


Diese Leseprobe haben Sie beim
 edv-buchversand.de heruntergeladen.
Das Buch können Sie online in unserem
Shop bestellen.

[Hier zum Shop](#)

Kapitel 1

Erste Schritte im Entwicklungsprozess

WordPress ist mittlerweile das beliebteste Content-Management-System der Welt, nachdem es lange als einfache oder gar unsaubere Blogging-Software abgetan wurde, mit der nicht wirklich ernsthafte Webprojekte umgesetzt werden können.

Dass WordPress nicht nur ein Blogging-System ist, beweist die Tatsache, dass viele namhafte Größen WordPress einsetzen. Im Vorwort hatte ich das Lifestylemagazin *Vogue* erwähnt. Dort kommt WordPress *headless* (dt. »kopflös«) zum Einsatz. Das heißt, die Kontrolle läuft zwar über WordPress, die Ausgabe des Inhalts aber über ein anderes System. Bei *Vogue* wurde das WordPress-Frontend durch eine Web-App mit *ReactJS* ersetzt,¹ die sich die darzustellenden Daten über eine REST-Schnittstelle von WordPress holt.

Es gibt also viele Einsatzzwecke für WordPress. Viele davon werde ich in diesem Buch ansprechen. Aber so komplex soll es jetzt erst einmal nicht werden. Wir fangen zunächst damit an, WordPress von Grund auf kennenzulernen. Und zwar aus Sicht eines Entwicklers bzw. einer Entwicklerin.

Modernes PHP in WordPress?

Wenn ich mit anderen Entwicklern spreche, höre ich sehr oft, dass PHP keine richtige Programmiersprache sei. Dementsprechend sei WordPress (WP) ein einziges Desaster. Es stimmt, dass WP in den alljährlichen Entwickler-Umfragen der Website *Stack Overflow* regelmäßig zur unbeliebtesten Plattform gewählt wurde. Das lag nicht zuletzt daran, dass PHP selbst in der Rangliste der unbeliebtesten Sprachen immer weiter nach oben kletterte.

Die Unbeliebtheit kommt aber aus einer anderen Ecke. In einem Beitrag des PHP-Magazins schrieb ich schon 2019, dass im Internet immer wieder dieselbe Aussage zu lesen ist: »Dadurch, dass PHP so einfach zu erlernen sei (die Einstiegshürden sollen

¹ WordPress Website Showcase. *Vogue*. <https://wordpress.org/showcase/vogue>. Abgerufen am 28.11.2021.

eher gering sein), Sorge es dafür, dass es viele User gibt, die nur Halbwissen darüber haben. Das wiederum Sorge dafür, dass halbgarer Code im Internet verstreut werde. Und das wiederum Sorge dafür, dass dieser Code oft geteilt werde«.² Natürlich ist das viel zu kurz gedacht.

Fakt ist, dass immer noch PHP-Code im Umlauf ist, der sehr alt ist. Dieser Code ist zumeist prozedural geschrieben. Es ist allgemein bekannt, dass speziell langer prozeduraler Code meistens schwieriger zu lesen, zu verstehen und zu pflegen ist als objektorientierter Code. WordPress ist aber genauso aufgebaut. »Wir haben im Kern fast ausschließlich prozedural programmierten Code, der im Laufe der Zeit durch ein paar Klassen ergänzt wurde. Entstanden ist also genau das, was Entwickler nicht wollen (...). Eine Mischung aus funktionellem und objektorientiertem Code, der schwer zu erlernen ist, weil die Codebasis riesig ist.«, schrieb ich im Artikel weiter.

PHP bekam 2015 aber einen Entwicklungsschub. Durch Version 7.0 erhielt die Programmiersprache nicht nur deutlich mehr Performance. Diese Version ebnete ebenso den Weg, der besseren objektorientierten Code erlaubt. PHP 8.0 und höher kann nun fast alles, was eine moderne Programmiersprache können muss. Das macht alle glücklicher, die heute Websites entwickeln. Und dass es mit der Beliebtheit nun auch wieder bergauf geht, zeigt auch der Aufwärtstrend in Abbildung 1.1.

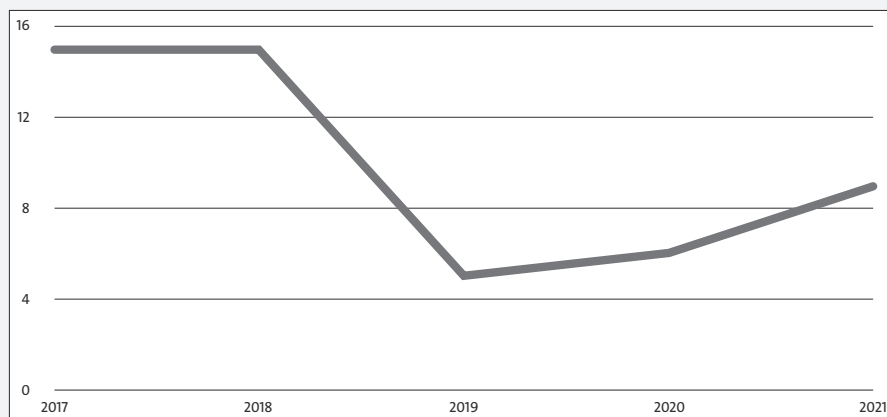


Abbildung 1.1 Der Rang von PHP in der Liste der unbeliebtesten Sprachen. Je niedriger die Position ist, desto unbeliebter ist die Sprache.³

² Vgl. Simeth, Florian: WordPress 2024: Ein Blick in die Zukunft, in PHPMagazin, Nr 5, 2019, S. 66.

³ Quellen: Developer Survey Results 2017 – 2021. <https://insights.stackoverflow.com/survey/2017>. <https://insights.stackoverflow.com/survey/2018/>. <https://insights.stackoverflow.com/survey/2019>. <https://insights.stackoverflow.com/survey/2020>. <https://insights.stackoverflow.com/survey/2021>. Abgerufen am 01.12.2021.

Nun ist es jedoch so, dass WordPress eben WordPress bleibt. Und im Kern ändert sich vorerst nur wenig, weil der Fokus unter anderem auf dem neuen Block-Editor sowie der Mehrsprachigkeit liegt. Zumindest bis 2022. Was darüber hinaus folgt, ist zu diesem Zeitpunkt noch nicht bekannt.

Es lässt sich jedoch beobachten, dass sich ganz langsam etwas tut. Mit Version 5.2 von WordPress wurde zumindest die minimale PHP-Version auf 5.6.20 angehoben. Empfohlen wird bereits Version 7.4. Für Außenstehende ist das nur ein kleiner Schritt. Bedenkt man jedoch, dass WordPress über mehr als ein Jahrzehnt hinweg immer abwärtskompatibel war, dann ist das schon beachtlich. Vergessen Sie dabei nicht, dass wir von einem riesigen Marktanteil von ungefähr 820 Millionen WordPress-Installationen reden!⁴ Aktualisierungen herauszubringen, ohne zu wissen, dass auf dem Server überhaupt eine neuere Version von PHP vorliegt, funktioniert da einfach nicht. Am Ende ist das auch ein Problem aufseiten der Webhoster, eher weniger ein Problem der WordPress-Community.

