Je nach konfigurierter Sprache, werden die entsprechenden Sprachdateien verwendet (sofern diese vorhanden sind). Bei dieser Art von Übersetzungen sind lediglich die Oberflächenelemente betroffen. Eine Übersetzung der Inhalte der Webseiten/Plugins (Einstellungsdaten, Beiträge, Seiten) erfolgt hierüber nicht. Dies muss über entsprechende Plugins geschehen, die es ermöglichen Inhalte für beispielsweise 'Artikel/Seiten in mehreren Sprachen abzulegen. Dazu stehen beispielsweise folgende Plugins zur Verfügung: *gTranslate* (Download (<http://wordpress.org/extend/plugins/qtranslate/screenshots/>)) oder *WPML* (Download unter: <http://wpml.org>).

⁶²Vgl. SHIRAH (abgerufen am 03.08.2012).

Wordpress nutzt im Speziellen das gettext-Framework (PHP), dass zur Internationalisierung einer Anwendung konzipiert ist⁶³. Weitere Informationen zum Framework unter: <http://www.php.net/manual/de/book.gettext.php>.

8.2 Lokalisierung

In diesem Abschnitt wird gezeigt, wie in Wordpress-Programmcode die Lokalisierung vorgenommen wird. Hierbei wird sich lediglich auf die Elemente der Oberfläche (Beschriftungen) beschränkt (vgl. Abschnitt 8.1).

Zur Lokalisierung werden spezielle Funktionen an die Stellen des Quellcode hinzugefügt, die ansonsten Zeichenketten (Beschriftungen von Oberflächenelementen, Fehlermeldungen) in eine Sprache enthalten würden.

Dabei sind folgende Funktionen zu nennen⁶⁴:

1. `__e($message,$domain)`

- Diese Funktion sucht das „Lokalisierungsmodul“ (eigene Übersetzung) nach einer Übersetzung, für die in der Variablen `$message` gespeicherten Zeichenkette. Wird eine entsprechende Übersetzung gefunden, wird diese auf dem Bildschirm ausgegeben (e steht für die echo-Anweisung). Sollte keine Übersetzung vorhanden sein bzw. gefunden werden, wird der Inhalt in der Variablen `$message` direkt ausgegeben. Der Parameter `$domain` steht für die Angabe einer Textdomäne, in der die Zeichenkette hier verwendet werden soll. Das Konzept der Textdomäne wird im Abschnitt 8.3 thematisiert.

Im Listing 33 ist ein Beispiel für die Verwendung der oben beschriebenen Funktion zu erkennen. Dort wird lediglich ein Hinweis lokalisiert, der sich auf dem CSV-Import des Mentoren-Suche-Plugins bezieht.

```
1 <?php
2 ...
3 __e( 'Bitte_eine_Datei_auswählen;hien_die_Mentee_Daten_beinhaltet.
      ', 'mentoren-suche' );
4 ...
5 ?>
```

Listing 33: Beispiel für die Verwendung von `__e()`

2. `__($message,domain)`

- Diese Funktion übernimmt dieselbe Aufgabe wie `__e()`. Auch gelten dieselben Erläuterungen zum Parameter `$domain`. Der Unterschied ist, dass der (übersetzte) Text nicht ausgegeben wird, sondern die Zeichenkette (string) zurückgegeben wird, sodass diese in einer return-Anweisung verwendet werden kann oder in einer Zeichenkette integriert werden kann.

Im Listing 34 ist ein Beispiel für die Verwendung der Funktion `__()` zu erkennen. Hier wird eine formatierte Fehlermeldung lokalisiert.

⁶³Vgl. GROUP (abgerufen am 28.08.12).

⁶⁴Vgl. BONDARI/GRIFFITHS (2011), Seite 238.

```
1 <?php
2 ...
3 echo "<h3 class=\"ms_error_message\">".__( 'Bitte alle
    Pflichtfelder ausfüllen!', 'mentoren-suche' ). "</h3>";
4 ...
5 ?>
```

Listing 34: Beispiel für die Verwendung von `__()`

Nach der Darstellung der beiden, wichtigsten Lokalisierungsfunktionen, sollen auch einige Hinweise zur praktischen Verwendung gegeben werden. Dazu liefern Bondari/Griffiths⁶⁵ einige „Best Practice“-Empfehlungen. Für die Entwicklung des Beispiel-Plugins „Mentoren-Suche“ waren drei Empfehlungen bedeutsam. Diese sollen nun kurz beschrieben werden:

1. Es sollen Leerzeichen am Anfang und am Ende der zu übersetzenden Zeichenkette vermieden werden. Dies kann überraschende Auswirkungen auf die Übersetzung der Zeichenkette haben. Denn aus der Programmierung ist bekannt, dass die Zeichenkette „Hallo Welt“ ungleich „ Hallo Welt“ ist.
2. Außerdem soll HTML und URLs in einer zu übersetzenden Zeichenkette vermieden werden. Als ein Grund kann sich vorgestellt werden, dass derjenige, der die Übersetzungen vornimmt (muss nicht unbedingt ein Programmierer sein, vgl. Abschnitt 8.4) auch daran denken müsste, die HTML-Befehle und URLs in der Übersetzung korrekt zu integrieren. Ein Beispiel für eine Fehlermeldung, die formatiert mit HTML (und CSS) wird, zeigt eine mögliche Realisierung (vgl. Listing 34). Es erfolgt eine Konkatenation mehrerer Zeichenketten. Die geöffneten und schließenden Tags sind jeweils eine Teil-Zeichenkette, die mit der zu übersetzenden Zeichenketten verknüpft wird.
3. Ferner sollen keine Variablen in der zu übersetzenden Phrase enthalten sein. Hierfür kann derselbe Grund angegeben werden, wie unter Punkt 2. Der Übersetzer müsste auch die Variablen wieder korrekt in die Übersetzung integrieren. Erfolgt dies nicht korrekt, könnte das Programm unsinnige Meldungen ausgeben.

Die Einbettung der Variablen in einer zu übersetzenden Phrase kann über Platzhalter in der Zeichenkette geschehen. Dies kann mit der `sprintf()`-Funktion realisiert werden, die eine formatierte Zeichenkette zurückliefert.⁶⁶

Das folgende Beispiel (vgl. Listing 35) demonstriert die Verwendung dieser Funktion im Lokalisierungskontext.

```
1 <?php
2 ...
3 echo "... "
4 sprintf( __( 'Mentor %s wurde mit seinem Nachnamen angelegt, bitte
    Mentoren Stammdaten vervollständigen', 'mentoren-suche' ),
    utf8_encode( $teile[0] ) )
5 ...
```

⁶⁵Vgl. BONDARI/GRIFFITHS (2011), Seite 240ff..

⁶⁶Vgl. GROUP (abgerufen am 28.08.12).

6 ?>

Listing 35: Beispiel für die korrekte Einbettung von Variablen mit der `sprintf()`-Funktion

Das aus dem „Mentoren-Suche“-Plugin entnommene Code-Fragment (vgl. Listing 35) wurde gekürzt (angedeutet mit ...). Denn es sollen hier nur die für die Erklärung benötigten Elemente enthalten sein. Es ist zu erkennen, dass innerhalb von `sprintf()` die Lokalisierungsfunktion `__()` verwendet wird. In dieser Funktion befindet sich die zu übersetzende Zeichenkette. Dort befindet sich für den Namen des Mentors der Platzhalter `%s`. Der entsprechende Wert wird vom Ausdruck zurückgeliefert, der nach dem (ersten) Komma angegeben ist: `utf8_encode($teile[0])`. Somit lässt sich sehr einfach, den Einsatz von Variablen in Zeichenketten vermeiden.

