universität wien

AUSZUG DER DIPLOMARBEIT / EXCERPT OF THE DIPLOMA THESIS

Titel der Diplomarbeit / Title of the Diploma Thesis

„Kekse ohne Salz schmecken nicht“
Ein innovatives Unterrichtskonzept zur Vermittlung von Security im Webdatenbereich

verfasst von / submitted by

Simon Marik

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Magister der Naturwissenschaften (Mag.rer.nat.)

Wien, 2017 / Vienna, 2017

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 190 884 313
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Lehramtsstudium UniStG UF Informatik und Informatikmanagement UniStG - 10/2003 UF Geschichte, Sozialkunde und Polit. Bildg. UniStG - 10/2008
Betreut von / Supervisor:	ao. Univ.-Prof. Dipl.-Ing. Dr. Renate Motschnig
Mitbetreut von / Co-Supervisor:	Ass.-Prof. Mag. Dr. Christian Cenker

Standards in der Webentwicklung

0.1 PHP

Obwohl PHP erst seit einigen Jahren existiert, war man in der Webentwicklung bereits vor seiner Erscheinung sehr aktiv. Anstatt der neuartigen Skriptsprache wurden damals noch serverseitige Skripte mittels C oder Perl geschrieben, welche im weiteren Verlauf für Webanwendungen angepasst wurden. Die erste aller PHP-Versionen wurde im Juni 1995 von Rasmus Lerdorf entwickelt, unter dem Vorwand, häufig vorkommende Webprogrammierungsaufgaben einfacher und weniger repetitiv zu gestalten. Interessanterweise stand der Name damals noch nicht für das rekursive Akronym, unter welchem wir es heute kennen *(Abb. 2.3.1)*, sondern für „Personal Home Page“. Durch das vorgegebene Ziel, Codieraufwand weit möglichst zu reduzieren, führte dies zu einer starken HTML-Orientierung, in der PHP-Code schlichtweg in HTML eingebettet wurde und nach wie vor wird. Erst die zweite Version namens „PHP/FI 2.0“, sowie die dritte Version „PHP 3“, konnten einen Wandel verzeichnen. Letztere wurde hauptsächlich von Zeev Suraski und Andi Gutmans entwickelt, welche PHP von Grund auf neu schrieben und damit die bekannten Parsing-Probleme der zweiten Version behoben. Ab dieser Version wurde es auch viel einfacher, die Sprache für Dritte zu erweitern, sowie eigene Module dafür zu schreiben, was den Kernfunktionsumfang erheblich erweiterte.[1]

Abbildung 0.1.1: Offizielles PHP Hypertext Preprocessor Logo von PHP.net

Zwar wuchs die Verbreitung von PHP exponentiell, doch anfänglich war es fast unmöglich die Skriptsprache, vor allem für große kommerzielle Webanwendungen, zu nutzen. Entwickler, welche Perl, ColdFusion oder gar andere Sprachen verwendeten, wollten nämlich nicht mehr über die Fähigkeiten von PHP lernen und Java-Entwicklern fehlte es zunehmend an Frameworks, Objektorientierung sowie statischer Typisierung. Heute existiert selbstverständlich keiner dieser beiden Standpunkte mehr, da PHP mittlerweile keine Hilfssprache mehr darstellt.[2] Der Nachfolger „PHP 4“ brachte dann schlussendlich, neben vielen weiteren Änderungen, den Wechsel zur sogenannten

[1] vgl. *Hudson* (2005), S.1 f.
[2] vgl. *Schlossnagle* (2006), S.23 f.

„Zend Engine“ und befriedigte damit sehr viele einstige Gegner. Zend war ein weiteres Vorhaben von Suraski und Gutmans, mit dem die beiden PHP im Unternehmenssektor verbreiten wollten. Dabei übernahm Zend das Herzstück von PHP und erweiterte es um Features wie z.B. Reference Counting, Webserver-Abstraktion und einer komplett überarbeiteten Ausführungsweise des PHP-Codes selbst. Komplettiert wurde PHP mit dem Sprung von „PHP 4“ auf „PHP 5“, obwohl dieser nicht mehr so signifikant war wie jener von der dritten auf die vierte Version. Dennoch ist es nennenswert, dass mit „PHP 5“ eine verbesserte Objektorientierung, die Aufnahme von Exceptions mit try/catch-Fehlerbehandlung, SimpleXML, sowie SQLite eingeführt wurde, wodurch vieles erleichtert und die moderne Webentwicklung nachhaltig umgemodelt wurde.[3]

Mittlerweile ermöglicht PHP WebentwicklerInnen die Erzeugung dynamischer Webseiten, mit denen auch Webapplikationen, wie z.B. E-Commerce-Systeme, Chats oder Foren erstellt werden können. Ganz im Gegensatz zu statischen Internetseiten kann sich hier der Inhalt aufgrund von unterschiedlichsten BenutzerInnenaktionen bzw. neuer Basisinformationen (z.B. aus der Datenbank) weitgehend verändern. Insbesondere die vereinfachte Auswertung von Formulardaten, mit denen es möglich ist, Daten an einen Webserver zu senden, wird endlich für eine verbesserte Zusammenarbeit zwischen unterschiedlichen DBMS, von PHP unterstützt.[4] Im Vergleich zu anderen Programmiersprachen bietet PHP sehr viele Vorteile, die unter anderem wie folgt lauten:

- dient zur Entwicklung von Internetanwendungen
- ermöglicht eine einfache Entwicklung von Programmen
- unterstützt verschiedene Plattformen
- lässt sich sehr einfach in den freien Apache-Webserver integrieren
- ist frei verfügbar und sehr flexibel
- ist leicht erlernbar und ideal für den Bildungseinsatz geeignet[5]

Einer der wohl wichtigsten Vorteile von PHP ist die Interpretation des Codes. PHP kompiliert nämlich jedes Mal vorab im Hintergrund den Code in eine Folge von Anweisungen, welche auch „Opcodes“ genannt werden. Diese Anweisungen werden anschließend der Reihe nach ausgeführt, bis das Skript beendet wird. Das unterscheidet PHP, genauso wie auch Java, von herkömmlichen kompilierten Sprachen wie C++, welche den Code in eine ausführbare Datei übersetzen und diese jedes Mal ausführen, wenn dieses Segment wieder benötigt wird. Die ständige Neukompilierung führt dabei zu enormen Performanceeinbußen, welche unter PHP allesamt wegfallen. Andererseits können PHP-Skripte länger zum Kompilieren als zum Ausführen brauchen, wodurch eine größere Zeitverzögerung entsteht. Diese kann aber ganz einfach mittels sogenannter serverseitiger „PHP-Code-Caches“ verhindert werden.[6]

Wie vorhin bereits erwähnt wird PHP-Code in ein bestehendes HTML-Dokument eingebettet. Dazu wird in den meisten PHP-Applikationen die Methode `<?php ... ?>` verwendet, welche im folgenden Minimalbeispiel *(Abb. 2.3.2)* nochmals vollständig einschließlich des HTML-Codes abgebildet ist.

