Wordpress Handbuch

http://wordpress.lernenhoch2.de/Handbuch/

---

1. Schritt 1 – die grobe Template-Struktur

Im ersten Schritt werden wir nach ein paar kurzen Vorbereitungen unser Template anlegen. Dabei wirst du dich mit den Wordpress Template Tags vertraut machen und sehen wie ein Template arbeitet. Für ein einfaches Wordpress Template reichen zwei Dateien, die **index.php** und die

style.css. Warum das so ist und wie das alles funktioniert, siehst du in den folgenden Kapiteln.

1.1. Vorbereitungen

Bevor du dein Theme erstellen und testen kannst, musst du natürlich Wordpress installiert haben.

- Wordpress auf deutsch – **download**
- Wordpress auf englisch – **download**
- **Anleitung – 5 Minuten Installation**

Template Ordner anlegen

Jetzt tauchen wir auch schon direkt ein und erstellen im Theme-Ordner (wo alle Wordpress Templates reinkommen) unseren eigenen Ordner für unser Theme.

/wp-content/themes/

Unser Template nennen wir “Tutorial”, erstelle daher einen Ordner mit dem Namen **tutorial** im Ordner **/wp-content/themes/**. Der Pfad sollte dann so aussehen:

/wp-content/themes/tutorial/

So, jetzt können wir mit unserem eigenen Template beginnen, hol dir noch nen Kaffee (Tee, Wasser, …) und los gehts.

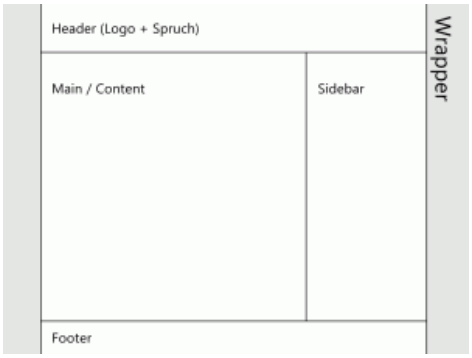
1.2. Die wichtigste Template-Datei: Index.php

Die wichtigste Template Datei ist index.php mit dieser Datei alleine, hast du schon ein komplettes Template. Wir werden später sehen, dass es übersichtlicher ist und Sinn macht, mehrere Dateien zu haben, doch erstmal beginnen wir mit dieser einen. Leg also in deinem Template-Ordner die Datei **index.php** an. Das ganze sollte nun folgendermaßen aussehen:

wp-content/themes/tutorial/index.php

Wordpress Theme “Tutorial” – Wireframe

Bevor wir in die Tasten hauen, möchte ich dir gerade noch die Struktur (das Wireframe) zeigen, die unser Template haben wird:



Zu sehen sind:

- **Wrapper** – er enthält alle anderen Bereiche und wird zentriert auf der Seite ausgegeben
- **Header** – der Kopfbereich unseres Layouts enthält den Namen der Seite, bzw das Logo der Seite, sowie das Motto der Seite, welches in Wordpress “Tagline” genannt wird.
- **Main / Content** – dort werden die Artikel/Inhalte/Blogbeiträge angezeigt
- **Sidebar** – das Menü auf der rechten Seite enthält Links zu Kategorien, zum Archiv, Auflistung der Seiten, etc.
- **Footer** – dort platzieren wir wichtige Links (z.b. zum Impressum unserer Seite), neueste Kommentare, etc.

HTML Struktur

Bislang ist unsere index.php noch leer, deshalb packen wir jetzt unser komplettes HTML Gerüst hinein, öffne deine index.php und füge folgenden Code ein:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//DE" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="de">
<head profile="http://gmpg.org/xfn/11">

   <!-- wird noch ausgefüllt -->

</head>
<body>

<div id="wrapper">

   <div id="header"></div><!-- header -->

   <div id="main"></div><!-- main -->

   <div id="sidebar"></div><!-- sidebar -->

   <div id="footer"></div><!-- footer -->

</div><!-- wrapper -->

</body>
</html>
```

Bislang ist unser Template immernoch unbrauchbar, aber du hast einen wichtigen Schritt gemeistert, die Hauptstruktur steht. Im nächsten Teil befassen wir uns mit dem CSS-Stylesheet um die einzelnen Blöcke in Form zu bringen.

1.3. Template Stylesheet

Für die Funktion eines Wordpress Templates sind die einzelnen PHP-Dateien verantwortlich, in unserem Fall bislang nur die index.php. Doch um HTML visuell aufzupeppen (und damit Wordpress unser Template “erkent”), brauchen wir ein CSS-Stylesheet. Deshalb lege jetzt in deinem Template-Ordner die Datei **style.css** an:

/wp-content/themes/tutorial/style.css

Nun fügen wir folgenden Code in unser Stylesheet ein, um die Hauptstruktur unseres Templates zu formen:

```
body        {color: #333; background: #e5e5e5;}

#wrapper    {width: 850px; margin: auto; text-align: left; background: #fff;}
#header        {height: 160px; padding: 20px; }
#main       {width: 520px; padding: 20px; float: left;}
#sidebar    {width: 270px; padding: 10px; padding-top: 20px; float: left;}
#footer        {clear: both; height: 100px; padding: 20px; }
```

Dieser Teil ist nur dafür da, unserem Template eine grobe Struktur zu geben, aber ein wichtiger Teil fehlt noch. Wenn du im Wordpress Backend auf der “Manage Themes” Seite gehst, siehst du alle Themes, die du in deinem Theme Ordner hast. Unser “Tutorial”-Theme wird aber nicht angezeigt, das werden wir jetzt ändern. Füge an den Anfang der **style.css** folgenden Code:

```
/*
Theme Name: Tutorial
Theme URI: http://wordpress.lernenhoch2.de
Description: Template zum Wordpress Tutorial auf http://wordpress.lernenhoch2.de
Author: Christian Strang
Author URI: http://lernenhoch2.de
*/
```

Erklärung:

- **Theme Name**: Dies ist der wichtigste Wert und zeigt im Theme Manager das Template mit seinem Namen an
- **Theme URI**: Wenn du vorhast, dein Template zu verteilen, solltest du hier die URL angeben, unter der man es finden kann. So können User, die dein Theme nutzen, nach einer aktualisierten Version, behobenen Bugs, etc. suchen

- **Description**: eine kurze Beschreibung zu deinem Template, wie es aussieht (2 Spalten, weiß/blau, simpel…) oder für welchen Zweck (Kinder-Blog, Schul-Blog, etc.) oder was du sonst über dein Template sagen möchtest
- **Author**: Der Name desjenigen, der das Template erstellt hat
- **Author URI**: Die URL zu deinem blog, deiner Website

Da wir noch keinen Inhalt haben, bzw. noch nichts mit Wordpress ausgeben, können wir nur hoffen, dass das Layout so funktioniert. Im nächsten Teil werden wir es dann erfahren :)

1.4. Loop – unsere erste Ausgabe

Die Wordpress Schleife, allgemein bekannt als "**the loop**", ist das wichtigste Konstrukt in unserem Template, denn ohne die Schleife wird kein Inhalt angezeigt. Im folgenden siehst du die Loop, die wir verwenden werden:

```
1 <?php if (have_posts()) : while (have_posts()) : the_post(); ?>
2    <h2><?php the_title(); ?></h2>
3    <div class="entry">
4       <?php the_content(); ?>
5    </div>
6 <?php endwhile; endif; ?>
```

Der Quellcode ist einfacher als er aussieht:

- **Zeile 1** – Wordpress prüft, ob Artikel vorhanden sind mittles "have_posts()". Findet die Wordpress Engine Artikel, startet es die Schleife mittels "while" und durchläuft alle Inhalte. ***1** weitere Erklärungen siehe unten.
- **Zeile 2** – das geschulte HTML-Auge sieht das öffnende und schließende h2-Tag. Darin nutzen wir eine Wordpress-Funktion namens "the_title()". Mit dieser Funktion geben wir den Titel des aktuellen Artikels aus. **Achtung**, diese Funktion funktioniert nur innerhalb der Schleife, das hat einen einfachen Grund: Angenommen wir befinden uns nicht innerhalb der Schleife, woher soll Wordpress wissen, welchen Artikel-Titel du ausgeben möchtest? Ist logisch, oder?
- **Zeile 3** – wir öffnen eine CSS-Box mit dem Klassen-Namen "entry". Innerhalb dieser Box wird jeweils ein Artikel ausgegeben
- **Zeile 4** – die Funktion "the_content()" arbeitet ähnlich wie "the_title" aber anstelle des Titels vom aktuellen Artikel, gibt sie den Artikel selbst aus, sprich: den Inhalt des Artikels
- **Zeile 5** – die Div-Box wird geschlossen
- **Zeile 6** – die Schleife wird beendet (wenn alle Artikel durchlaufen sind) und die Bedingungs-Abfrage wird geschlossen

***1** vielleicht hast du dich gefragt was "*Wordpress prüft, ob Artikel vorhanden sind*" bedeutet. Abhängig von der Seite wo du dich gerade befindest, beinhaltet "have_posts" unterschiedliche Artikel. Angenommen du befindest dich auf deiner Startseite, dann hat "have_posts" die letzten 10 Artikel die du geschrieben hast. Befindest du dich in einer Kategorie, dann hat "have_posts" die letzten 10 Artikel, die du in dieser Kategorie geschrieben hast. Wenn dein Blog eine Suchfunktion hat und du suchst nach dem Begriff "Wordpress", dann hat "have_posts" die letzten 10 Artikel, in denen der Begriff "Wordpress" gefunden wurde. Du siehst also, **have_posts** ist abhängig davon, wo du dich auf deinem Blog befindest.

Aus diesem Grund reicht für ein voll funktionsfähiges Wordpress Template, die index.php alleine vollkommen aus.

index.php mit "the loop" zum Leben erwecken

Du hast die Schleife gesehen und ihre Funktionsweise verstanden, jetzt muss sie nur noch ins Template und wir können unsere Inhalte ausgeben. Dazu modifizieren wir die index.php folgendermaßen:

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//DE" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
02 <html xmlns="http://www.w3.org/1999/xhtml" lang="de">
03 <head profile="http://gmpg.org/xfn/11">
04
05    <!-- wird noch ausgefüllt -->
06
07 </head>
08 <body>
09
10 <div id="wrapper">
11
12    <div id="header"></div><!-- header -->
13
14    <div id="main">
```

```
15
16       <?php if (have_posts()) : while (have_posts()) : the_post(); ?>
17          <h2><?php the_title(); ?></h2>
18          <div class="entry">
19             <?php the_content(); ?>
20          </div>
21       <?php endwhile; ?>
22
23          <p align="center"><?php next_posts_link('&laquo; &Auml;ltere Eintr&auml;ge') ?> | <?php
    previous_posts_link('Neuere Eintr&auml;ge &raquo;') ?></p>
24
25       <?php endif; ?>
26
27    </div><!-- main -->
28
29    <div id="sidebar"></div><!-- sidebar -->
30
31    <div id="footer"></div><!-- footer -->
32
33 </div><!-- wrapper -->
34
35 </body>
36 </html>
```

Die Schleife kommt in die “main”-Box, denn dort soll unser Content angezeigt werden und das wollen wir direkt mal ausprobieren. Geh ins Backend und aktiviere das “Tutorial”-Theme. Wenn du schon ein paar Artikel hast, solltest du diese jetzt (völlig unformatiert) angezeigt bekommen. Unser Template nutzt bislang noch nicht die style.css, da wir den “head”-Bereich bislang leergelassen haben (dazu kommen wir im nächsten Teil).

Ich habe noch eine kleine Ergänzung in Zeile 23 eingefügt. Standardmäßig zeigt Wordpress die letzten 10 Artikel an (läßt sich im Backend einstellen). Wenn aber dein Blog schon mehr als 10 Artikel hat, können die User diese nicht sehen. Deshalb fügen wir am Ende der Schleife die zwei Links “Ältere Einträge” und “Neuere Einträge” ein. Auf der Startseite wird nur der Link “Ältere Einträge” angezeigt (da es keine neueren gibt, als die, die auf der Startseite zu sehen sind).

Titel verlinken und Einzel-Artikel anzeigen

Jetzt wollen wir noch eine kleine Änderung vornehmen und zwar verlinken wir den Artikel-Titel mit dem Artikel selbst. Dadurch kann man von der Startseite ganz schnell zum Einzel-Artikel gelangen. Dazu nehmen wir nur eine kleine Änderung an der index.php vor:

```
01 <div id="main">
02    <?php if (have_posts()) : while (have_posts()) : the_post(); ?>
03       <h2><a href="<?php the_permalink() ?>"><?php the_title(); ?></a></h2>
04       <div class="entry">
05          <?php the_content(); ?>
06       </div>
07    <?php endwhile; ?>
08       <p align="center"><?php next_posts_link('&laquo; &Auml;ltere Eintr&auml;ge') ?> | <?php
    previous_posts_link('Neuere Eintr&auml;ge &raquo;') ?></p>
09    <?php endif; ?>
10 </div><!-- main -->
```

Die Änderung ist einfach: Um unseren Titel stricken wir einen Link und das href-Attribut des Links, füllen wir mit der Funktion “the_permalink()”. Wie **the_title()** und **the_content()** ist auch **the_permalink()** abhängig von der Loop und gibt den Link zum Artikel aus. Es gibt noch viele weitere, nützliche Angaben, die man über die loop von den einzelnen Artikeln erhalten kann, z.b. das Datum des Artikels, den Author, die Kategorie, etc. doch dazu später mehr.

Wenn du nun auf der Startseite einen Artikel-Titel anklickst, solltest du auf der Einzel-Artikel Seite landen. Und wie du siehst: Auch wenn es nicht schön aussieht, so haben wir doch mit einer einzigen Template Datei ein voll funktionsfähiges Wordpress Theme. Da wir aber die einzelnen Seiten spezialisieren wollen und unser Template übersichtlich bleiben soll, werden wir in den folgenden Tutorial-Teilen das Template in mehrere Template-Dateien aufsplitten.

1.5. Den Head-Bereich füllen

Im letzten Teil hast du deine erste Ausgabe mit Wordpress gemacht, doch leider war das Layout nicht vorhanden, weil wir die style.css noch nicht verlinkt hatten. Darum geht es nun in diesem Teil, den Head-Bereich unseres Layouts zu füllen.