Abschließend soll auf die Online-Ressource verwiesen werden, die weitere Hinweise zur Lokalisierung bietet.: http://codex.wordpress.org/I18n_for_WordPress_Developers. Im nächsten Abschnitt 8.3 wird das Laden einer Textdomäne behandelt.

8.3 Laden einer Textdomäne

In diesem Abschnitt soll das Konzept der „Textdomäne“ beschrieben werden und wie solch eine zur Laufzeit geladen werden kann. In der (natürlichen) Sprache sind Wörter mehrdeutig und damit vom Kontext abhängig. Damit Wörter für einen geltenden Kontext übersetzt werden können, gibt es das Konzept der Textdomäne. Ein Beispiel soll die Erklärungen verdeutlichen: Das Wort *football* gibt es zwar im britischen und amerikanischen Englisch, wird allerdings je nach Land anders interpretiert. Die Textdomäne hilft die Bedeutung gleicher Wörter zu unterscheiden⁶⁷.

Im Folgenden soll beschrieben werden, wie eine Textdomäne im Quellcode geladen wird. Dazu soll anhand von Listings 36 (aus dem Plugin „Mentoren-Suche“) die Funktionsweise erläutert werden.

```
1 <?php
2 ...
3 add_action( 'init', 'ms_load_translation_file' );
4 ...
5 ...
6 function ms_load_translation_file() {
7     $plugin_path =
8         plugin_basename( dirname(__FILE__) . '/lang' );
9     load_plugin_textdomain( 'mentoren-suche', false,
10                             $plugin_path );
11 }
12 ...
13 ?>
```

Listing 36: Laden einer Textdomäne

⁶⁷Vgl. BONDARI/GRIFFITHS (2011), Seite 239.

Die erste relevante Anweisung (vgl. Listing 36) fügt der Wordpress-Engine ein Ereignis hinzu. Mit dem (vordefinierten) Ereignis `init` wird nachdem Laden von Wordpress die hier angegebene Funktion `ms_load_translation_file` aufgerufen. In dieser befindet sich der eigentliche Code für die Registrierung der Textdomäne: Die Funktion `plugin_basename` ermittelt den Pluginname. Das macht die Funktion dadurch, indem sie vom übergebenen Dateinamen/Verzeichnisnamen den Pluginnamen extrahiert⁶⁸.

Dazu ist es erforderlich, dass der gesamte Pfad zur Plugin-Datei übergeben wird. Dies geschieht mittels der PHP-Funktion `dirname`, die als Parameter hier die vordefinierte PHP-Konstante `__FILE__` übergeben bekommt. Die PHP-Konstante `__FILE__` enthält den absoluten Pfad zur Datei, in der diese Konstante aufgerufen wird⁶⁹.

Die Funktion `dirname` gibt, auf Basis des übergebenen Pfads, das übergeordnete Verzeichnis zurück⁷⁰. Per Konkatenation wird das dynamisch ermittelte Verzeichnis des Plugins mit dem Verzeichnisnamen „lang“ verknüpft. In diesem Verzeichnis sollen die Sprachdateien werden die Sprachdateien (vgl. Abschnitt 8.4) für das Plugin abgelegt. Der Name hätte beispielsweise auch anders gewählt werden können: „language“.

Zusammenfassend erledigt die erste Zeile folgendes: es wird mehr oder weniger dynamisch das Verzeichnis festgelegt/bestimmt, in dem die relevanten Sprachdateien liegen. Diese Information wird in die Variable `$plugin_path` gespeichert. Die Funktion `load_plugin_textdomain` ist für das Laden der Textdomäne zuständig. Zur eindeutigen Identifikation der Textdomäne wird die Domainbezeichnung (erster Parameter) angegeben (hier: `mentoren-suche`). Der Domainnamen darf einzig aus alphabetischen Zeichen, Bindestrichen und Unterstrichen bestehen. Bondari/-Griffiths empfehlen den Pluginnamen zu verwenden⁷¹.

Die Textdomäne wird als zweiter Parameter der Lokalisierungsfunktionen `_e()` und `__()` übergeben (vgl. Abschnitt 8.2).

Der zweite Parameter ist veraltet (deprecated) und soll nicht mehr verwendet werden. Dieser wird standardmäßig auf `false` gesetzt und wird hier nicht weiter betrachtet. Als dritten Parameter wird die in der Variablen `$plugin_path` gespeicherten Information (Pfad, in der die Sprachdateien abgelegt werden) übergeben.

In diesem Abschnitt wurde das Konzept der Textdomäne vorgestellt, dass zur eindeutigen Verwendung von übersetzenden Wörtern/Phrasen eingesetzt wird. Es wurde vorgestellt, wie solch eine Textdomäne im Quellcode registriert wird. Im nächsten Abschnitt wird beschrieben, wie die eigentlichen Sprachdateien (mit den Übersetzungen) erzeugt werden.

8.4 Anlegen der Übersetzung

In diesem Abschnitt wird vorgestellt, wie die eigentlichen Übersetzungen entstehen. Dazu ist es erforderlich, ein Programm zu nutzen, um die lokalisierten Phrasen/Begriffe im Quellcode, in die gewünschte Sprache zu übersetzen und in spezielle Sprachdateien abzulegen. Das hier eingesetzte Programm heißt „Poedit“ (Download unter: <http://www.poedit.net>). Neben diesem Programm, existieren weitere für diesen Zweck. Eine Liste findet sich unter: [http:](http://)

⁶⁸Vgl. MULLENWEG (abgerufen am 09.02.13).

⁶⁹Vgl. GROUP (abgerufen am 09.02.2013b).

⁷⁰Vgl. GROUP (abgerufen am 09.02.2013a).

⁷¹Vgl. BONDARI/GRIFFITHS (2011), Seite 240.

`//codex.wordpress.org/Translating_WordPress#Translation_Tools`.

Im Folgenden soll die Erstellung von Sprachdateien mittels „Poedit“ geschildert werden⁷²:

1. Starten des Tools „Poedit“
2. „Datei“ > „Neuer Katalog“. Es erscheint der Dialog „Katalogoptionen“. Es müssen alle drei Reiter abgearbeitet werden, bevor auf OK geklickt wird. Dies bitte unbedingt beachten!
 - *Reiter „Projektinfo“*: Hier werden grundlegende Informationen zum Projekt angegeben. Zum Beispiel: Projektname und -version, Sprache, Land, Zeichensatz, Übersetzerteams. In diesem Beispiel soll die Übersetzung in die englische Sprache (US) erfolgen (vgl. Abbildung 13).



Abbildung 13: Katalogoptionen - Reiter „Projektinfo“

- *Reiter „Pfade“*: Hier wird der relative Pfad, ausgehend vom Plugin-Sprachverzeichnis zum Plugin-Basisverzeichnis angegeben. Zur Erinnerung: in diesem Beispiel nannte sich das Sprachverzeichnis „lang“, dass direkt im Plugin-Basisverzeichnis liegt.
- Als Pfad muss hier also eingegeben werden: „..“ (2 Punkte für das Wechseln in eine Verzeichnisebene nach oben) (vgl. Abbildung 14).