[3] vgl. *Hudson* (2005), S.2.
[4] vgl. *Theis* (2014), S.18.
[5] vgl. *ebd.*, S.18 f.
[6] vgl. *Hudson* (2005), S.3.

```
<!DOCTYPE html>
   <html>
      <head>
         <title>Thesis Project</title>
         </head>
      <body>
         <p>Erste Zeile in HTML</p><br>
         <?php echo "Zweite Zeile in PHP <br>"; ?>
         <p>Dritte Zeile in HTML</p><br>
         <?php
            echo "Vierte Zeile in PHP <br>";
            echo "Fünfte Zeile in PHP <br>";
         ?>
      </body>
   </html>
```

Abbildung 0.1.2: Minimalbeispiel zur Verdeutlichung der Einbettung von PHP-Code in HTML

Das Element „<?php“ kann dabei, wie man sieht, eine einzelne Anweisung oder auch einen ganzen Block an Anweisungen in das HTML-Dokument einfließen lassen. Dies geschieht solange bis der Endtag „?>“ erreicht wird. Demnach können PHP-Blöcke im ganzen Dokument verstreut bzw. auch zwischen HTML-Elementen untergebracht werden, da der Code nur von oben nach unten abgearbeitet wird. Die Anweisung `echo` gibt wiederum den angegebenen Text auf dem Bildschirm aus. Dabei ist es wichtig darauf zu achten, dass der Text entweder in doppelten Anführungsstrichen bzw. in einfachen Hochkommata geschrieben wird. HTML-Elemente wie z.B. das Zeilenumbruchselement
 können einfach in den PHP-Code eingearbeitet werden, um dabei automatisch als valide HTML-Sprache erkannt zu werden.[7] Sogenannter „Whitespace“, also Leerzeichen innerhalb des Codes, haben selbstverständlich keinerlei Auswirkungen auf die Funktionalität, sondern dienen eher der Übersichtlichkeit.[8] Zu den wichtigsten Stammfunktionen von PHP zählen:

- Erstellung von PHP-Skripte innerhalb normaler Editoren wie Notepad oder vim
- Kennzeichnung von Variablen durch ein Dollarzeichen vor dem eigentlichen Variablennamen (z.B. `$var`)
- direkte Ersetzung der Daten von Variablen in String-Konstanten
- Zeichencodierung für jede 8-Bit-Sequenz – also jedes Zeichen – muss bekannt sein
- Bedingungen und Verzweigungen wie if/else-Anweisungen oder switch-Anweisungen, sowie Schleifensteuerung durch for- oder while-Schleifen
- Möglichkeit der Erstellung und Verwendung von verschiedenen Datentypen, Konstanten und Funktionen[9]

0.1.1 PHP 5.6

Als im Juli 2004 „PHP 5“ erschien konnte zu dieser Zeit noch keiner ahnen, dass die Entwicklungszeit bis zur Version 5.6 ganze weitere zehn Jahre dauern würde. Während Version 5.3 im Juni 2009 ihr Debut feierte, konnten die PHP-EntwicklerInnen erst gegen März 2012 den Nachfolger 5.4 und erst ein weiteres Jahr später Version 5.5 auf den Markt bringen. Von PHP 5.5 bis zur

[7] vgl. *Theis* (2014), S.21 f.
[8] vgl. *Williams & Lane* (2005), S.19.
[9] vgl. *ebd.*, S.20 ff.

heutigen „finalen“ Version 5.6 (ziemlich relativ, da zum heutigen Stand „PHP 7“ bereits angekündigt und teilweise sogar schon verbreitet wurde)[10] verging dann erneut ein Jahr und somit wurde PHP 5.6 erst im August 2014 präsentiert.[11]

Wie man sieht, musste man sehr lange auf eine neue Version von PHP warten, da sehr viele neue Features und Verbesserungen eingebaut sowie andere Komponenten entfernt bzw. als veraltet markiert wurden. Dabei sind aber auch einige Features geändert worden, welche stellenweise keine Abwärtskompatibilität mehr gewährleisten (z.B. die Methoden `json_decode()`, `mcrypt_encrypt()`, `mcrypt_decrypt()` oder GNU Multiple Precision Ressourcen), obwohl das meiste von alten PHP 5 Codes unter 5.6 noch verwendet werden kann. Zusammenfassend beinhaltet PHP 5.6 folgende neue Features:

- konstante Skalare Ausdrücke, welche Zahlen und Strings beinhalten können
- Variable Funktionen mittels neuem `[...]` Operator anstatt `func_get_args()`
- neuer Potenzierungsoperator `[**]`
- Methode `_debuginfo()` (kann Eigenschaften und Werte von Objekten ändern, nachdem sie mit `var_dump()` ausgegeben wurden)
- Default Character Encoding (zum Einstellen einer allgemeinen Zeichenkodierung)
- größerer Dateiupload als bisher möglich (größer als 2 Gigabyte)[12]
- Parameter entpacken mittels neuem `[...]` Operator[13]
- Spiegeln von Bildern durch neue Methode `imageflip()`[14]
- Namensräume (ermöglichen Elemente gleichen Namens zu definieren, zu nutzen und eindeutig zuzuordnen mittels `use`, `global` und `class`)[15]

Durchforstet man ein bisschen die offizielle PHP.net Website, findet man sogar ein kleines Logbuch über alle Funktionen, welche für die Version 5.6 geändert wurden. Prinzipiell könnte man meinen, dass diese auch einfach in der vorhin angeführten Aufzählungsliste Platz finden würden, doch die geänderten Kernfunktionen von PHP 5.6 sind so essentiell wichtig, dass sie sich sogar einen eigenen Absatz verdient haben. Unter diesen sogenannten Kernfunktionen wurde nämlich unter anderem auch die Methode `crypt()` verändert, welche im weiteren Sinne die Grundfunktion eines jeden Salted-Hash verschlüsselten Login-Systems darstellt. Ab sofort wird ein sogenannter `E_NOTICE` Error (d.h. Laufzeitfehler) ausgegeben, falls der `salt` Parameter innerhalb der Methode fehlen sollte.[16] Man sieht also, dass selbst in den Entwicklerstudios von PHP die Salted-Hash-Methode bereits Einzug genommen hat und höchstwahrscheinlich eher weiterentwickelt wird, anstatt jemals entfernt bzw. als veraltet markiert zu werden.