Im Head-Bereich zeigt Wordpress z.b. den Artikeltitel an, bindet Stylesheets und Javascripts ein und fügt automatisch weitere nützliche Meta-Angaben ein, wenn wir Wordpress darum bitten (was wir definitiv machen werden, da es uns Arbeit erspart). Bislang haben wir folgendes:

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//DE" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2 <html xmlns="http://www.w3.org/1999/xhtml" lang="de">
3 <head profile="http://gmpg.org/xfn/11">
4
5    <!-- wird noch ausgefüllt -->
6
7 </head>
8 ...
```

Den HTML-Kommentar "**wird noch ausgefüllt**", ersetzen wir jetzt durch folgende Angaben:

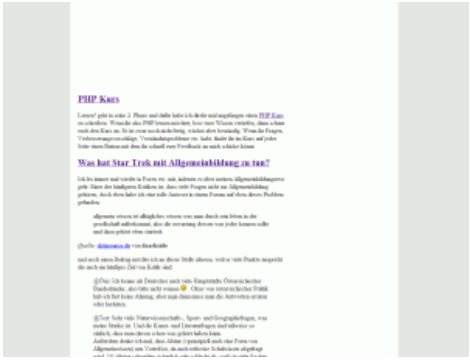
```
1 <meta http-equiv="Content-Type" content="<?php bloginfo('html_type'); ?>; charset=<?php bloginfo ('charset'); ?>" />
2
3 <title><?php wp_title(); ?> - <?php bloginfo('name'); ?></title>
4
5 <link rel="stylesheet" href="<?php bloginfo('stylesheet_url'); ?>" type="text/css" media="screen" />
6 <link rel="pingback" href="<?php bloginfo('pingback_url'); ?>" />
7
8 <?php wp_head(); ?>
```

Erklärung:

- **Zeile 1:** das Meta Tag füllen wir mit dem Charset, welches für den Blog festgelegt ist. In der Regel dürfte das UTF-8 sein und damit liegt man meistens richtig
- **Zeile 3:** das Title-Tag füllen wir mit dem Namen des Artikels (auf der Einzelansicht) + den Namen des Blogs. An dieser Stelle kann man recht kreativ mit den Wordpress Template Tags rumspielen, diese Struktur ist aber für die meisten Blogs gut und ausreichend.
- **Zeile 5:** jetzt verlinken wir endlich auf unsere **style.css**. Dazu nutzen wir das mächtige Wordpress Template Tag "**bloginfo()**" mit dem Parameter "**stylesheet_url**" um die Link zur style.css zu erhalten
- **Zeile 6:** damit Blogs, die in ihrem Artikel auf einen von deinen Artikeln gelinkt haben, deinen Blog darüber informieren können (Pingback), nutzen wir den Pingback-Link und nutzen wieder "bloginfo()" um uns den Link erzeugen zu lassen
- **Zeile 8:** dies ist eine sehr wichtige Funktion, denn viele Plugins funktionieren nur dann korrekt, wenn die Funktion im head-Bereich platziert wurde. Manche Plugins platzieren ihre eigenen Stylesheets oder Javascripts im Head-Bereich und das funktioniert nur, wenn dieser "Hook" gesetzt wurde.

Wenn wir uns jetzt unser Layout anschauen, ist es zwar immernoch schlicht, aber wenigstens so wie wir es geplant hatten. In den weißen Bereich oben platzieren wir später unser Logo und den Spruch unseres Blogs, sieh dir dazu nochmal das **Wireframe des Wordpress Theme "Tutorial" an**.

Ich habe das Tutorial Template mal testweise im Lernen² Blog verwendet und folgendes Ergebnis bekommen:



Bei dir sollte es ähnlich aussehen (natürlich mit anderen Titeln und Artikeln).

---

2. Schritt 2 – die einzelnen Template Dateien

Auch wenn für ein voll funktionsfähiges Wordpress Theme die index.php und die style.css vollkommen ausreichen, so will man doch hin und wieder

die unterschiedlichen Seiten individuell gestalten. So möchte man z.b. dass die Kategorie-Übersicht anders aussieht als die index.php oder das sich Seiten (page.php) optisch von Beiträgen (single.php) unterscheiden.

Wie arbeitet Wordpress mit Theme Dateien?

Um die Arbeitsweise von Wordpress in Bezug auf Template-Dateien zu verstehen, empfiehlt sich folgende Grafik anzuschauen: **Wordpress Template Hierarchie**.

Wordpress Template Dateien

Wordpress "kennt" einige typische Template Dateien, mit denen man sein Theme einfach und übersichtlich spezialisieren kann:

- **header.php** – Template Kopfteil, HTML Meta Angaben, Stylesheets und Javascripts
- **footer.php** – Template-Abschluss
- **sidebar.php** – Menübereich
- single.php – Template für die Anzeige von Beiträgen
- page.php – Template für die Anzeige von Seiten
- comments.php – Template für den Kommentarbereich und die Kommentare
- 404.php – Template für den 404 – Seite nicht gefunden – Fehler
- search.php – unsere eigene, kleine Blogsuche

Neben diesen typischen Wordpress Template Dateien, kann man auch spezialisierte Template Dateien erzeugen. So kann man z.b. eine page.php Datei haben, die für alle Seiten genutzt wird und eine page-9.php haben, die nur für die Seite mit der ID 9 genutzt wird. Dazu ist folgendes sehr aufschlussreich: **The Template Hierarchy In Detail**

2.1. header.php – den Kopfteil auslagern

Unsere erste spezialisierte Template-Datei nennt sich **header.php**
Unsere Header.php beinhaltet unter anderem den kompletten HTML-Head-Teil unseres Templates. Da sich dieser Bereich auf allen Seiten, Beiträgen, Kategorien, Archiven, etc. nicht ändert, ist er perfekt dafür geeignet, in einer extra Datei ausgelagert zu werden.

head auslagern und einbinden

Als erstes erstellen wir in unserem Template-Ordner die Datei "header.php". Danach öffnen wir unsere "index.php" und schneiden den markierten Teil aus, und kopieren ihn in die header.php:

index.php

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//DE" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
02 <html xmlns="http://www.w3.org/1999/xhtml" lang="de">
03 <head profile="http://gmpg.org/xfn/11">
04
05    <meta http-equiv="Content-Type" content="<?php bloginfo('html_type'); ?>; charset=<?php bloginfo('charset'); ?>" />
06
07    <title><?php wp_title(' - ', true, 'right'); ?> <?php bloginfo('name'); ?></title>
08
09    <link rel="stylesheet" href="<?php bloginfo('stylesheet_url'); ?>" type="text/css" media="screen" />
10    <link rel="pingback" href="<?php bloginfo('pingback_url'); ?>" />
11
12    <?php wp_head(); ?>
13
14 </head>
15 <body>
16
17 <div id="wrapper">
18
19    <div id="header"></div><!-- header -->
20
21    <div id="main">
22
23       <?php if (have_posts()) : while (have_posts()) : the_post(); ?>
24          <h2><a href="<?php the_permalink() ?>"><?php the_title(); ?></a></h2>
25          <div class="entry">
```

```
            <?php the_content(); ?>
         </div>
      <?php endwhile; endif; ?>

   </div><!-- main -->

   <div id="sidebar"></div><!-- sidebar -->

   <div id="footer"></div><!-- footer -->

</div><!-- wrapper -->

</body>
</html>
```

header.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//DE" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="de">
<head profile="http://gmpg.org/xfn/11">

   <meta http-equiv="Content-Type" content="<?php bloginfo('html_type'); ?>; charset=<?php bloginfo('charset'); ?>" />

   <title><?php wp_title(' - ', true, 'right'); ?> <?php bloginfo('name'); ?></title>

   <link rel="stylesheet" href="<?php bloginfo('stylesheet_url'); ?>" type="text/css" media="screen" />
   <link rel="pingback" href="<?php bloginfo('pingback_url'); ?>" />

   <?php wp_head(); ?>

</head>
<body>

<div id="wrapper">

   <div id="header"></div><!-- header -->
```

Jetzt haben wir zwar den head-Bereich komplett ausgelagert, aber davon weiß unsere index.php noch nichts. Um unserem Template das zu sagen, brauchen wir folgenden Befehl:

```
<?php get_header(); ?>
```

“get_header()” ist eine Wordpress-Funktion, die im Template-Ordner nach der Datei header.php sucht und wenn sie sie findet, an der Stelle einbindet, an der der “get_header”-Befehl steht. Unsere **index.php** sollte nach der Änderung folgendermaßen aussehen:

```
<?php get_header(); ?>

   <div id="main">

      <?php if (have_posts()) : while (have_posts()) : the_post(); ?>
         <h2><a href="<?php the_permalink() ?>"><?php the_title(); ?></a></h2>
         <div class="entry">
            <?php the_content(); ?>
         </div>
      <?php endwhile; endif; ?>

   </div><!-- main -->

   <div id="sidebar"></div><!-- sidebar -->

   <div id="footer"></div><!-- footer -->

</div><!-- wrapper -->

</body>
```

```
</html>
```

Wenn du jetzt dein Template testest, sollte sich eigentlich nichts geändert haben. Von aussen ist es gleich, doch unter der Haube haben wir einen wichtigen Schritt zu einem tollen Template gemacht. Funktionalität auszulagern, ist ein essentieller Bestandteil von Wordpress-Templates. Nicht nur, dass das Template übersichtlicher wird (wenn ein Problem im head-Bereich ist, findet man die Datei ganz schnell), wir können damit auch die Ansicht der unterschiedlichen Bereiche (Beiträge, Seiten, Kategorien, etc.) spezialisieren.

Wie du vielleicht schon richtig erahnt hast, gibt es neben der Funktion “get_header()” noch weitere, mit denen man andere Template-Dateien einbinden kann. Darauf gehe ich in den folgenden Kapiteln ein.

2.2. Footer.php – der Fußbereich

Mit der Datei **footer.php** wird unser Template abgeschlossen. Darin platzieren wir unter anderem die abschließenden HTML-Tags für **<html>** und **<body>**, sowie das schließende DIV-Element für unsere “Wrapper”-Box. Die footer.php bietet sich auch als Platz für einen Tracking-Code an, z.b. für **Google Analytics** oder andere Website-Analyse-Tools.

Da der footer-Bereich einer Website eher von wenigern Besuchern gesehen wird, sollten an dieser Stelle nur Elemente platziert werden, die nicht so wichtig sind. Unter anderem kann man hier:

- eine Liste von Zufallsartikeln ausgeben
- die neuesten Kommentare
- eine TagCloud
- eine Liste der beliebtesten Artikel
- usw.

Aus unserer jetzigen **index.php** schneiden wir den markierten Teil aus und fügen ihn in die Datei **footer.php** ein:

```
<?php get_header(); ?>

   <div id="main">

      <?php if (have_posts()) : while (have_posts()) : the_post(); ?>
         <h2><a href="<?php the_permalink() ?>"><?php the_title(); ?></a></h2>
         <div class="entry">
            <?php the_content(); ?>
         </div>
      <?php endwhile; endif; ?>

   </div><!-- main -->

   <div id="sidebar"></div><!-- sidebar -->

   <div id="footer"></div><!-- footer -->

</div><!-- wrapper -->

</body>
</html>
```

footer.php

```
   <div id="footer"></div><!-- footer -->

</div><!-- wrapper -->

</body>
</html>
```

Zusätzlich fügen wir noch die markierten Zeilen ein:

```
   <div id="footer">
      <!-- unwichtigere Funktionen -->
```

```
   </div><!-- footer -->

</div><!-- wrapper -->

<?php wp_footer(); ?>

<!-- Statistik/Analyse-Tool einbauen -->

</body>
</html>
```

Die Funktion “wp_footer()” ist genau wie “wp_head()” ein Template-”Hook”. Einige Plugins basieren darauf und funktionieren nur korrekt, wenn er gesetzt wurde, deshalb packen wir ihn hier mit rein. Der HTML-Kommentar dient als Platzhalter, damit du später an dieser Stelle deinen Tracking-Code einfügen kannst, wenn du magst. Ansonsten kannst du ihn auch einfach ignorieren, ist ja nur ein Kommentar.

Mit der Wordpress-Funktion **get_footer();** binden wir die footer.php in unserer index.php ein:

```
<?php get_header(); ?>

   <div id="main">

      <?php if (have_posts()) : while (have_posts()) : the_post(); ?>
         <h2><a href="<?php the_permalink() ?>"><?php the_title(); ?></a></h2>
         <div class="entry">
            <?php the_content(); ?>
         </div>
      <?php endwhile; endif; ?>

   </div><!-- main -->

   <div id="sidebar"></div><!-- sidebar -->

<?php get_footer(); ?>
```

Im nächsten Teil werden wir endlich unser Menü auf der rechten Seite mit Leben füllen. Dafür kommt die **Template-Datei sidebar.php** wie gerufen.

2.3. sidebar.php – das Menü mit Leben füllen

Nun wollen wir mal ein paar typische Wordpress Elemente in den noch leeren Bereich auf der rechten Seite bringen. Ein Standard-Theme platziert in der Sidebar unter anderem:

- die statischen Seiten im Blog (Impressum, About, etc.)
- Kategorien
- Archiv (bsp.: Januar 2010, Dezember 2009, November 2009, etc.)
- Meta-Daten (Login/Logout, RSS, etc.)

Da ich hier das Rad nicht neuerfinden, sondern dir lediglich die Grundlagen aufzeigen möchte, halte ich mich an die Standard-Sidebar. Fühl dich aber frei deine sidebar.php nach deinen Bedürfnissen anzupassen. Ich für meinen Teil, lasse in der Regel den Meta-Part weg und füge stattdessen eine Liste der 5 neuesten Kommentare oder der letzten 5 Artikel ein.

sidebar.php und Inhalte

Lege als erstes die Datei **sidebar.php** in deinem Theme-Ordner an. Nun wechsel in die index.php und nutze die funktion “get_sidebar()” um die Sidebar einzubinden:

index.php

```
<?php get_header(); ?>

   <div id="main">

      <?php if (have_posts()) : while (have_posts()) : the_post(); ?>
         <h2><a href="<?php the_permalink() ?>"><?php the_title(); ?></a></h2>
```

```
            <div class="entry">
                <?php the_content(); ?>
            </div>
        <?php endwhile; endif; ?>

    </div><!-- main -->

    <div id="sidebar">
        <?php get_sidebar(); ?>
    </div><!-- sidebar -->

<?php get_footer(); ?>
```

Die Sidebar wird nun verwendet, aber bislang natürlich noch nichts angezeigt. Damit sich das ändert, werden wir als erstes mal ein “mini About” verfassen, also wer wir sind und worum es in unserem Blog geht.

sidebar.php

```
<h2>&Uuml;ber dieses Blog</h2>
<p class="mini_about">
    Hallo, mein Name ist Christian und du siehst hier das Tutorial Template aus dem <a
href="http://wordpress.lernenhoch2.de/handbuch/">Wordpress Template Tutorial</a> auf <a
href="http://lernenhoch2.de/">Lernen²</a>
</p>
```

Du solltest nun in etwa folgendes erhalten:

*(ich habe zum testen wieder den **Lernen² Blog** verwendet und werde das auch für die zukünftigen Screenshots in diesem Tutorial)*

So, das war ja bislang noch nichts dynamisches, aber zumindest funktioniert die Sidebar und wird korrekt angezeigt. Als nächstes wollen wir uns die Kategorien unseres Blogs anzeigen lassen:

```
<h2>&Uuml;ber dieses Blog</h2>
<p class="mini_about">
    Hallo, mein Name ist Christian und du siehst hier das Tutorial Template aus dem <a
href="http://wordpress.lernenhoch2.de/handbuch/">Wordpress Template Tutorial</a> auf <a
href="http://lernenhoch2.de/">Lernen²</a>
</p>

<h2>Kategorien</h2>
<ul>
    <?php wp_list_categories('orderby=name&order=ASC&title_li='); ?>
</ul>
```

Wordpress bietet uns einige mächtige Template-Funktionen, die wir fast nach Belieben einsetzen können. “wp_list_categories()” ist eine davon und über die Parameter können wir die Ausgabe steuern. Eine Liste aller Parameter und welche Werte akzeptiert werden, findest du hier: **wp_list_categories()** oder auf deutsch: **wp_list_categories() deutsch**.

In unserem Beispiel sagen wir Wordpress: Gib uns eine Liste mit allen Kategorien, sortiere die Liste anhand der Namen (“orderby=name”) und sortiere sie aufsteigend (“order=ASC” – also von a-z). Wir können auch das selbe erreichen, indem wir wp_list_categories() komplett leerlassen, denn die DEFAULT-Werte für diese Funktion sind die selben. Wenn wir den Parameter “title_li” nicht auf “leer” setzen, wird uns ein “Categories”

oben an die Liste geklatscht.

Du kannst nun ein bisschen mit dieser Funktion rumspielen:

- Versuche die Reihenfolge von aufsteigend, zu absteigend (DESC) zu ändern
- Versuche eine Kategorie auszuschließen
- Wenn du mehr als 5 Kategorien hast, versuche die Liste auf 5 Kategorien zu beschränken
- Versuche die Liste nach der Anzahl an Beiträgen pro Kategorie zu sortieren. Die Kategorie mit den meisten Beiträgen soll oben stehen, die mit den wenigsten, unten.

Man kann zwar nicht alles, aber sehr sehr viel mit den Wordpress Template-Funktionen erreichen. Wenn dir also die Standard-Ausgabe nicht gefällt/genügt, dann hilft meist ein kurzer Blick in die Dokumentation um das gewünschte Ergebnis zu erzielen.

sidebar.php – Archiv anzeigen

Das Archiv ist eine typische Erkennungsmarke für einen Blog. Meist wird eine Liste der letzten 10-20 Monate aufgelistet, chronologisch absteigend. Das wollen wir auch mal probieren:

```
01 <h2>&Uuml;ber dieses Blog</h2>
02 <p class="mini_about">
03    Hallo, mein Name ist Christian und du siehst hier das Tutorial Template aus dem <a
   href="http://wordpress.lernenhoch2.de/handbuch/">Wordpress Template Tutorial</a> auf <a
   href="http://lernenhoch2.de/">Lernen²</a>
04 </p>
05
06 <h2>Kategorien</h2>
07 <ul>
08    <?php wp_list_categories('orderby=name&order=ASC&title_li='); ?>
09 </ul>
10
11 <h2>Archiv</h2>
12 <ul>
13    <?php wp_get_archives('type=monthly'); ?>
14 </ul>
```

Mit **wp_get_archives()** (oder **wp_get_archives()** de) können wir uns eine Liste des Archivs ausgeben lassen. Die Default-Werte geben ein Monats-Archiv aus, man kann aber auch ein Wochen oder sogar Tages-Archiv anzeigen lassen (das macht aber nur bei Blogs Sinn, die mehrere Artikel täglich haben, meiner Meinung nach).

Die Wordpress Blogroll

Ein weiteres, wesentliches Element eines Wordpress Themes, ist die Blogroll. In der Blogroll werden Blogs verlinkt, die du im Wordpress Backend einträgst.

```
01 <h2>&Uuml;ber dieses Blog</h2>
02 <p class="mini_about">
03    Hallo, mein Name ist Christian und du siehst hier das Tutorial Template aus dem <a
   href="http://wordpress.lernenhoch2.de/handbuch/">Wordpress Template Tutorial</a> auf <a
   href="http://lernenhoch2.de/">Lernen²</a>
04 </p>
```

```
05
06 <h2>Kategorien</h2>
07 <ul>
08    <?php wp_list_categories('orderby=name&order=ASC&title_li='); ?>
09 </ul>
10
11 <h2>Archiv</h2>
12 <ul>
13    <?php wp_get_archives('type=monthly'); ?>
14 </ul>
15
16 <h2>Blogroll</h2>
17 <ul>
18    <?php get_links(-1, '<li>', '</li>', '', FALSE, 'name', FALSE, FALSE, -1, FALSE); ?>
19 </ul>
```

wp_list_bookmarks() (oder **wp_list_bookmarks() de**)

Wir werden uns später noch weiter mit der Sidebar.php befassen, doch das sollte vorerst reichen. Wenn du magst, kannst du dir hier weitere Wordpress-Template-Funktionen anschauen und dich für deine Sidebar inspirieren lassen: **Wordpress Template Tags** (und **auf deutsch**)

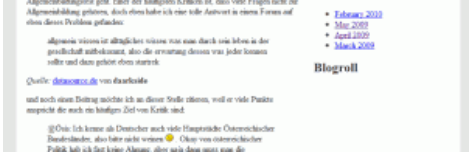
2.4. Logo und Tagline (Spruch)

Jetzt wollen wir endlich mal den leeren Bereich oben im Template füllen. Laut unserem Wireframe soll oben auf der Seite das Logo, bzw. der Name der Website + ein passender Spruch angezeigt werden. Diese Daten können im Wordpress Backend unter “Settings -> General” eingestellt werden und betreffen den “Blog Title” und die “Tagline”. Wenn diese Felder ausgefüllt sind, können wir mit der Wordpress-Template Funktion “bloginfo()” darauf zugreifen:

header.php

```
01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//DE" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
02 <html xmlns="http://www.w3.org/1999/xhtml" lang="de">
03 <head profile="http://gmpg.org/xfn/11">
04
05    <meta http-equiv="Content-Type" content="<?php bloginfo('html_type'); ?>; charset=<?php bloginfo('charset'); ?>" />
06
07    <title><?php wp_title(' - ', true, 'right'); ?> <?php bloginfo('name'); ?></title>
08
09    <link rel="stylesheet" href="<?php bloginfo('stylesheet_url'); ?>" type="text/css" media="screen" />
10    <link rel="pingback" href="<?php bloginfo('pingback_url'); ?>" />
11
12    <?php wp_head(); ?>
13
14 </head>
15 <body>
16
17 <div id="wrapper">
18
19    <div id="header">
20       <h1><?php bloginfo('name'); ?></h1>
21       <h3><?php bloginfo('description'); ?></h3>
22    </div><!-- header -->
```

Im h1-Tag zeigen wir den Blogtitel an, dieser wird über den Parameter-Wert “name” ausgegeben. Die Tagline, also der Spruch des Blogs, erhält man mit dem Parameter-Wert “description”.



Zugegeben, das sieht nicht besonders schick aus, aber darum gehts hier ja erstmal nicht. Durch eine kleine Modifikation wird unser Titel gleichzeitig zu einem Link zurück zur Startseite des Blogs:

```
<h1><a href="<?php bloginfo('url'); ?>"><?php bloginfo('name'); ?></a></h1>
```

Jetzt nehmen wir noch eine simple Änderung an unserem Stylesheet vor, um eine Trennlinie zwischen dem **header** und dem **main** Bereich zu erhalten:

style.css