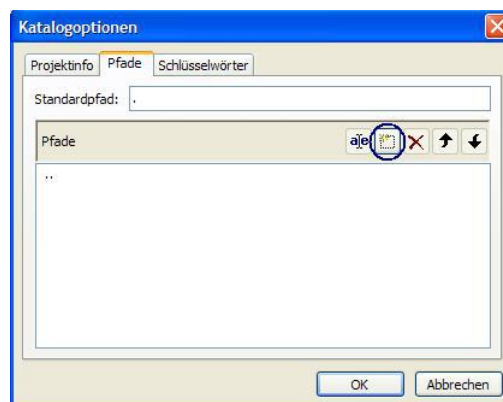


Abbildung 14: Katalogoptionen - Reiter „Pfade“

⁷²Vgl. BONDARI/GRIFFITHS (2011), Seite 245 - 249.

- *Reiter "Schlüsselwörter"*: Hier werden die Funktionsnamen der Liste hinzugefügt, an denen die zu übersetzenden Begriffe/Phrasen erkannt werden. Das wären hier in diesem Fall: `__` und `_e`. Alle bereits vorhandenen Funktionsnamen müssen aus der Liste entfernt werden (vgl. Abbildung 15).

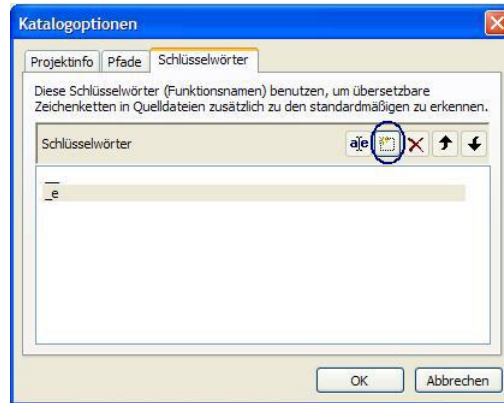


Abbildung 15: Katalogoptionen - Reiter „Schlüsselwörter“

- Nach Berücksichtigung aller Reiter, darf nun die Schaltfläche "OK" angeklickt werden. Es erscheint der Dialog „Speichern unter“. Die Datei sollte den Namen des Plugin-Ordners (hier: `mentoren-suche`) im `.pot`-Format – innerhalb des Sprachverzeichnisses – abgespeichert werden. Also: `mentoren-suche.pot` (Anmerkung: Die POT-Datei enthält alle Übersetzungen). Anschließend den Speichervorgang ausführen, indem die Schaltfläche „Speichern“ betätigt wird (vgl. Abbildung 16).

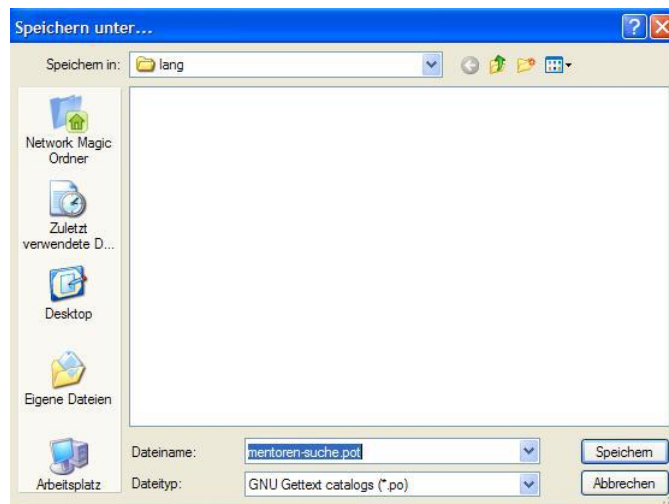


Abbildung 16: Katalogoptionen - Dialog "Speichern unter"

- Nun werden alle Zeichenketten aus dem Quellcode herausgefiltert. Dazu erscheint die in der Abbildung 17 dargestellte Meldung.

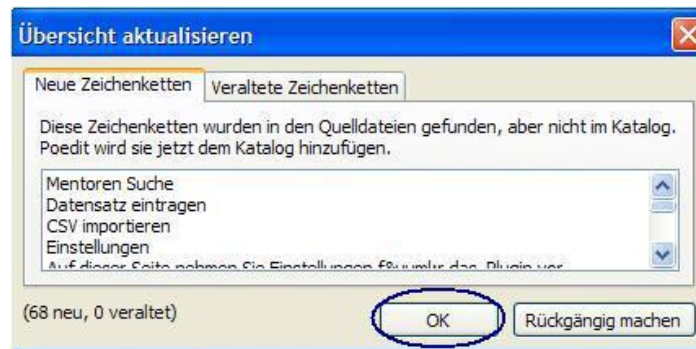


Abbildung 17: Katalogoptionen - Meldung "Übersicht aktualisieren"

- Nachdem Betätigen der Schaltfläche "OK", erscheinen alle zu übersetzenden Zeichenketten in der Oberfläche von "Poedit" (vgl. Abbildung 18). Nun erfolgt die Übersetzung wie folgt: es wird sukzessive jede Zeichenkette vom Anwender ("Übersetzer") angeklickt. Daraufhin erscheint im ersten unteren Textfeld die aktuelle, ausgewählte Originalzeichenkette. Im darunter liegenden Textfeld wird die entsprechende Übersetzung eingegeben. Danach wird auf das Icon "Katalog speichern" angeklickt. Nun ist die nächste Zeichenkette zu übersetzen. Die übersetzten Zeichenketten werden nach unten angehängt. Dies erfolgt gewöhnlich solange, bis alle Zeichenketten übersetzt sind.

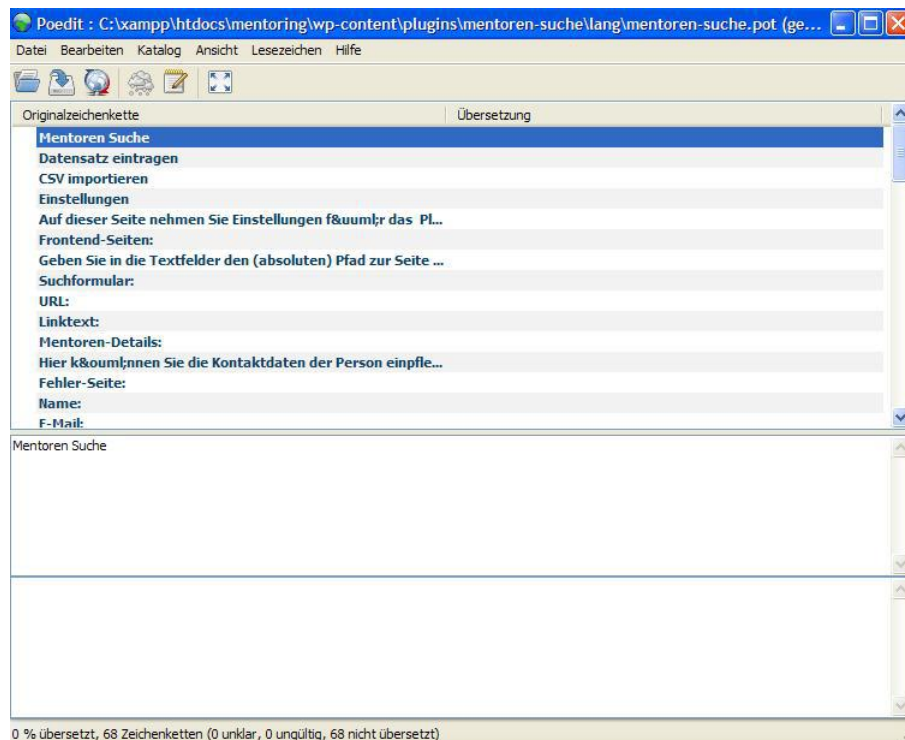


Abbildung 18: Poedit - Übersetzungen erstellen

- Sind alle Zeichenketten übersetzt, muss eine Kopie der POT-Datei als PO-Datei abgespeichert werden. Dies geschieht über "Datei" "Speichern unter". Der Dateiname

hat dabei folgende Konvention: *pluginname-sprachcode_Landcode.po*.