0.1.2 Externer Dateibezug

In PHP ist es möglich, einzelne Funktionen sowie komplette Skripte in externe Dateien auszulagern, damit sie an verschiedenen Stellen erneut verwendet werden können. Durch die verbesserte Objektorientierung seit „PHP 5“ erfreut sich diese Methodik unter Entwicklern mittlerweile großer Beliebtheit. Hierfür stellt PHP die Anweisungen `include()`, `include_once()`, `require()` und `require_once()` zur Verfügung, welche mehrere direkte sowie indirekte Angriffspunkte aufweisen. Selbstverständlich sind diese Schwachstellen keinesfalls PHP zuzuordnen, sondern eher einem sorglosen Umgang mit Dateien und der Programmiersprache selbst. Dabei treten häufig Fallen in Verbindung mit sogenannten „Include-Dateien“ auf, die wie folgt lauten:

[10] s. *PHP.net*: Migrating from PHP 5.6.x to PHP 7.0.x, (Stand: 28.09.2016).
[11] vgl. *Maurice* (2015), S.5.
[12] s. *Envatotuts Plus*: PHP 5.6 What's New, (Stand: 28.09.2016).
[13] vgl. *Theis* (2014), S.100.
[14] vgl. *ebd.*, S.505.
[15] vgl. *ebd.*, S.281 f.
[16] s. *PHP.net*: PHP 5.6 Changed Functions, (Stand: 28.09.2016).

- Dateiendungen der inkludierten Dateien werden nicht vom Webserver geparst
- Dateien liegen in einem öffentlich zugänglichen Verzeichnis
- Dateinamen, welche von außen als Parameter kommen, werden für `include()` und `require()` verwendet[17]

Aufgrund der hohen Flexibilität der Sprache spielt es lediglich eine Rolle, wie der Dateiname der zu inkludierenden Datei zum Zeitpunkt der Interpretation lautet. Ist diese Datei dann auch noch existent und eine für PHP valide Dateiform, wird diese ohne Nachfragen im Code eingebunden. Dieses Verhalten birgt aber eine wesentliche Gefahr in sich: Wird der Wert des Parameters durch externe Daten gespeist, können bei ungenügender Konfiguration des Webservers sensible Daten ausgelesen bzw. schadhafter Code am System ausgeführt werden. Dies wird ermöglicht, da PHP in der Lage ist, bereits entfernte Dateien in ein Skript einzubinden. Das bedeutet, dass nicht zwingend auf eine lokale Datei verwiesen werden muss, sondern Dateien auch per HTTP oder FTP nachgeladen werden können. Der Zugriff kann nur erfolgen, wenn die Option `allow_url_fopen` der PHP-Konfigurationsdatei (php.ini) auf 1 steht. Eine allgemeine Deaktivierung dieser Problematik ist nicht möglich.[18] Wie man sieht, gibt es immer einen Weg, Manipulation über einen Webserver zu betreiben – auch wenn es sich dabei nur um Include-Dateien handelt. Man kann dieser Problematik allerdings einen Riegel vorschieben, indem man eine Aufrufkombination aus `header()` und dem Auslesen bzw. dem Ausgeben einer solchen Datei verwendet. Mittels `header()` wird nämlich der sogenannte „Content-Type“ festgestellt, damit der Client stets weiß, um welche Datei es sich gerade handelt und wie er damit umzugehen hat.[19]

Des Weiteren ist es möglich, in einer einzigen PHP-Datei gleich mehrere Include-Dateien einzubinden und zwar ganz unabhängig davon, an welcher Stelle sich diese im Code befinden. Prinzipiell können die oben genannten Anweisungen als eine Art „Einfügen“-Befehl gesehen und verstanden werden, bei dem der Aufruf der Funktion durch den Inhalt des Skripts ersetzt wird. So können beispielsweise Variablen für Datenbankverbindungsparameter bzw. einfache Header- und Footer-Dateien sinnvoll ausgelagert werden, anstatt diese jedes Mal erneut aufzurufen und damit eine Mehrfacheinbindung zu riskieren *(Abb. 2.3.3)*. Diese Methodik dient vor allem zur Codeorganisation und ist bei größeren Webprojekten kaum auszuschlagen.[20]

Um die eben abgebildete Mehrfacheinbindung zu verhindern, zahlt es sich aus, die Befehle `include_once()` bzw. `require_once()` anstatt `include()` oder `require()` zu verwenden. Obwohl alle Befehle prinzipiell gleich funktionieren, werden die angeforderten Dateien durch den Zusatz `_once` nur einmal in die Ursprungsdatei geladen (auch bei mehrfacher Anforderung), sofern diese noch nicht in der Liste der bereits eingefügten Dateien stehen.[21] Wichtig dabei ist, dass hier ganz penibelst auf Groß- und Kleinschreibung geachtet wird und man demnach bei der Namensverteilung besonders gut aufpassen muss. Wird versucht eine nicht existierende Datei einzufügen, wird entweder eine Warnmeldung ausgegeben, bzw. unter `require()` oder `require_once()` ein Fatal Error, welcher wiederum das Skript abbricht. In älteren PHP-Versionen war das allerdings noch nicht so. Der Befehl `require()` stellte nämlich ein unbedingtes Äquivalent zu `include()` dar, wodurch PHP versuchte, die angegebene Datei zu laden, selbst wenn die Zeile, in der `require()` stand, nie ausgeführt wurde. Mittlerweile überprüft PHP zuerst das angegebene Verzeichnis, in welchem das Skript läuft. Wird die Engine dort nicht fündig, wird der Include-Pfad unter der Direktive `include_path` in der php.ini Datei überprüft.[22]

[17] vgl. *Wassermann* (2007), S.67.
[18] vgl. *ebd.*, S.76 f.
[19] vgl. *ebd.*, S.81.
[20] vgl. *Beighley & Morrison* (2009), S.255 f.
[21] s. *PHP.net*: Funktion require_once, (Stand: 29.09.2016).
[22] vgl. *Hudson* (2005), S.41 f.