Zwar hatte ich in meinem Artikel gehofft, dass die minimale PHP-Version schon 2019 auf 7.0 aktualisiert werden könnte. Dem war aber leider nicht so. Dennoch gibt es in WordPress interessante Bewegungen. Ein Feature namens *PHP Error Protection* schützt beispielsweise seit Version 5.2 vor Totalausfällen durch einen *PHP Fatal Error*, der möglicherweise durch ein Plugin oder ein Theme verursacht wurde. Es geht also in die richtige Richtung – auch wenn die Mühlen bei WordPress, speziell wenn es um die eingesetzte PHP-Version geht, etwas langsamer mahlen.

1.1 WordPress installieren

Wie ich bereits angeführt habe, ist die Installation getreu dem Leitspruch von WordPress »Democratize Publishing«⁵ sogar für den Laien einfach. In der Regel übernimmt der Webhoster die Installation vollautomatisch über ein von ihm hinterlegtes Skript. Dieses Skript legt eine Datenbank mit entsprechenden Zugangsdaten an, holt sich die aktuelle Version von WordPress und hinterlegt diese auf einem Webserver. Die nötigen Einstellungen lassen sich in der Datei `wp_config.php` ebenfalls automatisch befüllen. Und ehe Sie sich versehen, sind Sie mit der Installation durch. Manchmal werden sogar noch ein erster Benutzer angelegt sowie ein Passwort festgelegt. Der Laie muss sich dann nur noch anmelden und kann sofort loslegen. Wie aber funktioniert das Ganze Schritt für Schritt? Das sehen wir uns im Folgenden an.

⁴ Als Ausgangswert der Berechnung diente der angezeigte Wert von www.internetlivestats.com mit Stand vom 1.12.2021 und einem WordPress-Marktanteil von 43 %.

⁵ Zu Deutsch sinngemäß in etwa »Das Publizieren allgemein einfacher machen«.

1.1.1 WordPress downloaden

Die jeweils aktuelle Version von WordPress ist unter wordpress.org/download erhältlich. Auf der Seite müssen Sie etwas nach unten scrollen, bis schließlich der Button mit der Beschriftung **DOWNLOAD WORDPRESS X.X.X** sichtbar wird. Das **X.X.X** steht dabei für die jeweils aktuelle Versionsnummer. Ein Klick darauf lädt die ZIP-Datei auf den heimischen Rechner.

Der Unterschied zwischen wordpress.com und wordpress.org

Eine immer wiederkehrende Diskussion innerhalb der WordPress-Community ist der nicht richtig erkennbare Unterschied zwischen wordpress.com und wordpress.org.

Der bereits erwähnte Mitgründer von WordPress, Matt Mullenweg, ist Geschäftsführer der *Automattic Incorporated*, eines Unternehmens, das sich auf WordPress-Hosting spezialisiert hat. Automattic darf als einziges Unternehmen weltweit das Wort »WordPress« für unternehmerische Zwecke nutzen und schaltet auch gerne TV-Werbepots im US-amerikanischen Fernsehen. Es liegt also nahe, dass es die Domain wordpress.com nutzt, um seine Dienste anzubieten. Das stößt der Community von Zeit zu Zeit sauer auf.⁶ Denn oft wird Automattic als das Unternehmen bezeichnet, das »die digitale Veröffentlichungsplattform WordPress betreibt«. So stand es in einem Artikel der New York Times, der später berichtigt wurde.⁷ Richtig ist, dass die Community die Software aufgebaut hat und auch weiterhin pflegt. Automattic vertreibt sie nur weiter.

Kontrovers ist, dass Automattic einer der größten Spendengeber für das WordPress-Projekt ist. Durch das bereits erwähnte *Five for the Future*-Programm stellt Automattic über 3.600 Arbeitsstunden pro Woche für das WordPress.org-Projekt zur Verfügung.⁸

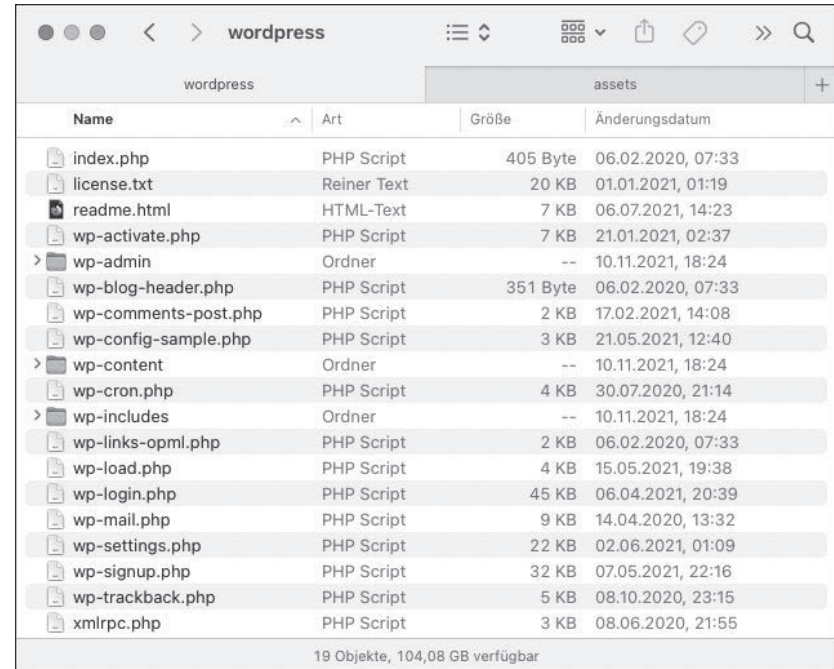
Die Website wpisnotwp.com erklärt auf Englisch noch einmal den Unterschied im Detail.

Die ZIP-Datei lässt sich mit den üblichen Programmen entpacken. Sie erhalten eine Liste mit Dateien, wie in Abbildung 1.2 zu sehen ist.

⁶ Siehe dazu auch den Beitrag von Gooding Sarah, *Automattic Updates Press Page to Clarify Distinction between WordPress.org and WordPress.com*, in: WPTavern, 2020, <https://wptavern.com/automattic-updates-press-page-to-clarify-distinction-between-wordpress-org-and-wordpress-com> (abgerufen am 01.12.2021).

⁷ Caspar Hübinger (ein Community-Mitglied) hatte auf die Ungenauigkeit in dem von der New York Times veröffentlichten Artikel hingewiesen. In dem Artikel ging es aber nicht direkt um WordPress, sondern um Matt Mullenweg als starken Befürworter der Remote-Arbeit. Der Link zum Artikel: <https://www.nytimes.com/2020/07/12/business/matt-mullenweg-automattic-corner-office.html>

⁸ WordPress.org: *Pledges (sorted by total hours)*, in WordPress, 2021, <https://wordpress.org/five-for-the-future/pledges/?order=hours> (abgerufen am 01.12.2021).