- a) Der *Sprachcode* entspricht dem Code nach ISO 639-1: siehe beispielsweise http://en.wikipedia.org/wiki/List_of_ISO_639-1_codes.
- b) Der *Ländercode* entspricht dem Code nach ISO 3166-1: siehe beispielsweise http://en.wikipedia.org/wiki/ISO_3166-1.

Für die in diesem Beispiel englische Übersetzung (Land: USA), entsteht folgender Dateiname: *mentoren-suche-en_US.po*.

Damit ist der Übersetzungsprozess abgeschlossen. Es ist darauf hinzuweisen, dass neben den .po- und .pot-Dateien auch eine .mo-Datei angelegt wird. Dabei handelt es sich um eine binäre Version der .po-Datei. Diese ist nicht menschenlesbar.⁷³

Der Prozess der Aktualisierung von bereits bestehenden Sprachdateien, soll hier nicht betrachtet werden, da dies den Rahmen dieses Tutorials sprengen würde. Denn dieser Vorgang wurde für die Entwicklung des Plugins "Mentoren-Suche" bisher nicht benötigt. Stattdessen wird auf Literatur Bondari/Griffiths⁷⁴ verwiesen.

Damit das Ergebnis der Übersetzung sichtbar ist, muss Wordpress auf die entsprechende Sprache eingestellt sein. Dies wird im nächsten Abschnitt 8.5 beschrieben.

8.5 Konfiguration der Wordpress-Sprache

Im Kapitel 8.1 wurde bereits erwähnt, dass die Sprachkonfiguration von Wordpress zentral in der Konfigurationsdatei wp-config.php geschieht. In diesem Abschnitt soll vorgestellt werden, wie diese Einstellung vorgenommen wird.

Die entscheidende Code-Zeile in der wp-config.php-Datei ist folgende (vgl. Listing 37).

```
1 <?php
2 ...
3 define( 'WPLANG' , 'de_DE' );
4 ...
5 ?>
```

Listing 37: wp-config.php – Die in PHP definierte Konstante WPLANG

In Listing 37 ist die definierte Konstante WPLANG für die zentrale Spracheinstellung der Wordpress-Oberfläche zu sehen. Der Wert der Konstante (hier: de_DE) entspricht demselben Schema (Sprachcode/Ländercode nach ISO), wie es im Abschnitt 8.4 beschrieben wird.

Dabei ist zu beachten, dass sowohl für die jeweiligen Plugins die entsprechenden Sprachdateien mitgeliefert werden müssen als auch im Wordpress-Verzeichnis wp-content/languages die Sprachdateien vorhanden sind. Ansonsten würde die Spracheinstellung keine oder nur unbefriedigende Auswirkungen haben.

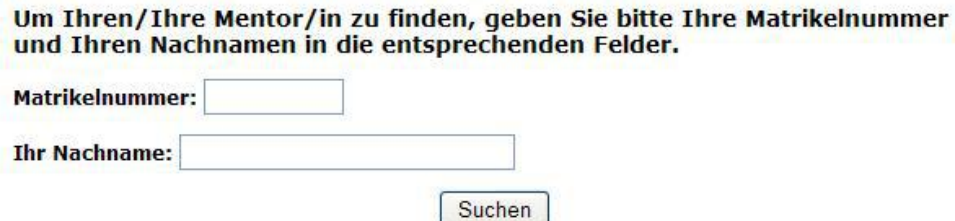
Zum Abschluss soll gezeigt werden, wie die Übersetzung anhand des Beispiel-Plugins "Mentoren-Suche" aussieht. Dazu muss in der wp-config.php (wie in diesem Abschnitt beschrieben) die Sprache englisch (für das Land USA) eingestellt werden. Der Wert für die Konstante lautet wie

⁷³Vgl. BONDARI/GRIFFITHS (2011), Seite 249.

⁷⁴Vgl. BONDARI/GRIFFITHS (2011), Seite 251.

folgt: en_US (kann von den Sprachdateien abgeschaut werden. Zum Beispiel: mentoren-suchen_US.po).

Das Ergebnis der Übersetzung wird im Folgenden für die Suchmaske im Frontend demonstriert (vgl. Abbildung 19 für die deutsche Spracheinstellung [Standard] und vgl. Abbildung 20 für die englische Spracheinstellung).

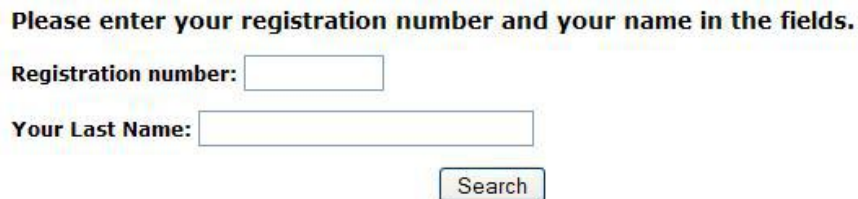


Um Ihren/Ihre Mentor/in zu finden, geben Sie bitte Ihre Matrikelnummer und Ihren Nachnamen in die entsprechenden Felder.

Matrikelnummer:

Ihr Nachname:

Abbildung 19: Deutsche Spracheinstellung (de_DE) [Standard]



Please enter your registration number and your name in the fields.

Registration number:

Your Last Name:

Abbildung 20: Englische Spracheinstellung (en_US)

9 Pluginverwendung in Wordpress

Nachdem die Programmierung des Plugins soweit abgeschlossen ist und die fundamentalen Kenntnisse erklärt und entsprechende Funktionen erläutert wurden, widmet sich dieses Kapitel mit der Installation und Deinstallation des Plugins.

Dabei sollte beachtet werden, dass auch hier aus Programmier- und nicht primär aus Anwendersicht erläutert und erklärt wird. Zuvor soll allerdings eine kleine Begriffsbestimmung für dieses Kapitel geschehen.

9.1 Unterscheidung der Installationstypen

Wie schon bereits angedeutet, soll an dieser Stelle zuerst einmal die unterschiedlichen Typen von Installation und Deinstallation eines Wordpress-Plugins angesprochen werden. Allgemein wird dabei in drei verschiedenen Zuständen unterschieden. Diese werden nun dargestellt, um anschließend mit den einzelnen Installations- und Deinstallationsroutinen fortzufahren.

Laut Williams/Richard/Tadlock⁷⁵ sind die drei Zustände folgendermaßen beschrieben:

1. Aktivierungsfunktionen

- Diese Art von Funktionen werden dann aufgerufen, wenn ein Benutzer mit entsprechenden Administrationsrechten dieses Plugin aktiviert.