```
/*
Theme Name: Tutorial
Theme URI: http://wordpress.lernenhoch2.de
Description: Template zum Wordpress Tutorial auf http://wordpress.lernenhoch2.de
Author: Christian Strang
Author URI: http://lernenhoch2.de
*/

body        {margin: 0px; padding: 0px; color: #333; background: #e5e5e5;}

#wrapper    {width: 850px; margin: auto; text-align: left; background: #fff;}
#header        {height: 160px; padding: 20px; border-bottom: 1px solid #ccc;}
#main       {width: 520px; padding: 20px; float: left;}
#sidebar    {width: 270px; padding: 10px; padding-top: 20px; float: left;}
#footer        {clear: both; height: 100px; padding: 20px; }
```

Das Ergebnis sollte nun folgendermaßen aussehen:

Im nächsten Teil werden wir…

2.5. single.php – Template für Beiträge/Posts

Bevor wir die single.php anlegen, schau dir nochmal die **Wordpress Template Hierarchie** an. Anfangs habe ich schon erwähnt, dass die Dateien index.php + style.css zusammen schon ein voll funktionsfähiges Template ausmachen. Möchte man das Template aber spezialisieren, braucht man weitere Template-Dateien, in diesem Fall die **single.php**.

Wofür ist die single.php

Wenn Wordpress einen Einzel-Beitrag anzeigen soll, schaut es im Template-Ordner erstmal nach der Datei “single.php”. Wenn er sie findet, benutzt er diese Template-Datei um die Seite anzuzeigen, findet er sie nicht, greift er auf die “index.php” zur Anzeige zurück. Dies ist bislang bei uns so der Fall, doch das wollen wir nun ändern.

Erzeuge die Datei “single.php” im Template-Ordner und kopiere den kompletten Code aus der index.php und füge ihn in die single.php ein (kopieren **nicht** ausschneiden):

single.php

```php
<?php get_header(); ?>

   <div id="main">

      <?php if (have_posts()) : while (have_posts()) : the_post(); ?>
         <h2><a href="<?php the_permalink() ?>"><?php the_title(); ?></a></h2>
         <div class="entry">
            <?php the_content(); ?>
         </div>
      <?php endwhile; endif; ?>

   </div><!-- main -->

   <div id="sidebar">
      <?php get_sidebar(); ?>
   </div><!-- sidebar -->

<?php get_footer(); ?>
```

Nun sollten wir zwei Dateien mit identischem Code haben. Ab jetzt nehmen wir nur noch Änderungen an der “single.php” vor. Als erstes wollen wir unterhalb des Beitrag-Titels den Namen des Autor, das Datum der Veröffentlichung und die Kategorie des Beitrags anzeigen:

```php
<?php get_header(); ?>

   <div id="main">

      <?php if (have_posts()) : while (have_posts()) : the_post(); ?>
         <h2><a href="<?php the_permalink() ?>"><?php the_title(); ?></a></h2>
         <div id="meta">
            erstellt am: <?php the_date('d.m.Y'); ?> |
            von: <?php the_author(); ?> |
            Kategorie(n): <?php the_category(', '); ?></p>
         </div>
         <div class="entry">
            <?php the_content(); ?>
         </div>
      <?php endwhile; endif; ?>

   </div><!-- main -->

   <div id="sidebar">
      <?php get_sidebar(); ?>
   </div><!-- sidebar -->

<?php get_footer(); ?>
```

Im Grunde greife ich einfach nur über die **Template Tags** auf die Werte zu, die ich brauche. **the_date()** gibt mir das Datum aus, wann der Artikel erstellt wurde (es gibt einige Formatierungsmöglichkeiten, man muss lediglich den Parameter-Wert anpassen). Als nächstes gebe ich mit “the_author()” den Autor des Beitrags aus und mit “the_category(‘, ‘)” folgt eine kommaseparierte Liste der Kategorien, mit denen der Beitrag assoziiert ist. Alles in allem recht einfach, du überlegst welchen Wert du anzeigen möchtest, schaust nach, welche Funktion dafür zuständig ist und passt dir das Ergebnis mit den nötigen Parameter-Werten an, *easy as pie*.

2.6. page.php – Template für Seiten/Pages

Die Template-Datei “page.php” funktioniert im Grunde genau wie “single.php” doch anstelle für die Beitrags-Seite, wird die “page.php” für die Anzeige von Seiten verwendet. Wie du weißt, kannst du in Wordpress statische Seiten erstellen, die losgelöst von der chronologischen Reihenfolge existieren. Das ist praktisch für Seiten, die ständig zugänglich sein sollen und sich eher selten verändern und zudem in der Regel keine Kommentare zulassen. Dies sind unter anderem das “Impressum” oder auch die “About me” Seite.

Wenn du also deinen Seiten ein anderes Layout als den Beiträgen oder der index.php geben möchtest, kannst du dies mit der page.php umsetzen. Ich werde aber hier nicht weiter darauf eingehen, da es nicht viel neues zu lernen gibt. Nur als kleiner Tipp:

- Seiten kommen in der Regel ohne einen Verweis auf Autor und Datum aus
- Achte darauf, dass das Kommentarformular (was wir bislang noch garnicht erstellt haben) nicht angezeigt wird, ausser du möchtest, dass deine Besucher auch Seiten kommentieren können

---

2.7. comments.php – Kommentare schreiben und anzeigen

Kein Blog kommt ohne vernünftige Kommentarfunktion aus, deshalb soll auch unser Template mit einem Kommentarformular ausgestattet werden. Lege als erstes die Datei **comments.php** in deinem Template-Ordner an.

Das Kommentar-Formular anlegen

In die comments.php fügen wir nun folgenden Code ein:

```
<div id="kommentar_formular">

<h3 id="respond">Kommentar schreiben</h3>

<form action="<?php echo get_option('siteurl'); ?>/wp-comments-post.php" method="post"
id="commentform">

   <p>
      <input type="text" name="author" id="author" value="<?php echo $comment_author; ?>" size="22" />
      <label for="author">Name</label>
   </p>

   <p>
      <input type="text" name="email" id="email" value="<?php echo $comment_author_email; ?>" size="22"
tabindex="2" />
      <label for="email">Email <small>(wird nicht ver&ouml;ffentlicht)</small></label>
   </p>

   <p>
      <input type="text" name="url" id="url" value="<?php echo $comment_author_url; ?>" size="22"
tabindex="3" />
      <label for="url">Webseite</label>
   </p>

   <p>
      Kommentar
      <textarea name="comment" id="comment" style="width: 100%;" rows="10" tabindex="4"></textarea>
   </p>

   <p>
      <input name="submit" type="submit" id="submit" tabindex="5" value="Kommentar abschicken" />
      <input type="hidden" name="comment_post_ID" value="<?php echo $id; ?>" />
   </p>

   <?php do_action('comment_form', $post->ID); ?>

</form>

</div> <!-- kommentar_formular -->
```

Erklärung:
In die Div-Box “kommentar_formular” kommt unser kompletter Code der für das Formular nötig ist.