Name	Art	Größe	Änderungsdatum
index.php	PHP Script	405 Byte	06.02.2020, 07:33
license.txt	Reiner Text	20 KB	01.01.2021, 01:19
readme.html	HTML-Text	7 KB	06.07.2021, 14:23
wp-activate.php	PHP Script	7 KB	21.01.2021, 02:37
wp-admin	Ordner	--	10.11.2021, 18:24
wp-blog-header.php	PHP Script	351 Byte	06.02.2020, 07:33
wp-comments-post.php	PHP Script	2 KB	17.02.2021, 14:08
wp-config-sample.php	PHP Script	3 KB	21.05.2021, 12:40
wp-content	Ordner	--	10.11.2021, 18:24
wp-cron.php	PHP Script	4 KB	30.07.2020, 21:14
wp-includes	Ordner	--	10.11.2021, 18:24
wp-links-opml.php	PHP Script	2 KB	06.02.2020, 07:33
wp-load.php	PHP Script	4 KB	15.05.2021, 19:38
wp-login.php	PHP Script	45 KB	06.04.2021, 20:39
wp-mail.php	PHP Script	9 KB	14.04.2020, 13:32
wp-settings.php	PHP Script	22 KB	02.06.2021, 01:09
wp-signup.php	PHP Script	32 KB	07.05.2021, 22:16
wp-trackback.php	PHP Script	5 KB	08.10.2020, 23:15
xmlrpc.php	PHP Script	3 KB	08.06.2020, 21:55

Abbildung 1.2 Die Liste der Dateien und Verzeichnisse einer WordPress-Installation

1.1.2 Der Upload zum Webhosting-Anbieter

Die gesamte Palette an Dateien, die Sie nun vorliegen haben, müssen Sie (zum Beispiel per FTP) auf einen Server des Webhosting-Anbieters hochladen. Ich werde hier nicht anführen, wie das im Einzelnen funktioniert, da sich der Vorgang von Anbieter zu Anbieter unterscheidet. Nehmen Sie gegebenenfalls Kontakt mit dem jeweiligen Anbieter auf, um die nötigen Schritte in Erfahrung zu bringen.

1.1.3 Den Datenbank-Zugang einrichten

Vom Webhoster erhalten Sie ebenfalls die nötigen Zugangsdaten für die Datenbank. Oft lassen sich neue Datenbanken aber auch über die Weboberfläche des Anbieters einrichten. Sie benötigen folgende Informationen:

- ▶ Datenbank-Benutzername
- ▶ Datenbank-Passwort
- ▶ Datenbank-Name
- ▶ Server/Hostname
- ▶ Portnummer

WordPress empfiehlt derzeit MySQL in der Version 5.6 oder MariaDB 10.1 oder höher.

1.1.4 Die 5-Minuten-Installation

Die oben genannten Informationen lassen sich sicher innerhalb weniger Minuten zusammentragen. Was nun noch fehlt, ist der Aufruf der Domain, die vom Webhoster zugewiesen (oder die bestellt) wurde. Ich benutze hier als Beispiel die Domain *mein-wp.test*.

Im Browser geben Sie die URL *mein-wp.test* ein. Sie werden sofort bemerken, dass Sie zur Datei */wp-admin/setup-config.php* umgeleitet werden. Der erste Schritt der Installation beginnt mit der Auswahl der Sprache.

Da WordPress ohne Sprachpakete in Umlauf gebracht wird, müssen diese nachgeladen werden. Dies geschieht vollautomatisch im Hintergrund, wenn Sie die entsprechende Sprache (Deutsch) auswählen und auf FORTFAHREN klicken (siehe Abbildung 1.3).

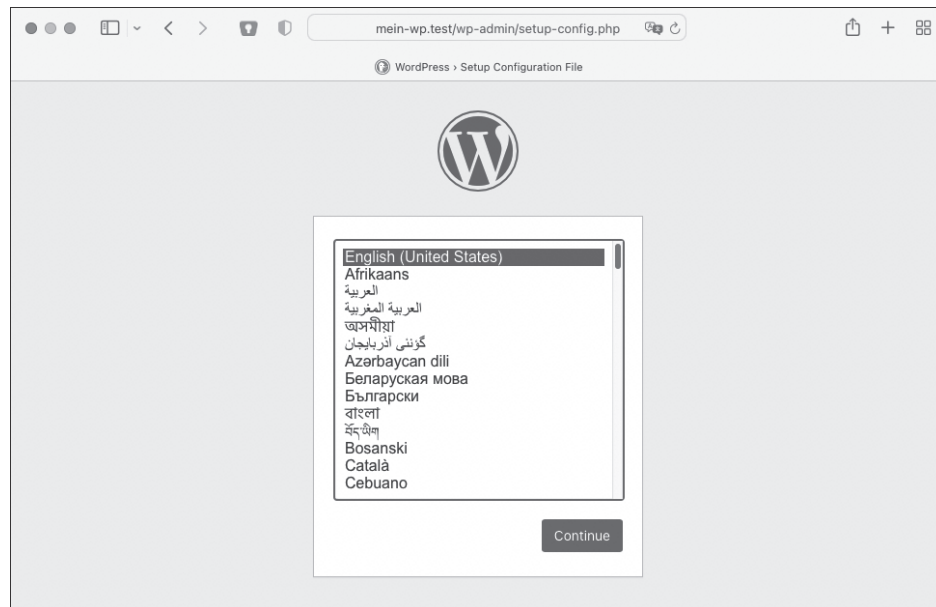


Abbildung 1.3 Erster Schritt in der WordPress-Installation: Die Auswahl der Sprache.

Welche Sprache soll ich wählen?

Allein für Deutsch haben Sie fünf verschiedene Auswahlmöglichkeiten:

- ▶ Deutsch (Österreich)
- ▶ Deutsch (Schweiz, Du)
- ▶ Deutsch (Schweiz)
- ▶ Deutsch
- ▶ Deutsch (Sie)

Die Hauptsprache in WordPress ist Englisch. Das bedeutet, dass zum Beispiel alle Plugins im Plugin-Verzeichnis von WordPress mindestens auf Englisch verfügbar sind. Alle anderen Sprachen müssen von den Übersetzern gesondert eingepflegt werden. In der Regel wird dann zuerst in Standard-Deutsch übersetzt. Meiner Erfahrung nach wird selten ein Plugin in Deutsch mit Höflichkeitsform (oder in andere Sprachformen) übersetzt. Warum ist das ein Problem?

Nehmen wir an, Sie wählen die Übersetzung in Höflichkeitsform (also mit der Anrede Sie) und installieren dazu ein Plugin, das keine Übersetzung dafür hat. Dann wird WordPress die Textinhalte des Plugins in englischer Sprache anzeigen – und das, obwohl es eine deutsche Standard-Übersetzung gibt. WordPress hat nämlich keinen internen Mechanismus, mit dem eine Fallback-Sprache (oder eine Ersatzsprache) definiert werden könnte.