2. Deaktivierungsfunktionen

- Auch diese Funktion verhält sich ähnlich wie die Aktivierungsfunktionen, mit dem kleinen Unterschied, dass diese erst dann ausgeführt werden, wenn der Administrator das zuvor aktivierte Plugin deaktiviert.

3. Deinstallationsfunktionen

- Nachdem ein Plugin deaktiviert wurde, kann es auch deinstalliert werden. Dabei werden alle Einstellungen sowie das Plugin selbst gelöscht. Eine solche Funktion sollte in jedem Plugin vorhanden sein, um dem User einen angenehmen Weg zur Deinstallation zu bieten und das Plugin professionell abzurunden.

Soweit an dieser Stelle zu den einzelnen Funktionsunterscheidungen. Das nächste Kapitel beschäftigt sich mit den einzelnen Routinen.

9.2 Installation

Ein jedes abgeschlossene Plugin sollte bei entsprechender Funktionalität und Entwicklungsstadium irgendwann installiert werden. Genau dies wird in diesem Unterkapitel angesprochen. Allgemein lässt sich formulieren, dass ein Plugin über die Administratoroberfläche unter dem Pfad *Plugins/Installieren* installiert werden kann.

Was dabei im Hintergrund passiert, das heißt genauer: Was aus Entwicklerperspektive passiert, ist im Unterkapitel 9.2.1 weiter beschrieben.

⁷⁵Vgl. WILLIAMS/RICHARD/TADLOCK (2011), Seite 20 - 22.

9.2.1 Installationsroutinen

Sogenannte Installationsroutinen werden bei der Aktivierung (vgl. Abschnitt 9.2.1.1), Deaktivierung (vgl. Abschnitt 9.3) und Löschen (vgl. Abschnitt 9.3.1) des Plugins angestoßen - also automatisch ausgeführt.

Der Vorteil bei solchen Installationsroutinen liegt darin, bestimmte Standardeinstellungen des Plugins zu setzen. Die hier angesprochenen Funktionen sind Fundamental und lassen sich für alle Plugins einsetzen.⁷⁶

Um die genaueren Eigenschaften einer solchen Aktivierungsfunktion kennen zu lernen, wird diese nun in Abschnitt 9.2.1.1 besprochen.

9.2.1.1 Aktivieren

Bei der Aktivierung werden einzelne Definitionen, Action, Options und Funktionen ausgeführt. Beispielsweise können somit die initialen Tabellen oder bestimmte Funktionen ausgeführt werden. An dieser Stelle sollen solche in dem Plugin verwendeten Objekte vorgestellt werden.

Plugin-Activation-Funktion

Diese Funktion dient dazu, direkt bei der Aktivierung des Benutzers ausgeführt zu werden. Hiermit lassen sich einige Standardeinstellungen festlegen. Natürlich lassen sich beispielsweise auch Versionsprüfungen oder verschiedenen Datenbanktabellen anlegen.

Dies soll an folgenden Listing 39 weiter erklärt werden:

```
1 <?php
2 ...
3 register_activation_hook( $file , $function );
4 ...
5 ?>
```

Listing 38: Beispiel für einen Activation-Hook

Wie dargestellt, sieht syntaktisch so eine Funktion aus, welche bei aktivieren des Plugins ausgeführt wird. Dabei sind die folgenden Parameter zu beachten⁷⁷:

1. **\$file**

- Hierbei handelt es sich um einen String-Parameter, welcher für den Pfad zu der Hauptdatei des Plugins bestimmt ist. Dies könnte also beispielsweise die Datei *mentoren_suche.php* sein.

2. **\$function**

- Diese hier angegebene Funktion wird dann aufgerufen, wenn die Aktivierung des Plugins stattfindet.

Ein Beispiel kann diese Definition veranschaulichen (siehe Listing 39)

⁷⁶Vgl. WILLIAMS/RICHARD/TADLOCK (2011), Seite 18.

⁷⁷Vgl. WILLIAMS/RICHARD/TADLOCK (2011), Seite 18.


```
1 <?php
2 ...
3 register_activation_hook( __FILE__, 'boj_myplugin_install' );
4 function INSTALL() { //do cool activation stuff
5 }
6 ...
7 ?>
```

Listing 39: Beispiel für einen Activation-Hook

Wenn also die Aktivierung des Plugins erfolgt, wird die Funktion `INSTALL()` aufgerufen. Allerdings soll es hier ein Anliegen sein, die PHP-Konstante `__FILE__` zu erläutern.

Wie bereits in der Definition erwähnt, soll der erste Parameter den Pfad zur Hauptdatei des Plugins darstellen. Mittels die `FILE`-Konstante von PHP, wird automatisch der absolute Pfad des Plugins zu der Datei ermittelt, welche die Funktion aufruft.⁷⁸

Nachdem die grundlegende Aktivierungsfunktion besprochen wurde, sollen nun auch ein paar Actions besprochen werden, welche für die Aktivierung des Plugins interessant sind.

Das Event `init`

Das erste Event, welches hier besprochen wird, ist das `init`-Event von Wordpress.

Wie in Listing dargestellt, wird mittels des Hooks *Action* (siehe Kapitel 3.5) und dem Event *Init* die Funktion `MS_MIN_WORDPRESS_VERSION` aufgerufen.

```
1 <?php
2 ...
3 define( 'MS_MIN_WORDPRESS_VERSION', '3.0' );
4 add_action( 'init', 'MS_MIN_WORDPRESS_VERSION' );
5 ...
6 ?>
```

Listing 40: Prüfung der Mindestversion in Wordpress

Dabei dient das Event *init* dazu, bestimmte Funktionen auszuführen. Dies passiert in aller Regel, nachdem Wordpress seine eigenen internen Funktionalitäten, wie das Digitalisieren von Widgets fertig gestellt hat. Diese können erst dann sicher erfolgen, wenn Wordpress von seiner Seite aus alle Einstellungen geladen und initialisiert hat.⁷⁹

So wird in dem oben genannten Beispiel die Funktion entsprechende Funktion aufgerufen. Dabei wird in der Zeile 3 definiert, dass der Parameter 3.0 mitgegeben werden soll. Hierbei handelt es sich um die Mindestversion, damit das Plugin überhaupt installiert werden darf, um auf bestimmte Funktionalitäten ab der Version 3.0 zuzugreifen. Dies ist Gegenstand des nächsten Abschnittes.

Prüfung der Mindestversion

Wie eine Kontrollfunktion zur Mindest-Wordpress-Version aussieht, wird in Listing 41 dargestellt.

⁷⁸Vgl. WILLIAMS/RICHARD/TADLOCK (2011), Seite 18.