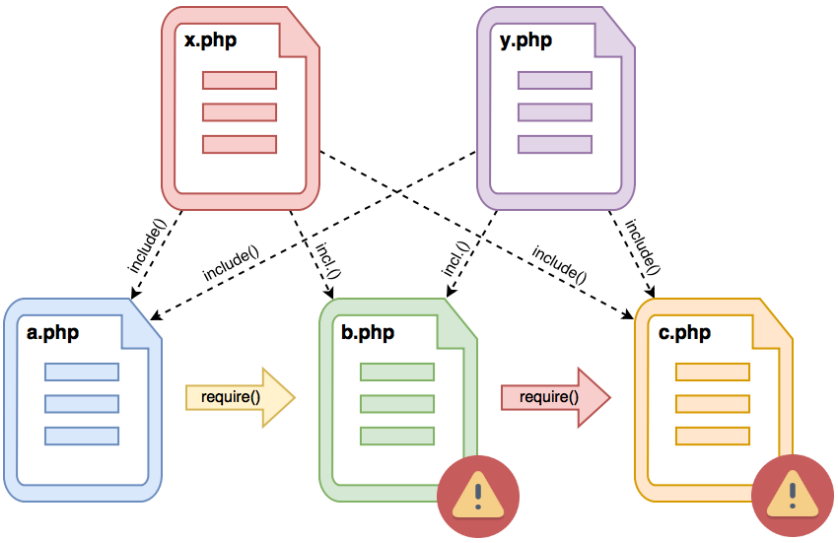

Abbildung 0.1.3: Mehrfacheinbindung von PHP-Skripte durch `inlcude` oder `require`

0.1.3 PHP-Sessions

Ein grundlegendes Erkennungsmerkmal des Internets ist die zustandslose Interaktion zwischen Browsern und Webservern, da das verwendete HTTP-Protokoll ebenfalls zustandslos ist. Dadurch ist kein HTTP-Request vom jeweils anderen abhängig, was für kleinere Internetanwendungen vollkommen ausreicht. Die zustandslose Natur ist allerdings bei Anwendungen, welche eine komplexere Interaktion mit der BenutzerIn erfordern wie z.B. bei einem Onlinewarenkorb, ein regelrechtes Problem, da man nicht mehrere zusammenhangslose sowie zustandslose Webseiten aneinanderhängen kann. Für solche zustandsbehafteten Webdatenbankanwendungen eignet sich aber der Einsatz von sogenannten „Sessions“. Eine Session verwaltet die entstehenden Interaktionen zwischen Webbrowsern und Webservern. Dadurch wird es möglich, bestimmte Informationen, welche durch Benutzeraktionen entstehen, festzustellen und zu verwenden. Wird eine Session gestartet, erhält die BrowsernutzerIn eine eindeutige Session-ID, welche bei zukünftigen Serveranfragen als eine Art Cookie mitverwendet wird. Führt der Browser dann demnach eine Anfrage aus, nutzt der Server die gegebene Session-ID, um etwaige Sessionvariablen zu finden, zu lesen oder zu beschreiben. Bei jedem HTTP-Request und jedem HTTP-Response wird also die Session-ID zwischen dem Webbrowser und dem Webserver transferiert, obwohl die Sessiondaten selbst lediglich am Server abgespeichert werden. Die Speicherung passiert nur serverseitig, da es von hier aus einfacher ist, ungewollte Änderungen der Zustandsinformation noch rechtzeitig zu verhindern.[23]

Zu den wichtigsten Elementen des PHP Session-Managements zählen unter anderem die Identifikation von Sessions, das Speichern von Sessionvariablen sowie das Bereinigen alter Sessions, welche allesamt vom PHP Session-Handler abgearbeitet werden. All diese Aufgaben werden bereits von der PHP-Bibliothek direkt für das allgemeine Session-Management abgedeckt. Die Standardkonfiguration des Session-Managements nutzt dabei vor allem lokal gespeicherte Dateien, um die Sessionvariablen abzuspeichern. Dateien als Sessionspeicher zu verwenden ist aber nur für jene Webanwendungen kein Problem, bei denen die Anzahl der gleichzeitigen Zugriffe beschränkt ist. Eine bessere und vor allem skalierbarere Lösung wäre der Einsatz von MySQL-Datenbanken als Speicherort für Sessioninformationen.[24] Der Session-Handler kümmert sich dabei um eine reibungslose Ablaufsteuerung aller Aktionen während einer einzigen

[23] vgl. *Williams & Lane* (2005), S.363 ff.
[24] vgl. *ebd.*, S.366.

Operation. Dabei initialisiert er seine Daten nach dem Start, führt eine Speicherplatzbereinigung durch und liest dabei die Sessiondaten der BenutzerIn aus. Die Seitenlogik wird dabei nach dem Befehl `session_start` ausgeführt, wodurch das Skript das Array `$_SESSION` nach Belieben modifizieren und verwenden kann. Wird die Session dann schlussendlich über die Funktion `session_destroy()` geschlossen, werden die Informationen wieder auf die Festplatte geschrieben und die internen Sessiondaten gelöscht *(Abb. 2.3.4)*.[25]