Glücklicherweise gibt es für fast jedes Problem eine Lösung in Form eines Plugins. So auch in diesem Fall. Gleich zwei Plugins, die sich dieses Themas angenommen haben, sind:

- ▶ *Language Fallback* von Bernhard Kau (<https://de.wordpress.org/plugins/language-fallback>)
- ▶ *Preferred Languages* von Pascal Birchler (<https://de.wordpress.org/plugins/preferred-languages>)

Im zweiten Schritt weist WordPress Sie darauf hin, dass Sie sich die Zugangsdaten für die Datenbank zurechtlegen sollten (siehe Abbildung 1.4).

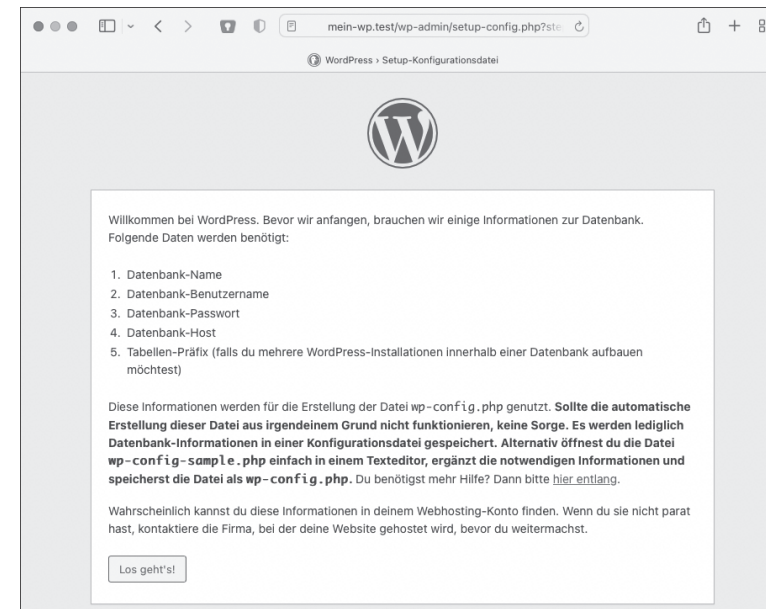


Abbildung 1.4 Allgemeine Hinweise auf die bevorstehende Datenbank-Installation

Die Installationsroutine betont (in fetter Schrift), dass sie die Datei *wp-config.php* aktualisieren wird. Denn die Datenbank-Zugangsdaten müssen an einer Stelle gespeichert werden. Im nächsten Bildschirm werden genau diese Daten abgefragt (siehe Abbildung 1.5).

Hier sollten die Zugangsdaten zu deiner Datenbank eingetragen werden. Im Zweifel frage bitte beim Support deines Webhostings nach.

Datenbank-Name Der Name der Datenbank, die du für WordPress verwenden möchtest.

Benutzername Dein Datenbank-Benutzername.

Passwort Dein Datenbank-Passwort.

Datenbank-Host Sollte localhost nicht funktionieren, erfrage bitte den korrekten Wert beim Support deines Webhostings.

Tabellen-Präfix Falls du mehrere WordPress-Installationen innerhalb einer Datenbank aufbauen möchtest, ändere diesen Eintrag.

Abbildung 1.5 Die Angaben zur Datenbank sind notwendig.

Datenbank-Verbindung mithilfe einer Socket-Datei

In den meisten Fällen lautet der Datenbank-Hostname *localhost*. Dies ist aber nicht immer der Fall: Es kommt darauf an, wie die Datenbank konfiguriert wurde. Deswegen kann es vorkommen, dass Sie einen Pfad zu einer Socket-Datei anstatt eines Hostnamens vom Webhoster bekommen. Sie erkennen das daran, dass die Angabe einer Pfadangabe ähnelt, zum Beispiel wie folgt:

```
/var/run/mysqld.sock
```

Ist dies der Fall, müssen Sie im Formularfeld **DATENBANK-HOST** ein *localhost*, gefolgt von einem Doppelpunkt, voranstellen. Auf diese Weise versteht WordPress, dass es sich um eine Socket-Verbindung handelt.

```
localhost:/var/run/mysqld.sock
```

Ändern der Datenbank-Portnummer

WordPress nimmt an, dass die Standard-Portnummer 3306 zu verwenden ist. Hat der Webhoster aber einen anderen Datenbank-Port gewählt, können Sie im Formularfeld **DATENBANK-HOST** die Portnummer durch einen Doppelpunkt getrennt angeben. Beispiele sind:

```
localhost:3309
```

```
datenbank.mein-wp.test:6619
```

Das Tabellenpräfix richtig wählen

Die WordPress-Datenbank ist so etwas wie das Gehirn der gesamten WordPress-Website. Jede einzelne Information ist dort gespeichert. Das macht diese Datenbank zu einem bevorzugten Ziel von Hackern. Diese führen automatisierte Codes für sogenannte *SQL-Injektionen* aus (ich gehe in Abschnitt 7.1.7, »Sichere SQL-Abfragen erzeugen«, noch detaillierter darauf ein). Wird das Standardpräfix nicht geändert, macht es das den Hackern leichter, Massenangriffe zu planen, weil sie auf das Standardpräfix *wp_* zielen können. Der beste Weg, Ihre Datenbank zu schützen, besteht also darin, das Datenbankpräfix zu ändern. Das ist während der Installationsroutine sehr einfach zu bewerkstelligen.

Nach dem Klick auf **SENDEN** erscheint im Erfolgsfall die Information, dass WordPress mit der Datenbank kommunizieren kann (siehe Abbildung 1.6). Im Hintergrund wurde die Datei *wp-config.php* entsprechend aktualisiert. Wie die Datei dann aussieht, zeige ich im nächsten Schritt.

Alles klar! Diesen Teil der Installation hast du geschafft. WordPress kann jetzt mit deiner Datenbank kommunizieren. Wenn du bereit bist, kannst du jetzt die ...

Abbildung 1.6 Die Datenbank-Zugangsdaten waren korrekt.

Im Fehlerfall zeigt WordPress an, dass die Datenbank-Verbindung fehlschlug (siehe Abbildung 1.7). In diesem Fall gelangen Sie über den Button ERNEUT VERSUCHEN zurück zum vorherigen Bildschirm.