⁷⁹Vgl. WILLIAMS/RICHARD/TADLOCK (2011), Seite 37.

```
1 <?php
2 ...
3 /**
4  * Prueft aktuelle Wordpress Version, ob Plugin installiert
5  * werden darf oder nicht.
6  */
7
8 function MS_MIN_WORDPRESS_VERSION() {
9     global $wp_version;
10
11     $exit_msg="'Mentoren_Suche'_benoetigt_WordPress'.
12     MS_MIN_WORDPRESS_VERSION.'_or_newer.'.
13     <a href='http://codex.wordpress.org/Upgrading_WordPress'>Wordpress_
14     Download</a>';
15
16     if (version_compare($wp_version,MS_MIN_WORDPRESS_VERSION,'<'))
17     {
18         exit ($exit_msg);
19     }
20     do_action('hk_mentees_db_create');
21     do_action('hk_mentor_db_create');
22 }
23 ...
24 ?>
25 %
```

Listing 41: Prüfen der Wordpressversion

Dabei wird zuerst in Zeile 9 eine entsprechende Variable verwendet *wp_version*, welches die Versionsnummer der Wordpress-Installation beinhaltet. Anschließend wird eine Fehlermeldung (Zeile 11 - 15) definiert, um schlussendlich zu überprüfen, ob die eingesetzte Wordpressversion über der Mindestversion (3.0) liegt. Dafür wird die Funktion *version_compare* verwendet. Es handelt sich dabei laut des offiziellen Handbuchs von PHP.net (<http://php.net/manual/en/function.version-compare.php>), um eine Funktion, welche 2 Nummern vergleichen kann. Dabei können beispielsweise die Operatoren *>*, *<*, *>=*, *<=* oder *!=* verwendet werden.

Falls dies nicht der Fall sein sollte, wird das Plugin nicht aktiviert und die definierte Fehlermeldung ausgegeben.

Bei erfolgreicher Überprüfung, werden hingegen die Initialisierungstabellen in die Datenbank geschrieben (siehe Zeile 21 - 22).

Das Event *admin_menu*

Eine weitere Funktion, welche beim Aktivieren des Plugins aufgerufen wird, ist das sogenannte Event *admin_menu*, welches beispielhaft in Listing 42 dargestellt ist.

```
1 <?php
2 ...
3 //Die Registrierung des Menues (Backend)
4 add_action('admin_menu','register_Mentees_menu');
```

```
5 ...
6 ?>
7 %
```

Listing 42: Menüerstellung im Adminbereich

Das Event *admin_menu* wird dazu benötigt, um im Administratorbereich von Wordpress Code auszuführen.⁸⁰ Hierbei wird die Funktion *register_Mentees_menu* aufgerufen, um ein Administratormenü für das Plugin zu erstellen.

Auch diese Action wird beim Aktivieren von Wordpress durch den Benutzer ausgeführt.

Das Event *dbDelta(\$sql)*

Als vorerst letztes zu beschreibendes Event bevor mit der Deinstallationsroutinen angefangen wird, soll noch das Event *dbDelta(\$sql)*.

Dieses ist dafür da, zu prüfen, ob eine Tabelle überhaupt vorhanden ist, bevor diese aktualisiert oder erstellt wird. Dabei wird geprüft, ob die gleiche Syntax bei der zu aktualisierenden Tabelle vorliegt und ändert oder fügt die erforderlichen Tabelle hinzu.⁸¹

Wie dies in unserem Plugin aussehen könnte, ist in Listing 43 beispielhaft dargestellt.

```
1 <?php
2 ...
3 function mentees_db_create() {
4     global $wpdb;
5     global $mentees_db_version;
6
7     //SQL-Anweisung
8     $sql = "CREATE_TABLE".MENTEE_TABLE." (
9     ...
10    ); ";
11
12    dbDelta($sql);
13    ...
14    ?>
```

Listing 43: Beispiel dbDelta

In diesem Code-Beispiel wird eine SQL-Anweisung als Zeichenkette in einer Variablen gespeichert werden. Anschließend wird mit dem entsprechenden Event geprüft, ob die Tabelle bereits vorhanden ist oder modifiziert werden muss. Abschließend wird dann der Befehl ausgeführt.

Soviel erst einmal zu den Installationevents. Im nächsten Kapitel geht es nun um die Deinstallation.

9.3 Deinstallation

In diesem Kapitel werden nun Funktionen erläutert, welche bei der Deaktivierung eines Plugins eine wichtige Rolle spielen.

⁸⁰Vgl. WILLIAMS/RICHARD/TADLOCK (2011), Seite 38.

⁸¹Vgl. WILLIAMS/RICHARD/TADLOCK (2011), Seite 193.

Um ein Plugin zu deinstallieren, muss im Administratormenü auf den Oberpunkt *Plugins* geklickt werden, um so eine Übersicht über die aktiven und inaktiven Plugins zu bekommen.

An dieser Stelle soll aber von der Anwendersicht auf die Entwicklersicht gewechselt werden.

Dabei soll an dieser Stelle ein besonderer Blick auf den *deactivation_hook* gesetzt werden.

Laut Williams/Richard/Tadlock⁸² wird die Funktion *register_deactivation_hook* eingesetzt, um die Deaktivierung des Plugins ausgeführt zu werden. Dabei hat diese zwei Parameter, welche in Listing 44 dargestellt sind.

```
1 <?php
2 ...
3 <?php register_deactivation_hook( $file , $function );
4 ...
5 ?>
```

Listing 44: Beispiel Syntax Deactivation-Hook

Anhand diesen kleinen Beispiels lassen sich die beiden Parameter erkennen. Diese sind folgendermaßen zu erklären:

1. **\$file**

- Hierbei handelt es sich um den gleichen String-Parameter wie bei der Aktivierungsfunktion: Dieser ist für den Pfad zu der Hauptdatei des Plugins bestimmt. Dies könnte also auch beispielsweise die Datei *mentoren_suche.php* sein.

2. **\$function**

- Diese hier angegebene Funktion wird dann aufgerufen, wenn die Deaktivierung des Plugins stattfindet.

Soviel erst einmal zu der Theorie. In dem Mentorenplugin ist folgendes Beispiel vorgekommen und soll an dieser Stelle erwähnt und erläutert werden:

```
1 <?php
2 ...
3 register_deactivation_hook( __FILE__, 'pluginUninstall' )
4 ...
5 ?>
```

Listing 45: Deactivation-Hook aus Mentorensuche

Wie bereits erwähnt, wird in Listing 45 die Funktion *pluginUninstall* aufgerufen, wenn das Plugin deaktiviert ist. Genau diese Funktion soll im folgenden Listing angeschaut werden: werden:

```
1 <?php
2 ...
3 function pluginUninstall() {
4     global $wpdb;
5
6     //Angelegte Optionen loeschen:
7     delete_option( 'mentees_db_version' );
```

⁸²Vgl. WILLIAMS/RICHARD/TADLOCK (2011), Seite 20.

```
8      ...
9
10     //Tabellen loeschen:
11     $sql = "DROP TABLE ".MENTOR_TABLE;
12     $wpdb->query( $sql );
13
14     $sql = "DROP TABLE ".MENTEE_TABLE;
15     $wpdb->query( $sql );
16 }
17 ...
18 ?>
```

Listing 46: Funktion Plugin Uninstall

In dem Listing 46 wird veranschaulicht, wie eine Uninstall-Funktion für ein Plugin aussehen könnte. Nachdem auf ein globales Objekt zur Interaktion mit der Wordpress-API zugegriffen wurde, werden nacheinander die einzelnen Options gelöscht. Anschließend werden dann beide Tabellen gelöscht. Dies ist eine Besonderheit bei dem Mentoren-Plugin: Normalerweise wird bei der Deaktivierung nur das Plugin inaktiv geschaltet, allerdings werden keine Tabellen gelöscht.

Allein aus Datenschutzrechtlichen Gründen haben wir uns entschieden, statt der Deinstallationsroutine, direkt das Löschen der Datenbank in die Deaktivierungsroutine einzubauen. Dies kann allerdings optional natürlich auch im zweiten Schritt (siehe Kapitel 9.3.1) passieren.