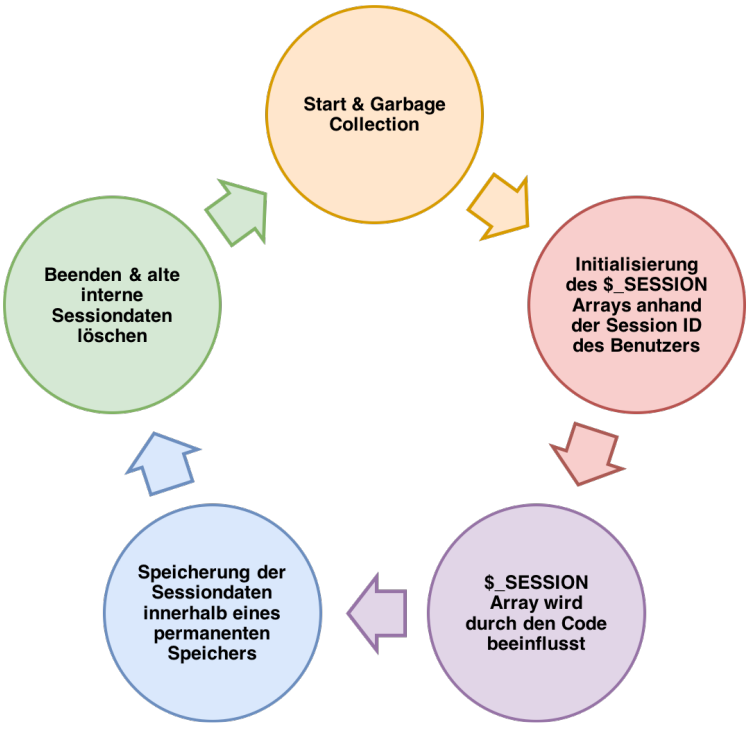

Abbildung 0.1.4: Ablaufplan eines PHP Session-Handlers

Prinzipiell wird im Session-Management zwischen serverseitigen und clientseitigen Sessions unterschieden. Serverseitige Sessions sind Verfahren, bei denen geringe Datenmengen zwischen Server und Client transferiert werden. Dabei wird lediglich die Session-ID übertragen und anschließend dessen Status in einer Art Sessioncache am Server abgelegt. Allerdings lassen sich serverseitige Sessiontechniken nur schwer an eine verteilte Architektur anpassen, wodurch clientseitige Sessions oftmals bevorzugt werden, obwohl sie eigentlich einen schlechten Ruf haben (bei jeder Anfrage werden nämlich alle Sessiondaten zwischen Client und Server ausgetauscht). EntwicklerInnen gehen oftmals viel zu kompliziert an eine Problemlösung heran, wodurch z.B. Applikationsserver oder gar intensive Session-Verwaltungstechniken zum Einsatz kommen. Sie sind aber dennoch sehr beliebt, da sie beispielsweise für eine große Anzahl von Clients sehr leicht skalierbar sind und sich einfach in verteilten Systemen einbetten lassen.[26]

Dadurch, dass Sessions temporäre Daten über ihre Besucher abspeichern, eignen sie sich besonders gut, wenn diese Daten außerhalb des Servers nicht zugänglich sein sollten. Dies wird durch die automatische Umschreibung von URLs ermöglicht, mit der PHP ganz einfach Session-IDs zum Client überträgt. Damit stellen sie eine sehr beliebte Alternative zu Cookies dar, falls die BenutzerIn die Verwendung von Cookies im Webbrowser deaktiviert haben sollte. Demnach stellt eine Session eine Kombination aus einer abgespeicherten serverseitigen Datei,

[25] vgl. *Schlossnagle* (2006), S.406 f.
[26] vgl. *ebd.*, S.395 f.

sowie einem clientseitigen Cookie dar, in dem eine Referenz zu den Serverdaten abgespeichert wird. Durch die Funktion `session_start()` im PHP-Code wird sowohl die Serverdatei, als auch das lokale Cookie erzeugt, wobei keinerlei Parameter der Funktion übergeben werden. Diese informiert original den Server, dass ab sofort Sessions zum Einsatz kommen und überprüft, ob die BenutzerIn ein Session-Cookie gesendet hat. Falls dies nicht zutrifft, wird eine neue Sessiondatei am Server erzeugt und eine neue ID an die BenutzerIn gesendet. Anderenfalls sendet PHP einfach die Sessiondaten weiter. Erst die Funktion `session_start()` ermöglicht den Zugriff auf nachfolgende spezifische Sessionvariablen der BenutzerInnen, da diese eindeutig benannt irgendwo am Server weggesperrt liegen und zuerst zugänglich gemacht werden müssen. Das ist auch der Grund, weshalb `session_start()` immer vor dem `<body>`-Tag in einem HTML-Dokument stehen muss.[27]

0.1.3.1 Sessionvariablen

Eine Session besteht, wie vorhin bereits erwähnt, aus zwei verschiedenen Komponenten: Einer Session-ID sowie einer selbst erzeugten Sessionvariablen. Sogenannte Sessionvariablen können Zustandsinformationen enthalten, welche auf Interaktionen der NutzerIn bzw. der verwendeten Anwendung beruhen. Als prädestiniertes Beispiel wäre an dieser Stelle wieder der Online-Warenkorb zu nennen. So können hier beispielsweise Sessionvariablen die verschiedenen Artikel, die Stückzahlen, die Preise sowie andere Artikel speichern, welche sich die BenutzerIn im Zuge seines Einkaufs sonst noch angesehen hat. Dafür werden die Sessionvariablen am Webserver bzw. in der DB gespeichert, wo sie mit Hilfe der Session-ID wieder abgerufen werden können. In der Praxis werden Sessionvariablen normalerweise vom Webserver in einer Datei bzw. vom Datenbankserver in einer Tabelle gespeichert. Allerdings wird es erst durch die vergebene Session-ID möglich, die aktuelle Benutzersession von anderen Sessiondaten zu unterscheiden.

[27] vgl. *Hudson* (2005), S.184.

Die folgende Abbildung *(Abb. 2.3.5)* zeigt dabei auf, wie Sessionvariablen in der Webserverumgebung gespeichert und indiziert werden, sodass keine Verwechslung oder ähnliches dabei auftreten kann.[28]

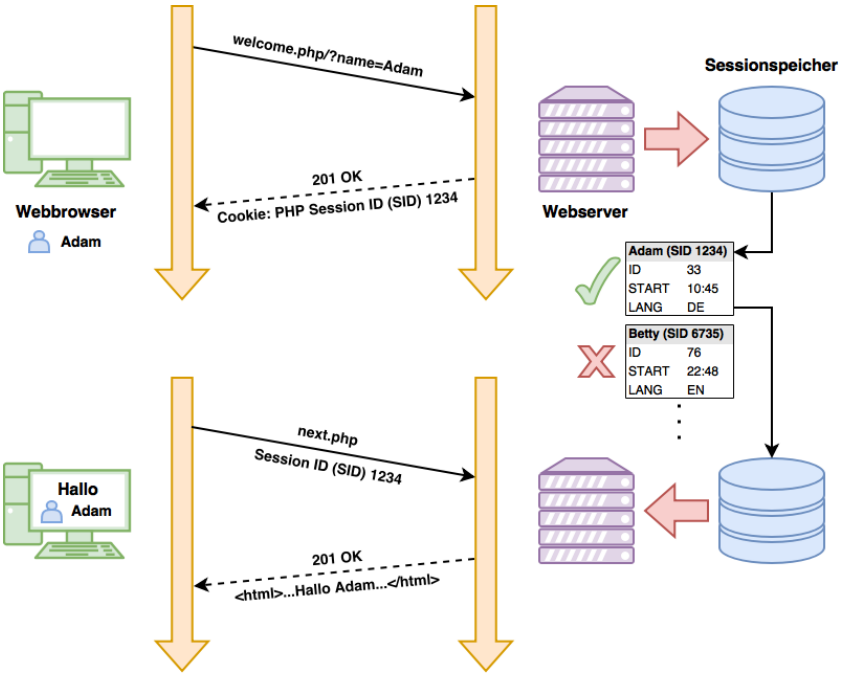

Abbildung 0.1.5: Speicherung & Indizierung von Sessionvariablen anhand der Session-ID