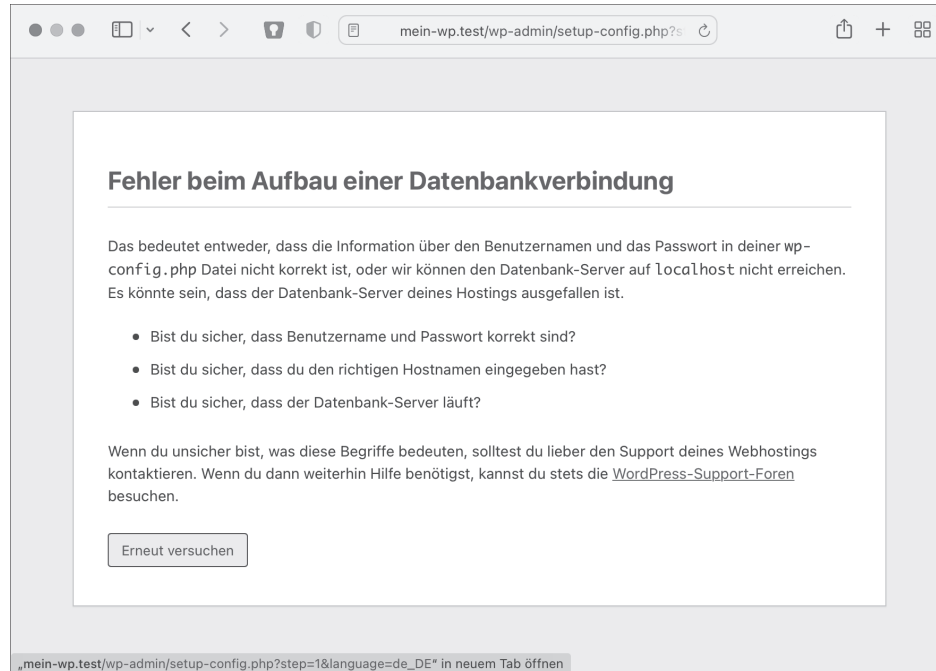


Abbildung 1.7 Fehler in der Datenbankverbindung.

Wenn Sie die Verbindung partout nicht einrichten können, fragen Sie beim Webhoster nach den korrekten Zugangsdaten und der entsprechenden Konfiguration.

Können Sie die Konfigurationsdatei nicht speichern?

Es ist möglich, dass WordPress Sie nach der Eingabe der Datenbank-Zugangsdaten darauf hinweist, dass die *wp-config.php*-Datei nicht auf dem Server abgespeichert werden kann (siehe Abbildung 1.8). Dies liegt möglicherweise daran, dass die Rechte im Dateisystem nicht entsprechend eingestellt wurden. Sie sollten daher bereits vorab gewährleisten, dass der Webserver oder gegebenenfalls PHP in das Verzeichnis schreiben darf, in dem WordPress installiert werden soll.. WordPress bietet Ihnen an, mit der Installation fortzufahren, indem der Inhalt (Abbildung 1.18) von Ihnen in eine Datei kopiert und manuell hochgeladen wird. Später werden Sie aber vor dem glei-

chen Problem stehen – genauer gesagt dann, wenn Sie Medien in die Medienbibliothek hochladen wollen. Diese werden nämlich ebenfalls im Dateisystem abgelegt, können aber nicht beschrieben werden.

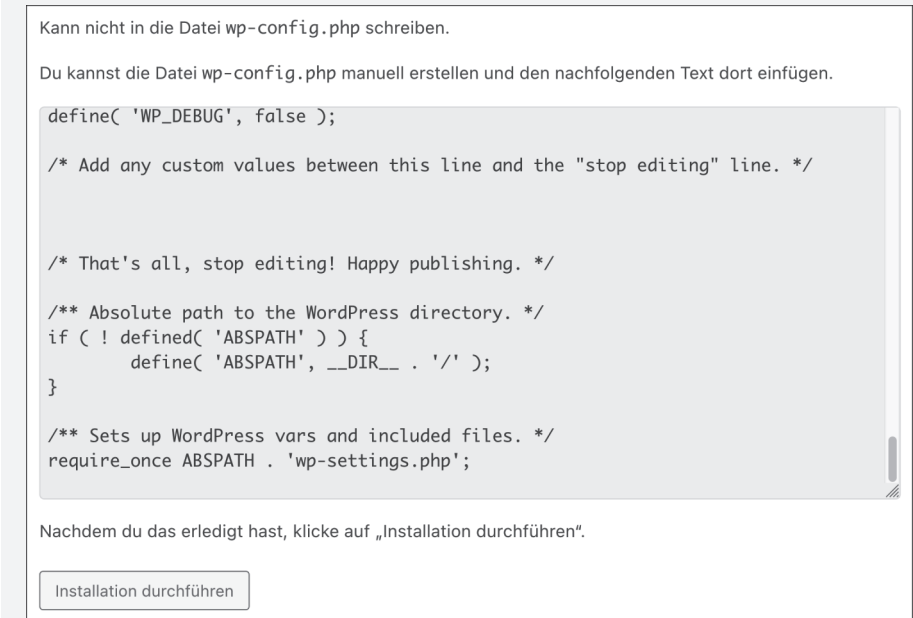


Abbildung 1.8 Der Inhalt der »wp-config.php«-Datei wird angezeigt und kann manuell kopiert werden.

Ich gehe davon aus, dass die Datenbank-Verbindung funktioniert hat und Sie deswegen mit dem Button INSTALLATION DURCHFÜHREN aus Abbildung 1.6 fortfahren können. Sie gelangen danach zum Begrüßungsbildschirm. In der Tat ist die Installation schon fast vorbei, aber WordPress scheut sich nicht, Sie an dieser Stelle noch mal zur 5-Minuten-Installation zu begrüßen.

Das Formular aus Abbildung 1.9 zeigt nur die allernötigsten Einstellungen, die zur Erstinstallation notwendig sind. Alles, was hier eingegeben wird, kann jederzeit nachträglich verändert werden.

Mit einem Klick auf WORDPRESS INSTALLIEREN wird die Datenbank installiert und der Benutzer mit den angegebenen Daten erstellt.

In der Regel bekommen Sie dann die Nachricht INSTALLATION ERFOLGREICH! angezeigt. Mit einem Klick auf ANMELDEN gelangen Sie dann zur Login-Maske.