9.3.1 Deinstallieren

Das Deinstallieren eines Plugins löscht alle vorhandenen Daten des Plugins. Dabei beherrscht Wordpress zwei verschiedene Arten. Diese werden nun vorgestellt und mit Beispielen erläutert.

9.3.1.1 Uninstall.php - Methode

Die erste Methode, um ein Plugin zu löschen ist mittels der *Uninstall.php*. Der Vorteil einer solchen Datei besteht darin, dass alle Deinstallationsfunktionen in einer separaten Datei liegen und so keine Probleme mit anderen Funktionen entstehen. Dabei ist es wichtig, dass diese Datei im untersten Verzeichnis des Plugins angelegt wird. An dieser Stelle soll ein kleines Beispiel erfolgen:⁸³

```
1 <?php
2 ...
3
4 // If uninstall not called from WordPress exit
5 if( !defined( 'WP_UNINSTALL_PLUGIN' ) )
6     exit ();
7 // Delete option from options table
8 delete_option( 'boj_myplugin_options' );
9 //remove any additional options and custom tables
10 ...
11 ?>
```

⁸³Vgl. WILLIAMS/RICHARD/TADLOCK (2011), Seite 21.

Listing 47: Beispiel Uninstall.php-Datei

Wie in Listing 47 dargestellt, könnte eine Uninstall.php-Datei aussehen.

Dabei ist die erste If-Abfrage dazu gedacht, zu überprüfen, dass WordPress und nicht irgend eine andere Funktion dieses Skript aufruft. Wenn dies der Fall sein sollte, wird direkt ausgestiegen. Falls allerdings es sich tatsächlich im Wordpress handelt, kann beispielsweise mit einer Delete oder Drop-Anweisung Optionen und Tabellen gelöscht werden.

Ein großer Vorteil dieser Funktion besteht darin, dass die Dateien und Verzeichnisse des Plugins automatisch bei Aufruf des Skriptes von Wordpress mitgelöscht werden.

9.3.1.2 Uninstall - Hook

Nachdem die Version mittels einer Uninstall.php-Datei erläutert wurde, soll zur Abrundung auch der Uninstall-Hook von Wordpress besprochen werden.

Laut Williams/Richard/Tadlock⁸⁴ hat Wordpress eine Reihenfolge, wie verfahren wird, wenn ein Administrator ein Plugin deinstalliert. Im ersten Schritt wird geschaut, ob eine Uninstall.php-Datei vorhanden ist. Ist dies nicht der Fall, wird der Uninstall-Hook aufgerufen (natürlich nur in soweit er auch im Plugin vorhanden ist). Die Syntax dieses Hooks unterscheidet sich prinzipiell nicht sehr vom Aktivations-Hook:

```
1 <?php
2 ...
3     register_uninstall_hook( $file , $function );
4 ...
5 ?>
```

Listing 48: Syntax Uninstall-Hook

Wie in Listing 48 dargestellt, besteht dieser Hook auch aus 2 Parametern. Dabei ist der erste Parameter der Pfad zur Haupt-Plugindatei, während der zweite Parameter die einzuleitende Funktion beinhaltet. Beispielsweise lässt sich das folgende Listing 49 anführen:

```
1 <?php
2 ...
3     register_activation_hook( __FILE__, 'boj_myplugin_activate' );
4
5     function boj_myplugin_activate() {
6         //register the uninstall function
7         register_uninstall_hook( __FILE__, 'boj_myplugin_uninstaller' );
8     }
9
10    function boj_myplugin_uninstaller() {
11        //delete any options, tables, etc the plugin created
12        delete_option( 'boj_myplugin_options' );
13    }
14 ...
15 ?>
```

Listing 49: Beispiel Uninstall-Hook

⁸⁴Vgl. WILLIAMS/RICHARD/TADLOCK (2011), Seite 21 - 22.

In Listing 49 wird bei der Aktivierung des Plugins automatisch die Aktivierungsfunktion *boj_myplugin_activate* aufgerufen. Diese wiederum ruft den Uninstall-Hook auf, welche dann die Funktion *boj_myplugin_uninstaller()* aufruft und Funktionen zum löschen des Plugins bereitstellt. Mit diesem Beispiel soll aufgezeigt werden, dass bei den Hooks schnell Fehler während der Programmierung entstehen können. In diesem Fall würde das Plugin nicht installierbar sein, weil es sich direkt wieder deinstalliert. Rein logisch betrachtet, weiß der Entwickler nun, auf was er achten sollte. Allerdings gibt es einen einfacheren Weg:

Insgesamt empfehlen Williams/Richard/Tadlock⁸⁵, die *Uninstall.php*-Methode zu verwenden. Diese Methode ist einfacher zu programmieren, vermeidet grobe Fehler bei der Benutzung des Hooks und ist vom Programmierstandard eine saubere Variante, da die einzelnen Funktionalitäten modular programmiert sind.

Weitere Informationen zu diesem Thema finden sich unter:

1. http://codex.wordpress.org/Function_Reference/register_deactivation_hook
2. http://codex.wordpress.org/Function_Reference/register_activation_hook

⁸⁵Vgl. WILLIAMS/RICHARD/TADLOCK (2011), Seite 22.

10 Zusammenfassung und Ausblick

Abschließend lässt sich formulieren, dass diese Dokumentation als Einleitung zur Entwicklung für Wordpress-Plugins darstellt.

Allerdings sind weitere Literatur und Internetquellen von notwendig, um in bestimmten Themengebieten tiefer einzusteigen.

Dafür sind im Quellverzeichnis entsprechende Literatur und Internetquellen angegeben, welche aus der Sicht der Autoren sind als zweckmäßig erwiesen hat und hier entsprechend verwendet wurde. Trotz der Themenbreite wurde versucht, jedes Thema zu erläutern. Dies sollte gewährleisten, dass Grundkenntnisse für die Entwicklung vorhanden sind.

Das Ziel des Tutorials war es, einen Überblick über einzelne Themengebiete der Wordpress-Plugin-Entwicklung aufzuzeigen. Dabei wurde sich an einen Ablauf der Entwicklung gehalten, welcher auch in der Praxis weit verbreitet ist. Genauer gesagt, wurde von der Einleitung, allgemeinen Grundlagen zur Entwicklung von Wordpress-Plugins über die Menüerstellung und Shortcodes schlussendlich zu Datenbankzugriffen, Formularen, der Internationalisierung und Lokalisierung bis zu Installations- und Deinstallationsroutinen inhaltlich beschrieben, wie Plugins entwickelt werden. Es sollte weiterhin darauf geachtet werden, dass zwar ab dem Kapitel 4 (Menüerstellung) die Grundlagen abgeschlossen ist und theoretisch in beliebiger Reihenfolge weiter verfolgt werden kann. Dies ist zwar möglich, ist aber aus Sicht der Autoren nicht zu empfehlen.

Aus diesem Dokument lassen sich verschiedene weitere Dokumente erzeugen lassen, da es sich um ein Grundagentutorial handelt. Dabei können sich auf einzelne Kapitel bezogen werden und eventuell zu bestimmten Problematiken Lösungsvorschläge- oder alternativen formuliert werden. Insgesamt lässt sich formulieren, dass zwar die Wordpressentwicklung nicht so komplex ist, wie es auf den ersten Blick scheint. Trotzdem sind fundamentale Programmierkenntnisse notwendig - gerade was dir Programmierung in HTML und PHP angeht. Andernfalls entstehen Problematiken, die nicht direkt mit der Wordpress-Entwicklung zu tun haben und entsprechend mehr Zeit in der Entwicklung kosten.