Prinzipiell werden alle Sessiondaten innerhalb des Superglobal-Arrays `$_SESSION` gespeichert. Das bedeutet also, dass jede Sessionvariable zusammen mit ihrem Wert, ein Element dieses globalen Arrays darstellt. NutzerInnen können diesem Array, genauso wie bei anderen, einfach Variablen hinzufügen, welche selbst nach dem erneuten Laden einer Seite noch vorhanden sind.[29] Um eine Sessionvariable anschließend wieder zu entfernen, verwendet man die Funktion `unset()` bzw. löscht das gesamte Array mittels des Befehls `array()` oder den gesamten Sessioninhalt via `session_unset()`. Sessionvariablen können dabei jeden Datentyp annehmen, welche auch von PHP selbst unterstützt werden. Allerdings ist es ratsam, sofern man Objekte als Sessionvariablen speichern möchte, die Klassendefinition dieser Objekte in jedem Skript einzuschließen, welches über `session_start()` aufgerufen wird. Dabei ist es vollkommen gleichgültig, ob das aufgerufene Skript diese Objekte auch tatsächlich verwendet oder nicht. Für PHP ist es nur wichtig, die Objekte korrekt aus dem Sessionspeicher zu lesen und später wieder richtig hineinzuschreiben.[30] Um diese Variablenoperationen besser zu verstehen, sind in der folgenden Abbildung *(Abb. 2.3.6)* nochmals alle in Codeform dargestellt.

[28] vgl. *Williams & Lane* (2005), S.364.
[29] vgl. *Hudson* (2005), S.184.
[30] vgl. *Williams & Lane* (2005), S.369.

```php
/* Sessionvariablen mit verschiedenen Datentypen hinzufügen */
$_SESSION['thesis-session'] = true;
$_SESSION['user'] = $id;
$_SESSION['start'] = time();
$_SESSION['expire'] = $_SESSION['start'] + (30 * 60);

/* Sessionvariablen entfernen */
$_SESSION['thesis-session'] = false;
unset($_SESSION['user']);
session_unset();

/* Sessionvariablen mit Objekten und Klassen hinzufügen */
$_SESSION['ein-objekt'] = new eine_klasse();
```

Abbildung 0.1.6: Minimalbeispiel zu den einzelnen Variablenoperationen innerhalb von Sessions

Möchte man Sessonvariablen auf dem Webserver verwalten, müssen diese für jede einzelne Browsersession gespeichert werden. Dabei stellt sich die Frage, wie lange man solche Daten am besten speichern sollte. Dadurch, dass HTTP zustandslos ist, gibt es keine Möglichkeit festzustellen, wann eine BenutzerIn eine Session nicht mehr benötigt. Im Idealfall loggt sich die BenutzerIn über den Aufruf eines Logout-Skripts, welches die Session zerstört, aus. Der Server kann sich aber dennoch nie sicher sein, ob die BenutzerIn immer noch da ist, wodurch er gezwungen ist, alte Sessions zu löschen, welche schon länger nicht mehr benutzt wurden. Diese stellen nämlich ein ziemliches Sicherheitsrisiko dar und verbrauchen noch nebenbei unnötige Serverressourcen. Zusammenfassend lassen sich in diesem Konsens drei unterschiedliche Erkennungsmerkmale feststellen, welche im weitesten Sinne Sessions innerhalb von Webanwendungen ausmachen:

- Informationen und Zustände, welche über mehrere HTTP-Requests erhalten bleiben sollen, müssen immer in Sessionvariablen gespeichert werden
- Jeder HTTP-Request muss eine eindeutige Session-ID mit sich führen
- Sessions müssen Verfallszeiten in Form von Sessionvariablen besitzen[31]

0.1.3.1.1 Einsatzmöglichkeiten

Sessionvariablen müssen nicht immer direkt etwas mit benutzerspezifischen Daten sowie aktionsbedingten Informationen zu tun haben. Dadurch, dass sich nämlich alle Datentypen innerhalb solcher Variablen problemlos einsetzen lassen, können Sessionvariablen in sehr vielen unterschiedlichen Einsatzgebieten angewendet werden. Im vorherigen Unterkapitel wurde bereits das Beispiel rund um den Online-Warenkorb durchgespielt, welches verdeutlicht hat, inwiefern Sessiondaten gespeichert sowie schlussendlich wieder indiziert werden können. Im Regelfall handelt es sich bei solchen Daten um sehr sensible Benutzerinformationen, welche normalerweise mit einer Verschlüsselungsmethode wie z.B. MD5 oder SHA1 gesichert sind. Man kann aber auch ganz einfache Systemdaten in solche Sessionvariablen schleusen, welche im weiteren Sinne Systemaufgaben anstatt Benutzeraktionen betreuen.

In der Praxis sind beispielsweise multilinguale Sprachunterstützungen zum Ändern der Sprache bzw. Captchafunktionen, welche eine starke Barriere gegen Internetbots darstellen, beliebte Tools, welche ebenfalls innerhalb von Sessionvariablen implementiert werden können. Verbindet

[31] vgl. *Williams & Lane* (2005), S.365 f.

man nämlich die Sessionvariablen mit sogenannten `$_GET`-Variablen, kann der Status, welcher über eine `$_GET`-Variable verschickt wurde, abgefragt und mit der aktuellen Sessionvariable abgeglichen werden. So lässt sich beispielsweise beim multilingualen Support die Sessionvariable `$_SESSION['lang'] = $lang;` durch die Variable `$lang = $_GET['lang'];` beeinflussen, welche bei jedem Aufruf den entsprechenden Wert innerhalb der Variable `$lang` ändert. Für Captchafunktionen gilt prinzipiell genau das gleiche. Hier wird der zufallsgenerierte Captchawert mittels der PHP Funktion `imagestring()` in ein PNG- oder JPEG-Bild geschrieben, wodurch sich der Text nicht mehr markieren bzw. kopieren lässt und damit gezwungenermaßen von Hand eingegeben werden muss. Dabei wird der Captchawert ebenfalls in einer Sessionvariable gespeichert, damit der Webserver – beispielsweise nach dem Absenden eines Formulars – nach wie vor den Captchawert kennt und diesen mit der Benutzereingabe abgleichen kann.