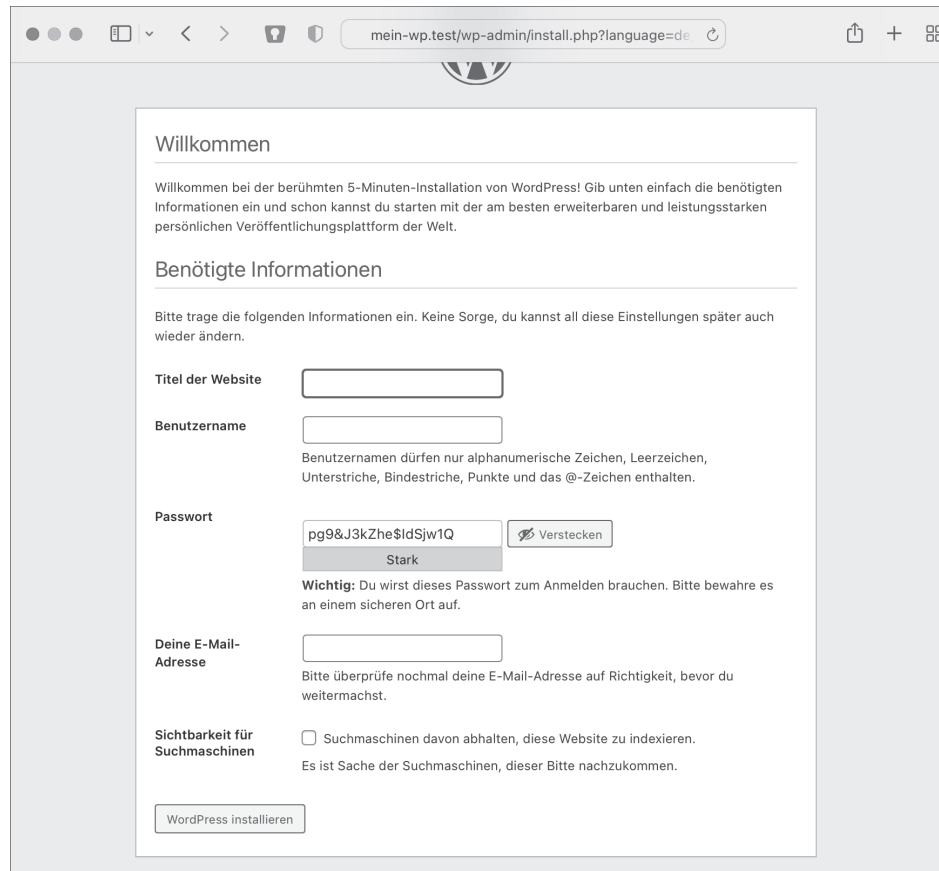


Abbildung 1.9 Begrüßungsbildschirm zur 5-Minuten-Installation

Nun werfen wir noch einen kurzen Blick in die *wp-config.php*, die automatisch erstellt wurde:

```
<?php
define( 'DB_NAME', 'local' );
define( 'DB_USER', 'root' );
define( 'DB_PASSWORD', 'root' );
define( 'DB_HOST', 'localhost' );
define( 'DB_CHARSET', 'utf8mb4' );
define( 'DB_COLLATE', '' );

define( 'AUTH_KEY', '2*dz#[/yRp>!+...' );
define( 'SECURE_AUTH_KEY', '^?bet,4L1@!,]df...' );
define( 'LOGGED_IN_KEY', 'wcyadU@>pv?/AucsQHP%' );
define( 'NONCE_KEY', 'w<~x2vb9vV1...' );
define( 'AUTH_SALT', '>{wRo*UtHtXK;xR5ZgcI10...' );
```

```
define( 'SECURE_AUTH_SALT', 'L%.B0]k3(mz~uT-is3}OIjprh...' );
define( 'LOGGED_IN_SALT', 'MVZfe/Nh()OBvKG=..Abx...' );
define( 'NONCE_SALT', '1;b05__^*o7~J1?krdFy7...' );

$table_prefix = 'mywpi_';

define( 'WP_DEBUG', false );

/* Add any custom values between this line and the "stop editing" line. */

/* That's all, stop editing! Happy publishing. */

if ( ! defined( 'ABSPATH' ) ) {
    define( 'ABSPATH', __DIR__ . '/' );
}

require_once ABSPATH . 'wp-settings.php';
```

Listing 1.1 Die Datei »wp-config.php« (Einige Zeilen wurden gekürzt und Kommentare teilweise entfernt.)

Im ersten Teil sehen Sie, dass die Zugangsdaten für die Datenbank in PHP-Konstanten hinterlegt worden sind. So kann jederzeit darauf zugegriffen werden.

Während der Installationsroutine wurden ein paar eindeutige Authentifizierungsschlüssel und sogenannte *Salts* angelegt. In der Regel werden diese von der Website <https://api.wordpress.org/secret-key/1.1/salt> abgerufen und eingebunden. In der Kryptografie dienen diese Salts dazu, die Entropie der Eingabe zu erhöhen, also um Passwörter und sonstige geheime Daten sicher in der Datenbank zu speichern.

Darunter finden Sie schließlich das Tabellenpräfix in der Variablen `$table_prefix` und eine Konstante `WP_DEBUG`, auf die ich in Abschnitt 11.2.1, »Das Error-Logging aktivieren«, noch eingehen werde.

Zwischen den zwei Kommentaren, die ich stehen gelassen habe, haben Sie Platz für eigene Variablen, die Sie seitenweit zur Verfügung stellen können. Vermeiden Sie tunlichst, hier PHP-Code einzufügen, der nicht zu Konfigurationszwecken verwendet wird. Wie schon erwähnt, handelt es sich hier um die Konfigurationsdatei. Diese sollte nicht zweckentfremdet werden.

Sind der Authentifizierungsschlüssel und die Salts leer?

Wie oben angesprochen, werden die entsprechenden Schlüssel von einer API abgerufen, die bei *wordpress.org* liegt. Erfahrungsgemäß ist die Website 99,9% der Zeit erreichbar. Ist sie das nicht, werden die Schlüssel lokal erstellt. Das gilt jedoch nur,

wenn die offizielle 5-Minuten-Installation durchlaufen wird. Ist das nicht der Fall (z. B. aufgrund einer schlecht konfigurierten 1-Klick-Installation eines Webhosters), sind die entsprechenden Schlüssel und »Salze« leer. In der Praxis habe ich das schon ein paarmal erlebt. Sie tun also gut daran, einmal einen Blick in die `wp-config.php` zu werfen – egal, ob es sich dabei um eine neue oder eine alte Installation handelt.

1.2 Lokale Laufzeitumgebungen

Wenn Sie WordPress auf dem heimischen Rechner zum Laufen bringen wollen, benötigen Sie drei Dinge: eine Datenbank (MySQL oder MariaDB), PHP und einen Webserver (in der Regel Apache oder *nginx*). Im Grunde könnte sich jeder diese drei Softwarepakete auf dem Rechner installieren oder Tools wie *Docker* oder *Vagrant* nutzen und entsprechend konfigurieren. Wer sich damit nicht befassen will, greift auf Softwarepakete zurück, die eine lokale Serverumgebung erstellen. So sind WordPress-Installationen innerhalb kürzester Zeit aufgesetzt und einsatzbereit.

Einige robuste Softwarepakete, die sich in der Vergangenheit bewährt haben, liste ich in Tabelle 1.1 auf. Möglicherweise nutzen Sie bereits eine der nachfolgend genannten oder eine ähnliche Software. Ist Letzteres der Fall, dann bleiben Sie bitte dabei. Sie müssen das Rad nicht neu erfinden, um WordPress in Betrieb nehmen zu können. Sie können WP wie in Abschnitt 1.1 beschrieben installieren.

	Mac	Win	Linux	
Local	+	+	+	<i>localwp.com</i>
MAMP	+	+	–	<i>www.mamp.info</i>
DevKinsta	+	+	+	<i>kinsta.com/devkinsta</i>
XAMPP	+	+	+	<i>www.apachefriends.org</i>
WAMP	–	+	–	<i>www.wampserver.com</i>
DesktopServer	+	+	–	<i>serverpress.com</i>

Tabelle 1.1 Eine Auswahl an Softwarepaketen, die in kürzester Zeit eine lokale Softwareumgebung einrichten können

Natürlich haben alle Tools ihre Vor- und Nachteile. Das fängt schon bei den Kosten an. Von den meisten Tools gibt es eine kostenfreie sowie eine kostenpflichtige Version. MAMP erlaubt beispielsweise nur die Installation von zwei PHP-Versionen, während die PRO-Version unlimitiert ist. Wenn Sie also mit mehreren verschiedenen

PHP-Versionen testen müssen, werden Sie mit der PRO-Version oder einem anderen Softwarepaket glücklicher. Hier müssen Sie nach Ihren persönlichen Präferenzen und den jeweiligen Anforderungen entscheiden.