Bei der aktiven Entwicklung ist es wichtig darauf zu achten, dass Wordpress regelmäßig neue Versionen ihrer Software veröffentlicht. So ist für eine Pluginentwicklung nicht nur entscheidend, für welche aktuelle Version das Plugin geschrieben wird, sondern auch, ob das Plugin für nächste Versionen bereit sein soll. Dazu sollte auf möglicherweise neue Funktionen oder auf jene, welche nicht benutzt werden, geachtet werden. Das würde auch mögliche Probleme bei der Weiterentwicklung einschränken.

Auch lässt sich feststellen, dass es gewisse Normen und Best-Practice-Ansätze in der Pluginentwicklung von den verschiedenen Autoren gibt. Diese wurden in diesem Umfeld nicht komplett 1:1 umgesetzt. Dies ist damit zu erklären, dass dies das erste Plugin für Wordpress aus Sicht der Autoren war und in Zukunft anders gehandhabt wird.

Gerade bei größerer Softwareentwicklungsprozessen sind auf gewisse Standards zu achten - gerade auch in dem Hinblick dieses aus `wordpress.com` hochzuladen und zu veröffentlichen.

Dies wäre beispielsweise ein Kapitel, welches hier nicht angesprochen wurde: Die Veröffentlichung über `wordpress.org/plugins` lässt sich daher auf der offiziellen Pluginseite und den Literaturangaben zum Selbststudium nachlesen und das Plugin entsprechend anpassen. Gerade

in den angegebenen Büchern finden sich schon zu Anfang entsprechende Kapitel und Abschnitte, welche zusätzlich zu dem hier beschriebenen Einstieg hervorragend geeignet sind. Auch für speziellere Themen sind diese zu empfehlen.

Während der Entwicklung und Verfassung des Tutorials, hatten die Autoren zwar viel Spaß bedingt durch dieses interessante Thema, es gab allerdings auch Hindernisse. Diese wurden durch intensive Recherche und Ausprobieren gelöst. Beispielsweise durch Aufsetzung einer zweiten Wordpress-Test-Umgebung, um die Hauptentwicklung nicht zu gefährden.

Dabei ist ein wichtiger Punkt, dass eine Versionskontrollsoftware eingesetzt wird, um die Konsistenz des Quellcodes zu gewährleisten. Dies ist auch von Vorteil, um eine Übersicht über verschiedene Projektphasen zu bekommen.

Die Autoren wünschen eine interessante Pluginentwicklung und sind für Kritik und Verbesserungsvorschläge immer offen.

Timo Amling (Autor): timo.amling@fh-koeln.de,

Anatol Tissen (Entwicklung): anatol.tissen@fh-koeln.de und

Ludger Schönfeld (Entwicklung und Co-Autor): ludger.schoenfeld@fh-koeln.de

Quellenverzeichnis

Das Literaturverzeichnis ist in zwei Abschnitte, nämlich den Buch- und Internetquellen. Dies soll so der besseren Überschaubarkeit des Lesers dienen.

Gedruckte Quellen

Bondari, Brian/Griffiths, Everett: Wordpress 3 Plugin - Development Essentials, Birmingham: Packt Publishing Ltd., 2011, 978-1-849513-52-4

Hetzel, Alexander: Wordpress 3 - Das umfassende Handbuch, Bonn: Galileo Press, 2012, 978-3-8362-1727-9

Williams, Brad/Richard, Ozh/Tadlock, Justin: Professional WordPress Plugin Development, Indianapolis: Wiley Publishing, Inc., 2011, 978-0-470-91622-3

Internetquellen

Gardner, Martin: How to Get Your WordPress Plugin To DROP TABLE From The Database, <http://www.martin-gardner.co.uk/how-to-get-your-wordpress-plugin-to-drop-table-from-the-database/>, abgerufen am 02.12.12 – Internetartikel

Group, The PHP: dirname, <http://php.net/manual/de/function.dirname.php>, abgerufen am 09.02.2013a – Internetartikel

Group, The PHP: Magische Konstanten, <http://php.net/manual/de/language.constants.predefined.php>, abgerufen am 09.02.2013b – Internetartikel

Group, The PHP: isset, <http://php.net/manual/de/function.isset.php>, abgerufen am 15.02.13 – Internetartikel

Group, The PHP: gettext, <http://www.php.net/manual/de/function.gettext.php>, abgerufen am 28.08.12 – Internetartikel

Mullenweg, Matt: Function Reference/plugin basename, http://codex.wordpress.org/Function_Reference/plugin_basename, abgerufen am 09.02.13 – Internetartikel

Mullenweg, Matt: Function Reference/do action, http://codex.wordpress.org/Function_Reference/do_action, abgerufen am 12.02.13 – Internetartikel

Mullenweg, Matt: The Shortcode API, http://codex.wordpress.org/Shortcode_API, abgerufen am 12.08.2012 – Internetartikel

Mullenweg, Matt: About Wordpress, <http://wordpress.org/about/>, abgerufen am 13.08.2012 – Website

Mullenweg, Matt: Function Reference/add menu page, http://codex.wordpress.org/Function_Reference/add_menu_page, abgerufen am 16.02.13a – Internetartikel

- Mullenweg, Matt:** Function Reference/add submenu page, http://codex.wordpress.org/Function_Reference/add_submenu_page, abgerufen am 16.02.13b – Internetartikel
- Mullenweg, Matt:** Protect Queries Against SQL Injection Attacks, http://codex.wordpress.org/Class_Reference/wpdb, abgerufen am 16.08.2012 – Internetartikel
- Mullenweg, Matt:** Function Reference/add option page, http://codex.wordpress.org/Function_Reference/add_options_page, abgerufen am 17.02.13 – Internetartikel
- Mullenweg, Matt:** Plugin API, http://codex.wordpress.org/Plugin_API, abgerufen am 22.10.12 – Internetartikel
- Mullenweg, Matt:** Function Reference/add option, http://codex.wordpress.org/Function_Reference/add_option, abgerufen am 23.10.12a – Internetartikel
- Mullenweg, Matt:** Plugin API/Action Reference/admin menu, http://codex.wordpress.org/Plugin_API/Action_Reference/admin_menu, abgerufen am 23.10.12b – Internetartikel
- SELFPHP.info:** Formulare und PHP, <http://www.selfphp.info/praxisbuch/praxisbuch.php?group=32>, abgerufen am 22.02.13 – Internetartikel
- Shirah, Joe Sam:** Java internationalization basics, <http://www.ibm.com/developerworks/java/tutorials/j-118n/j-118n-pdf.pdf>, abgerufen am 03.08.2012 – Technischer Bericht
- Vincent, Justin:** ezsql, <http://justinvincent.com/ezsql>, abgerufen am 22.08.12
- Wikipedia.de:** SQL-Injection, <http://de.wikipedia.org/wiki/SQL-Injection>, abgerufen am 01.12.12 – Internetartikel
- Wikipedia.de:** Wordpress, <http://de.wikipedia.org/wiki/Wordpress>, abgerufen am 16.08.2012 – Internetartikel