```
<form action="<?php echo get_option('siteurl'); ?>/wp-comments-post.php" method="post"
```

```
id="commentform">
```

Das “action”-Attribut des Formulars erwartet den Link zur Datei “wp-comments-post.php”, die sich im root-Ordner des Blogs befindet (schau nach :)). Diese Datei erwartet ein paar Werte und trägt den Kommentar ein, wenn alles in Ordnung ist. “get_option(’siteurl’);” gibt uns die Blog-Root Url, daran hängen wir noch den Namen der Datei und der Formular-Kopf ist fertig.

Als nächstes erstellen wir die Input-Felder für die Werte “author”, “email” und “url”, sowie eine Textarea für den “comment”. Je nach Einstellung im Wordpress Backend erwartet die Datei “wp-comments-post.php” diese Werte. Die Value-Werte von author, email und url sind jeweils mit einer Variable ausgestattet:

```
<?php echo $comment_author; ?>
<?php echo $comment_author_email; ?>
<?php echo $comment_author_url; ?>
```

Wenn ein Besucher schonmal kommentiert hat und seine Werte noch vorhanden sind, werden die Formular-Felder vorausgefüllt, was dem Besucher ein bisschen Arbeit erspart.

Die beiden letzten Input-Elemente sind der Submit-Button und ein verstecktes Feld.

```
<input name="submit" type="submit" id="submit" tabindex="5" value="Kommentar abschicken" />
<input type="hidden" name="comment_post_ID" value="<?php echo $id; ?>" />
```

Der Submit-Button ist soweit selbsterklärend und das hidden-Feld erwartet die ID des aktuellen Beitrags, ebenfalls wichtig für die “wp-comments-post.php”-Datei.

```
<?php do_action('comment_form', $post->ID); ?>
```

Bei der letzten Zeile handelt es sich mal wieder um ein **Plugin-API-Hook**, sprich es gibt Plugins, die diesen Hook brauchen um korrekt zu funktionieren. Also, einfach rein und glücklich sein ;)

Kommentare ausgeben

Jetzt haben wir zwar ein Formular, über das der Besucher seinen Kommentar schreiben kann, aber noch keine Möglichkeit die Kommentare anzuzeigen. Dazu gibt es, ähnlich wie bei den Beiträgen, eine Loop, die Kommentar Loop. Füge unterhalb des Kommentarformulars in der Datei **comments.php** den folgenden, markierten Code ein:

```
</form>

</div> <!-- kommentar_formular -->

<div id="kommentare">
   <?php foreach ($comments as $comment) : ?>

      <div class="comment" id="comment-<?php comment_ID() ?>">

         <small class="commentmetadata"><?php comment_author_link() ?> <strong>|</strong> am <?php comment_date('j. F Y') ?> um <?php comment_time('H:i') ?> Uhr</small>

         <?php comment_text() ?>

          <?php if ($comment->comment_approved == '0') : ?>
            <strong>Achtung: Dein Kommentar muss erst noch freigegeben werden.</strong><br />
         <?php endif; ?>

      </div>
   <?php endforeach; /* end for each comment */ ?>
</div><!-- kommentare -->
```

Der Code ist schnell erklärt: Wir wollen alle Kommentare in der Div-Box “kommentare” ablegen. Danach startet die Comment-Loop mit:

```
<?php foreach ($comments as $comment) : ?>
```

Jeder Kommentar erhält seine eigene Div-Box mit der Klasse “comment”. Da wir uns in der Loop befinden, können wir auf alle Kommentardaten des jeweiligen Kommentars zugreifen. Wir brauchen

- **den Autor** – “comment_author_link()”, das ist der Name + der Link, wenn der User eine URL beim eintragen mit angegeben hat
- **das Datum und die Uhrzeit** – “comment_date(‘j. F Y’)” und “comment_time(‘H:i’)”
- **den Kommentar** – “comment_text()”

Der letzte Teil ist eine Anzeige-Prüfung. Wenn der Kommentar erst von einem Admin freigeschaltet werden muss, bekommt der User eine Nachricht und weiß, dass sein Eintrag erfolgreich war.

Fazit – mehr zur comments.php

Die hier erklärte comments.php ist die absolute Basis. Es erfolgen so gut wie keine Überprüfungen, also ob nur registrierte Besucher kommentieren dürfen, ob der Beitrag passwortgeschützt ist und demnach wahrscheinlich auch keine Kommentare ohne Passwort angezeigt werden sollen und viele weitere. Zudem werden keine Gravatare angezeigt (User-Bilder neben dem Kommentar) und auch kein Captcha verwendet (wobei das eh besser durch ein Plugin gelöst werden sollte).

Ich denke aber, dass diese Basis besser ist als ein Beispiel was alles kann, denn so kann man selbst entscheiden welche Features man noch haben möchte. In den **Template Tags** gibt es einen extra Bereich für die **Comment Tags**, für alle die mehr aus der comments.php rausholen wollen.

2.7.1. comments.php nicht direkt aufrufen

2.7.2. Passwort geschützte Seiten

2.7.3. Gravatare anzeigen

2.8. 404.php – Template für den 404 – Seite nicht gefunden – Fehler

Es gibt in Wordpress auch ein extra Fehler-Template, die Datei **404.php**. Wenn mal ein Link nicht zum Ziel führt, ein Artikel gelöscht wurde oder die URL eines Artikels sich verändert hat, bekommt der Benutzer als Antwort den 404 – Seite nicht gefunden – Fehler.

Die 404.php funktioniert genau wie die index.php, single.php, page.php etc. Binde einfach deine Standardelemente ein (also get_header(), get_footer(), get_sidebar, usw.) und nutze den Content-Bereich um den Besucher auf die fehlende Seite hinzuweisen. Ausserdem empfiehlt es sich:

- eine eMail-Adresse anzugeben *(so können dich die Besucher auf fehlende Seiten hinweisen)*
- eine Liste mit deinen Top 10 Artikeln *(für Besucher die über Suchmaschinen kommen und so vielleicht doch noch was länger auf deinem Blog verweisen)*
- Die Suche nochmal prominent anzuzeigen, falls der Beitrag einfach nur eine andere URL hat, kann der Besucher so vielleicht trotzdem noch zum Ziel kommen

Wir beschränken uns hier nur auf den Hinweis, dass die Seite nicht gefunden wurde. Die restlichen Dinge werden meist über Plugins gelöst (related posts, top10 posts, etc.) und wären an dieser Stelle zu speziell:

404.php

```
01 <?php get_header(); ?>
02
03    <div id="main">
04       <h2>404 - Seite nicht gefunden</h2>
05       <p>Tut mir leid, aber die Seite nach der du gesucht hast, wurde leider nicht gefunden. Schreib
   mir eine <a href="mailto:name@email.de?subject=<?php echo get_bloginfo('name'); ?> - 404 Seite nicht
   gefunden">eMail</a>, dann helf ich dir weiter.</p>
06    </div><!-- main -->
07
08    <div id="sidebar">
09       <?php get_sidebar(); ?>
```

```
10      </div><!-- sidebar -->
11
12  <?php get_footer(); ?>
```

An sich alles selbsterklärend, aber vielleicht ein kurzer Kommentar zu **Zeile 5**:
Wir geben dem Besucher unsere Email-Adresse, damit er uns auf den Fehler hinweisen kann. Ausserdem fügen wir in den Betreff (Subject) der eMail den Namen unseres Blogs ein, so können wir schnell erkennen, in welchem Blog der 404-Fehler aufgetreten ist.

Auf Smashing Magazine gibt es eine Auflistung kreativer 404 Seiten, als Inspirationsquelle durchaus sehenswürdig.

- **Wanted: Your 404 Error Pages**
- **404 Error Pages: Reloaded**
- **404 Error Pages, One More Time**

Andererseits sollte man eher weniger Zeit in die 404.php stecken und mehr darauf verwenden, solche Fehler zu vermeiden :)

2.9. search.php – unsere eigene, kleine Blogsuche

Als nächstes wollen wir eine kleine Suche für unseren Blog bauen. Dazu werden wir als erstes ein Suchformular in unserer sidebar.php einbauen:

```
01  <h2>Suche</h2>
02  <p>
03  <form method="get" id="searchform" action="<?php echo $_SERVER['PHP_SELF']; ?>">
04      <input type="text" value="<?php echo wp_specialchars($s, 1); ?>" name="s" id="s" />
05      <input type="submit" id="search_submit" value="Suchen" />
06  </form>
07  </p>
08
09  <h2>&Uuml;ber dieses Blog</h2>
10  <p class="mini_about">
11      Hallo, mein Name ist Christian und du siehst hier das Tutorial Template aus dem <a
    href="http://wordpress.lernenhoch2.de/handbuch/">Wordpress Template Tutorial</a> auf <a
    href="http://lernenhoch2.de/">Lernen²</a>
12  </p>
13
14  <h2>Kategorien</h2>
15  <ul>
16      <?php wp_list_categories('orderby=name&order=ASC&title_li='); ?>
17  </ul>
18
19  <h2>Archiv</h2>
20  <ul>
21      <?php wp_get_archives('type=monthly'); ?>
22  </ul>
23
24  <h2>Blogroll</h2>
25  <ul>
26      <?php wp_list_bookmarks(); ?>
27  </ul>
```

Dieses Suchformular reicht theoretisch schon aus, um die Ergebnisse anzuzeigen, denn Wordpress nutzt dafür einfach die Fallback-Template-Datei **index.php**. Aber wir wollen dem Besucher auch anzeigen, dass seine Suche ausgeführt wurde (und für welche Eingabe die Suche war). Deshalb legen wir jetzt die Datei **search.php** an:

search.php

```php
<?php get_header(); ?>

   <div id="main">
      <?php if (have_posts()) : ?>
         <p class="info">Deine Suchergebnisse f&uuml;r <strong><?php echo $s ?></strong></p>

         <?php while (have_posts()) : the_post(); ?>
            <h2><a href="<?php the_permalink() ?>"><?php the_title(); ?></a></h2>
            <div class="entry">
               <?php the_content(); ?>
            </div>
         <?php endwhile; ?>

         <p align="center"><?php next_posts_link('&laquo; &Auml;ltere Eintr&auml;ge') ?> | <?php previous_posts_link('Neuere Eintr&auml;ge &raquo;') ?></p>

      <?php else : ?>
         <h2>Leider nichts gefunden</h2>

      <?php endif; ?>
   </div><!-- main -->

   <div id="sidebar">
      <?php get_sidebar(); ?>
   </div><!-- sidebar -->

<?php get_footer(); ?>
```

Erklärung:
Die Search.php unterscheidet sich nicht sonderlich von der index.php mit einer entscheidenden Ausnahme: Wir prüfen ob für die Suchphrase überhaupt Artikel gefunden wurden (Zeile 4), ist das nicht der Fall wird der Besucher in Zeile 17 darauf hingewiesen. Ausserdem geben wir in Zeile 5 die Suchphrase aus, die wir in der Variable “$s” abgelegt haben. Zeile 14 ist dazu da, einen “Vor-” und “Zurück”-Link auszugeben, damit man mehr als 10 Artikel (oder wieviele im Backend eingetragen wurden) anschauen kann.

Natürlich kann man mit der Suchergebnisseite noch viel mehr anstellen, z.b. häufige oder ähnliche Suchen anzeigen (nur über Plugins) oder Tags ausgeben, die der Suchphrase entsprechen. Doch für die meisten Blogger dürfte unsere Funktionalität ausreichen.

---

3. Schritt 3 – Template optimieren

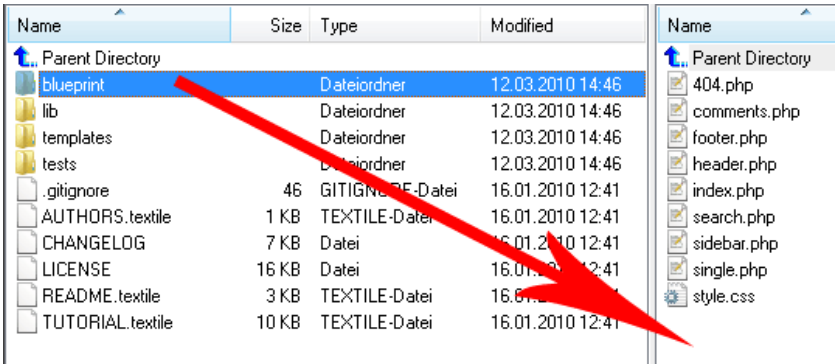
Die Funktion unseres Template, bzw. die einzelnen Template Dateien, haben wir in Schritt 2 behandelt. Auf dieser Grundlage kann jeder ein Basis-Template in Wordpress bauen und dieses dann an seine Bedürfnisse anpassen. In Schritt 3 werden wir nun unser Tutorial Template optimieren.

3.1. Blueprint CSS – Grundformatierung und Typography

Unser Basis Template soll zwar schlicht sein, damit es jeder als Grundlage für sein eigenes Template nutzen kann, doch wir können mit einem kleinen Handgriff noch etwas mehr aus diesem schlichten Template rausholen. Der Trick ist, dass wir einfach das CSS Framework **Blueprint CSS** in unser Template einbinden. Dieses gibt uns eine saubere Grundformatierung, die in allen Browsern gleich aussieht.

1. Schritt – Blueprint CSS runterladen & installieren

Als erstes musst du dir das Blueprint CSS Framework runterladen: **Download**. Als nächstes entpacke die Datei und lade den Ordner “blueprint” in den Tutorial Template Ordner:



2. Schritt – Stylesheets einbinden

Damit unser Tutorial Template die Stylesheets auch verwendet, müssen wir sie noch in der header.php einbinden. Dazu öffne die Datei “header.php” und füge den markierten Code ein:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//DE" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="de">
<head profile="http://gmpg.org/xfn/11">

   <meta http-equiv="Content-Type" content="<?php bloginfo('html_type'); ?>; charset=<?php bloginfo('charset'); ?>" />

   <title><?php wp_title(' - ', true, 'right'); ?> <?php bloginfo('name'); ?></title>

   <link rel="stylesheet" href="<?php bloginfo('template_directory'); ?>/blueprint/screen.css" type="text/css" media="screen, projection">
   <link rel="stylesheet" href="<?php bloginfo('template_directory'); ?>/blueprint/print.css" type="text/css" media="print">
   <!--[if lt IE 8]><link rel="stylesheet" href="<?php bloginfo('template_directory'); ?>/blueprint/ie.css" type="text/css" media="screen, projection"><![endif]-->

   <link rel="stylesheet" href="<?php bloginfo('stylesheet_url'); ?>" type="text/css" media="screen" />
   <link rel="pingback" href="<?php bloginfo('pingback_url'); ?>" />

   <?php wp_head(); ?>

</head>
<body>

<div id="wrapper">

   <div id="header">
      <h1><a href="<?php bloginfo('url'); ?>"><?php bloginfo('name'); ?></a></h1>
      <h3><?php bloginfo('description'); ?></h3>
   </div><!-- header -->
```

Damit sollte das Template nun optisch was besser aussehen und trotzdem ein leicht anpassbares Basis Template bleiben.

Vorher:

Nachher:

Wer daran interessiert ist, kann sich hier das **Quickstart Tutorial für Blueprint CSS** anschauen.

3.2. header.php Title-Tag optimieren

3.3. Sidebar Widget-fähig machen

3.4. Template Screenshot

für Template-Vorschau