Mein Favorit ist seit Jahren das Tool *Local*. Es hieß früher *Pressmatic*. Dann wurde es vom Webhoster Flywheel (und dieser wiederum von WPEngine) gekauft, umbenannt und zum kostenlosen Download angeboten.

Während alteingesessene Tools wie MAMP, XAMPP und WAMP keinen direkten WordPress-Support hatten, wurde *Local* von Anfang an auf die WordPress-Entwicklung ausgelegt. Es enthält beispielsweise die vorinstallierte und vorkonfigurierte *WP-CLI* (also das Kommandozeilentool von WordPress, siehe Kapitel 20). Mit ein paar Klicks wird WordPress auf einer lokalen Domain (zum Beispiel *mein-wp.test*) installiert. Kurz danach können Sie loslegen.

Aber auch erstgenannte Tools holen mittlerweile auf. So beherrscht *MAMP PRO* ebenso die 1-Klick-Installation. Diese Installationsvariante ist etwas, das Sie sicherlich schnell kennen und lieben lernen werden. Vor allem, wenn Sie viele Projekte betreuen. Auch das schnelle Wechseln von PHP-Versionen, der rasche Einsatz der *WP-CLI* und die Unterstützung des Software-Tools *PhpStorm* eröffnen Möglichkeiten, die ich persönlich nicht mehr missen möchte.

1.2.1 Die lokale Laufzeitumgebung einrichten

In diesem Abschnitt lernen Sie, wie Sie das Tool *Local* und wie Sie mit ihm umgehen. Sie werden merken, dass die anderen Programme ähnlich funktionieren.

Als Erstes können Sie das Tool unter <https://localwp.com/releases/> für das Betriebssystem Ihrer Wahl herunterladen. Die Installation selbst sollte selbsterklärend sein. Sie funktioniert genauso, wie Sie auch andere Programme installieren.

Nach der Installation und dem Öffnen der Applikation können Sie über das große +-Zeichen (siehe Punkt ⑤ in Abbildung 1.10) eine neue Seite anlegen. Im sich dann öffnenden Fenster reicht es im Grunde, wenn Sie einen Namen eingeben und dann zweimal auf CONTINUE klicken. Danach werden Sie aufgefordert, einen Benutzernamen und ein Passwort zu vergeben. Ein Klick auf ADD SITE wird die entsprechende Seite erstellen.

Ist eine Passwort-Eingabe nötig?

Möglicherweise werden Sie in den unterschiedlichen Betriebssystemen aufgefordert, Ihr Passwort einzugeben. Das ist nötig, weil die verschiedenen Server-Dienste (Webserver, Datenbank, PHP) im Hintergrund laufen und dazu bestimmte Ports geöffnet werden. Unter Windows müssen Sie womöglich die Firewall-Einstellungen

anpassen, indem Sie die entsprechende Meldung des Windows Defenders durch Klick auf ZUGRIFF GEWÄHREN (oder ähnlich) akzeptieren. Unter macOS wird das Passwort abgefragt, weil die `/etc/hosts`-Datei angepasst wird. Damit wird dem Betriebssystem mitgeteilt, wo es die Domain findet, die Sie im Browser aufrufen. Es handelt sich also um eine Art Umleitung vom Browser hin zur *Local*-App.

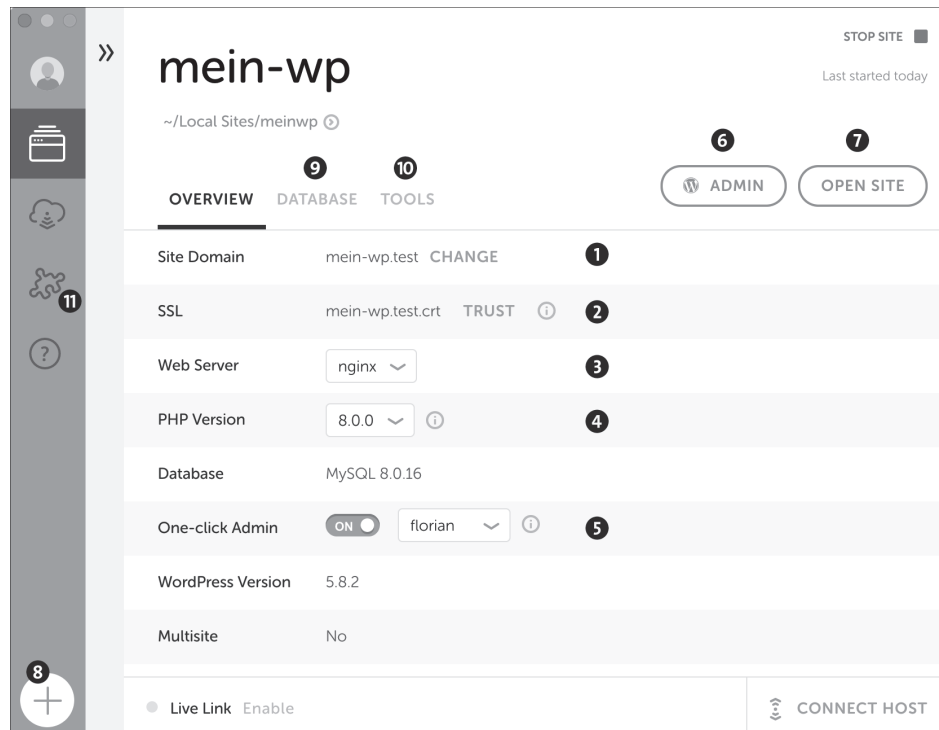


Abbildung 1.10 »Local« zeigt die erstellte Webseite.

Im Tab OVERVIEW gibt es eine Menge Einstellmöglichkeiten. Da wäre zum einen die Domain **1**, mit der Sie die lokal installierte WordPress-Seite aufrufen können. Mit jeder Website wird auch ein SSL-Zertifikat **2** erstellt, das sich durch einen Klick auf TRUST installieren lässt. Damit umgehen Sie die Fehlermeldung im Browser, die darauf hinweist, dass die Verbindung unsicher ist.

In der jüngsten Vergangenheit wurde die Option zur Wahl des Webservers zur *Local*-App hinzugefügt **3**. Sie haben aktuell die Wahl zwischen *nginx* und *Apache*.

Die Einstellung zur PHP-Version **4** dürfte selbsterklärend sein. *Local* reicht neue PHP-Versionen zeitnah nach, also nach dem Erscheinen einer neuen Version. Schließlich lässt sich noch der sogenannte *One-Click-Admin*-Zugang **5** einrichten. Damit können Sie sich automatisch in WordPress anmelden lassen, wenn Sie den

Button ADMIN **6** anklicken. Der Button OPEN SITE **7** führt Sie schließlich zum Frontend der Installation, damit Sie sich die Website ansehen können.