0.1.4 Cookies

Dadurch, dass HTTP ein zustandsloses Protokoll ist, geraten Daten spätestens dann in Vergessenheit, sobald die Website an den Client gesendet und die Verbindung wieder getrennt wurde. Im Jahr 1994 implementierte Netscape im hauseigenen Browser schlussendlich das Cookie, welches nur von jener Webseite gelesen werden konnte, von der es auch geschrieben wurde. Somit stellte es eine sichere Art dar, um Informationen dauerhaft über Seiten hinweg zu speichern, obwohl es anfänglich eher einem schlechten Ruf unterlag, da der Host darüber feststellen kann, wie oft eine BenutzerIn die jeweilige Webseite aufruft und was dort gemacht wird. Viele Menschen hatten plötzlich große Sorge um ihre Privatsphäre innerhalb des Internets, wodurch unter anderem auch zahlreiche Gerüchte entstanden. Beispielsweise wurde Cookies nachgesagt, dass sie jede Information auf der Festplatte lesen könnten, wodurch viele Zeitschriften und Blogs auf diesen Zug aufsprangen und sogar zur Anwendungsdeaktivierung von Cookies im Webbrowser rieten. Mittlerweile hat sich die vorerst angespannte Situation wieder beruhigt und Cookies werden im Allgemeinen von der Mehrheit akzeptiert.[32]
Prinzipiell stellen Cookies lediglich reine Textinformationen dar, welche auf Aufforderung eines Webservers durch den Internetbrowser auf der Festplatte des Clients abgespeichert werden.[33] Wenn die gleiche Webseite oder eine Seite der gleichen Domäne erneut aufgerufen wird, so schickt der Browser die im Cookie enthaltenen Textinformationen an den Webserver zurück. Der Webserver selbst ist aber lediglich in der Lage, ihm bereits bekannte Informationen im Cookie abzuspeichern, sodass auch nur solche Informationen vom Webbrowser an ihn zurückgesandt werden. Die oben genannte Einschränkung kann aber auch soweit technisch umgangen werden, dass andere Domänen ebenfalls Gebrauch von einem abgespeicherten Cookie machen können. In diesem Fall spricht man allerdings von Cookies von Drittanbietern *(Abb. 2.3.7)*.[34] Selbstverständlich besteht auch die Möglichkeit, in aktuellen Webbrowsern wie Safari, Chrome, Firefox oder Opera, die Speicherung von Cookies komplett zu deaktivieren. Diese Option kann aber unter Umständen dazu führen, dass der ordnungsgemäße Aufbau einer Seite, sowie ein reibungsloses Surfen an sich, nicht mehr gewährleistet werden kann.[35] Grundsätzlich kann man also sagen, dass es sich bei Cookies im weiteren Sinne um eine Art Cachespeicher handelt, durch den eine bestimmte Anzahl von Informationen gespeichert wird, wodurch eine BenutzerIn vom Webserver wiedererkannt werden kann.[36]

Als WebentwicklerIn kann man sowohl auf Cookies als auch auf Sessions problemlos zugreifen. Es ist schlussendlich mit beiden Techniken möglich, Daten einer Seite über mehrere andere hinweg

[32] vgl. *Hudson* (2005), S.181.
[33] s. *Cookie Central*: The Cookie Concept, (Stand: 04.10.2016).
[34] vgl. *Peyton* (2002), S.53 f.
[35] vgl. *Christl* (2014), S.21.
[36] vgl. *Voss* (2003), S.229.

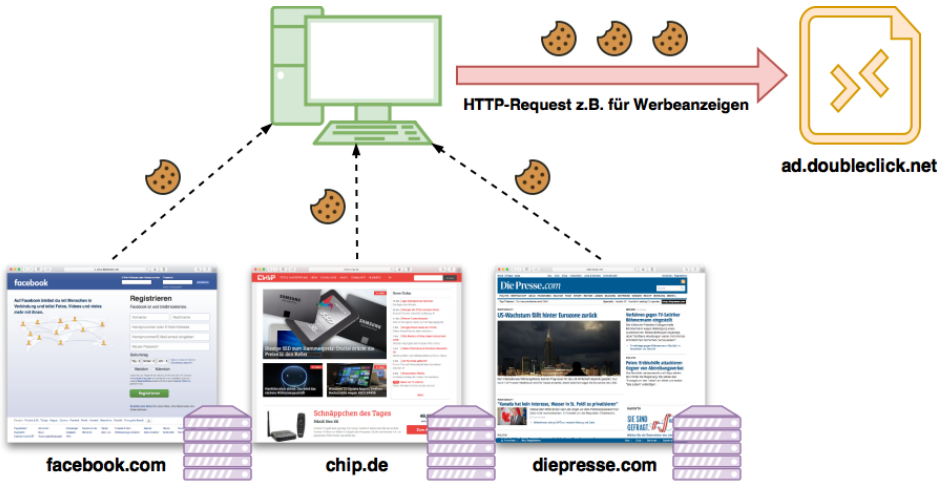

Abbildung 0.1.7: Abfrage von Cookieinformationen durch Drittanbieter anderer Domänen

zu speichern, wobei diese auch einige Unterschiede aufweisen. Cookies können nämlich – ganz im Unterschied zu Sessions – auf eine lange Lebensdauer eingestellt werden, wodurch Daten innerhalb eines Cookies über Monate bzw. Jahre gespeichert werden können. Da Cookiedaten clientseitig gespeichert werden, ist die Speicherung sensibler Daten allerdings nur dann sinnvoll, wenn z.B. in einem Cluster mehrerer Webserver gearbeitet wird. Sessions wären hier nicht nützlich, da nach der ersten fertig abgearbeiteten Anfrage eines Webservers die Informationen im Cluster schlichtweg nicht mehr verfügbar wären. Außerdem weisen viele moderne Webbrowser eine Speicherplatzbegrenzung bei der Speicherung von Cookies auf, um schädliche Internetseiten davor zu hindern, immens viel Speicherplatz in Form von Cookies auf der Festplatte zu verschwenden. Schlussendlich beruht die Entscheidung, ob man in seinem Webprojekt nun Cookies oder doch lieber Sessions verwenden sollte darauf, ob die Daten noch am nächsten Tag für die BenutzerIn zur Verfügung stehen sollen. Falls dem so sein sollte, kommt man um Cookies nicht herum. Falls sensible Daten gespeichert werden sollen, ist es empfehlenswert, diese in einer gesicherten Datenbank abzulegen und lediglich eine Referenz zur ID im Cookie zu speichern. Dies deshalb, da Cookies dadurch, dass sie am Computer gespeichert werden, von BenutzerInnen sehr leicht bearbeitet werden können. So könnte man beispielsweise die Benutzer-ID, welche zur automatischen Anmeldung bei einer Webseite benötigt wird, umschreiben und somit in den Account der jeweils geänderten Benutzer-ID einsteigen. Genau deshalb werden heutzutage auch oft Sicherheitstoken innerhalb solcher Login-Systeme als eine Art zusätzliche Hackerbarriere implementiert, welche sich aus verschiedenen Benutzerinformationen zusammensetzen und mittels der Benutzer-ID und sonstigen Daten einen eindeutig feststellbaren Hashtag ergeben.[37]

Verlagert man nun die gesamte Thematik zurück zu PHP, muss unbedingt erwähnt werden, dass der Aufruf der Methode `setcookie()` immer vor dem HTML-Formular stehen muss, da HTTP alle Headerinformationen vor dem `<body>`-Tag sendet. In den Kopfzeilen werden Informationen wie beispielsweise der Webservertyp (z.B. Apache), die Seitengröße in Bytes sowie andere wichtige Daten übertragen. Cookies gehören ebenfalls zu diesen Headerinformationen, was heißt, dass nach jedem `setcookie()` der Webserver eine Zeile in den Header schreibt. Die `setcookie`-Funktion kann dabei drei Hauptparameter annehmen, nämlich den Cookienamen, den Wert sowie das Datum wann das Cookie wieder verfallen soll *(Abb. 2.3.8)*.[38] Im einfachen Fall wird in einem Cookie genau ein Wert gespeichert (max. 4 Kilobyte). PHP bietet aber auch die Möglichkeit, eine Arrayvariable als Cookie abzuspeichern, wodurch es ermöglicht wird, die

[37] vgl. *Hudson* (2005), S.181 f.
[38] vgl. *ebd.*, S.182 f.

Werte in einer Schleife abzuarbeiten. Für den Fall, dass noch komplexere Daten in einem Cookie gespeichert werden sollen, gibt es die Möglichkeit, den Variableninhalt mittels der Methode `serialize()` zu serialisieren, indem eine Zeichenkette aus der komplexen Variable erstellt wird.[39]

```
1  /* Erstellung eines Cookies namens "thesis-cookie" */
2  /* Ablaufdatum: aktuelle Zeit + 5000 Tage */
3  setcookie('thesis-cookie', $data, time() + (86400 * 5000), "/");
4
5  /* Entfernung des Cookies "thesis-cookie" */
6  /* Inhalt wird gelöscht / Ablaufdatum: aktuelle Zeit */
7  setcookie('thesis-cookie', '', time() - (86400 * 5000), "/");
```

Abbildung 0.1.8: Minimalbeispiel zur Erstellung und Entfernung eines Cookies

Der größte Nachteil von Cookies liegt sicherlich darin, dass man sich einfach nicht auf sie verlassen kann. Es lässt sich nämlich nicht mit Gewissheit sagen, ob ein Browser auch wirklich ein Cookie nach einer Sitzung gelöscht hat, oder ob sie eine BenutzerIn aufgrund von Sicherheitsbedenken eigenhändig entfernte bzw. ob sie sogar deaktiviert wurden. Darum ist es generell keine gute Idee, gerade bei sicherheitskritischen Bereichen wie z.B. Anmeldesystemen, Cookies zu verwenden, außer man rüstet sein Login-System mit einem Sicherheitstoken aus. Ein weiterer großer Nachteil von Cookies ist, dass man sie nur vor dem Ausliefern eines Seiteninhalts erstellen bzw. bearbeiten kann. Diese Einschränkung ist innerhalb der Spezifikation von HTTP geregelt und ist nicht PHP bedingt. Des Weiteren werden Cookies nach dem Aufruf der Methode `setcookie()` lediglich erstellt. Erst nach dem Laden der nächsten Seite wird das Cookie auch im Webbrowser als aktiv angezeigt, was ein häufig auftauchender „Fehler“ in so manchem Forum ist.[40]

[39] vgl. *Kofler & Öggl* (2010), S.171.
[40] vgl. *ebd.*, S.171 f.

Literatur- & Quellenverzeichnis

Beighley, Lynn & Morrison, Michael (2009): PHP & MySQL von Kopf bis Fuß; übersetzt von: Schulten, Lars; 1. Auflage, O'Reilly Verlag, Köln.

Christl, Alexander (2014): Datenschutz im Internet - Cookies, Web-Logs, Location Based Services, eMail, Webbugs, Spyware; 1. Auflage, Disserta Verlag, Hamburg.

Hudson, Paul (2005): PHP in a nutshell; übersetzt von: Speidel, Sigrid & Ulrich; 1. Auflage, O'Reilly Verlag, Köln.

Kofler, Michael & Öggl, Bernd (2010): PHP 5.3 & MySQL 5.4 - Programmierung, Administration, Praxisprojekte; 1. Auflage, Addison-Wesley Verlag, München.

Maurice, Florence (2015): PHP 5.6 und MySQL 5.7 - Ihr praktischer Einstieg in die Programmierung dynamischer Websites; 4. Auflage, Dpunkt Verlag, Heidelberg.

Peyton, Christine (2002): Das neue PC Lexikon für Alle; 1. Auflage, Sybex Verlag, Düsseldorf.

Schlossnagle, George (2006): Professionelle PHP 5 - Programmierung; 1. Auflage, Addison-Wesley Verlag, München.

Theis, Thomas (2014): Einstieg in PHP 5.6 und MySQL 5.6; 10. Auflage, Rheinwerk Verlag, Bonn.

Voss, Andreas (2003): Das große PC & Internet Lexikon; 1. Auflage, Data Becker Verlag, Düsseldorf.

Wassermann, Tobias (2007): Sichere Webanwendungen mit PHP; 1. Auflage, Mitp Verlag, Heidelberg.

Williams, Hugh E. & Lane, David (2005): Webdatenbank-Applikationen mit PHP und MySQL; 2. Auflage, O'Reilly Verlag, Köln.

Cookie Central: The Cookie Concept; siehe online unter: http://www.cookiecentral.com/c_concept.htm (Stand: 04.10.2016).

Envatotuts Plus: PHP 5.6 What's New; siehe online unter: http://code.tutsplus.com/articles/php-56-whats-new--cms-22101 (Stand: 28.09.2016).

PHP.net: Funktion require_once; siehe online unter: http://php.net/manual/de/function.require-once.php (Stand: 29.09.2016).

PHP.net: Migrating from PHP 5.6.x to PHP 7.0.x; siehe online unter: http://php.net/manual/de/migration70.php (Stand: 28.09.2016).

PHP.net: PHP 5.6 Changed Functions; siehe online unter: http://php.net/manual/en/migration56.changed-functions.php (Stand: 28.09.2016).

Abbildungsverzeichnis