Im Tab DATABASE **9** finden Sie einen weiteren Helfer, der Ihnen das Entwickeln deutlich einfacher machen kann. Die Entwickler von *Local* haben nämlich das PHP-Skript *Adminer* in das Projekt eingebunden. Es ist ein Tool zur Verwaltung der SQL-Datenbanken im Browser. Wenn Sie also einmal einen Blick in eine der Datenbank-Tabellen werfen müssen, können Sie das hier tun.

Unter TOOLS **10** finden Sie noch *MailHog*. Es fängt alle E-Mails von WordPress ab und zeigt sie dann in der Browseroberfläche an, die Sie bequem per Klick öffnen können.

Natürlich kann *Local* noch viel mehr. Über Plugins **11** lässt sich der Funktionsumfang weiter ausbauen. Eine Erweiterung, die standardmäßig aktiv ist, nennt sich *Xdebug + PhpStorm*. Sie integriert automatisch die sogenannte *Run-Konfiguration* in *PhpStorm*. Mehr dazu lesen Sie in Abschnitt 11.3.2, »Xdebug mit Local und *PhpStorm* verbinden«.

1.3 Entwickeln mit dem Texteditor?

Gut, dass es Programmierung nicht erst seit gestern gibt. Dementsprechend groß ist die Auswahl an Editoren. Für jede Programmiersprache gibt es Dutzende Tools. Das Schöne daran ist: Man muss nicht immer Geld ausgeben, um einzusteigen. Denn es gibt auch kostenlose Programme oder Open-Source-Projekte, auf die Sie zurückgreifen können.

Neben *NotePad* unter Windows oder *TextEdit* unter macOS gibt es mittlerweile deutlich leistungsfähigere Tools, die Ihnen das Entwickeln erleichtern. Mit Betriebssystem-Bordmitteln lassen sich zwar ebenfalls umfangreiche Programme schreiben, wirklich komfortabel ist es nicht. Meines Erachtens dienen diese übersichtlichen Tools eher dazu, mal schnell ein paar Zeilen Code zu ergänzen, anstatt große Plugins, Themes oder gar Blöcke zu entwickeln.

Deswegen möchte ich in einem ersten Schritt einige Programme auflisten, die sich zur Arbeit mit PHP, HTML sowie CSS und JavaScript eignen. Denn all diese Programmier- und Auszeichnungssprachen sind für ein WordPress-Projekt vonnöten.

Im folgenden Abschnitt 1.4 stelle ich dann zwei sehr populäre Programme vor (*PhpStorm* und *Visual Studio Code*), die in der WordPress-Welt häufig verwendet werden und zu denen Sie daher viele Informationen im Internet finden. Das hilft, wenn Sie sich später tiefer in die Thematik einarbeiten wollen. Denn die Tools werden mittlerweile immer umfangreicher und ausgeklügelter. Es lohnt sich also, ein wenig tiefer einzusteigen, denn Sie können Funktionen entdecken, die Ihnen viel Arbeit ersparen. Einige davon werde ich später vorstellen.

Name	Website	Lizenz	Kosten	B*
PhpStorm	jetbrains.com/phpstorm	Proprietär, Abo	199 € im ersten, 159 € im zweiten, 110 € jedes weitere Jahr.	MWL
Eclipse PDT	eclipse.org/pdt	OpenSource, EPL v2	Kostenlos	MWL
Visual Studio Code	code.visualstudio.com	OpenSource, MIT	Kostenlos	MWL
NetBeans	netbeans.apache.org	OpenSource, Apache v2	Kostenlos	MWL
Sublime Text	sublimetext.com	Proprietär	99 \$ für drei Jahre	MWL
Atom	atom.io	OpenSource, MIT	Kostenlos	MWL
Notepad++	notepad-plus-plus.org	OpenSource, GPL v2	Kostenlos	W
TextMate	macromates.com	Proprietär	Kostenlos, 59 \$ für den vollen Funktionsumfang	M
Vim	vim.org	OpenSource, GPL-kompatibel	kostenlos	MWL
UltraEdit	ultraedit.com	Proprietär, Abo	99 \$ pro Jahr	MWL
Nova	nova.app	Proprietär, Abo	99 \$, Updates für ein Jahr	M
Brackets	Brackets.io	OpenSource, eigene	kostenlos	MWL

Tabelle 1.2 Übersicht über verschiedene Editoren (* Betriebssystem: M = macOS, W = Windows, L = Linux)

Sicherlich ist die Liste in Tabelle 1.2 nicht vollständig. Ich habe diejenigen Editoren herausgesucht, die derzeit aktiv weiterentwickelt werden. Es gibt noch viele weitere

Editoren, die interessant sind – auch reine Cloud-Lösungen, also Entwicklungsumgebungen, die nur im Browser laufen. Sicherlich werden Sie Ihren Lieblingseditor finden. Mein persönlicher kostenloser Favorit wäre *Visual Studio Code*.

Was ist eine IDE?

Bei einigen der Programme aus Tabelle 1.2 handelt es sich um sogenannte *IDE*-Programme, also um integrierte Entwicklungsumgebungen. Eine IDE ist eine Sammlung von Programmen, vereint in einem einzigen Software-Tool. Sie enthält umfangreiche und hilfreiche Werkzeuge, die Sie bei täglichen und oft wiederkehrenden Aufgaben unterstützen. *PhpStorm* hat zum Beispiel einen eingebauten FTP-Client mit dem sich Daten auf einen Server (vollautomatisch) hochladen lassen. Auch lässt sich Code direkt in dem Programm per *SVN* (Subversion) oder *Git* versionieren. Theoretisch könnten Sie Ihren ganzen Arbeitstag nur in dem Programm verbringen, ohne es ein einziges Mal verlassen zu müssen. Denn alle Tools, die Sie für das tägliche Arbeiten benötigen, finden Sie dort.

Ein großer Nachteil ist, dass man von solchen Programmen buchstäblich erdrückt wird, wenn man sie zum ersten Mal öffnet. Eine schier unendliche Anzahl von Konfigurationsmöglichkeiten und Schaltflächen wirkt zunächst überfordernd. Eine gewisse Einarbeitungszeit ist also nötig, während Sie mit Texteditoren wie *SublimeText* schnell loslegen können, weil sie so einfach strukturiert sind, dass die Handhabung sofort klar ist.

Visual Studio Code versucht hier die Brücke zu schlagen. Es startet als überschaubares Editor-Programm mit der Möglichkeit, es durch Plugins zu erweitern. Das hat den Vorteil, dass die Einarbeitung leicht ist. Falls Sie eine entsprechende Funktion benötigen, ergänzen Sie diese durch eine Erweiterung. Erst danach können Sie sich mit der Konfiguration beschäftigen. Die Lernkurve ist also deutlich flacher als bei den großen IDEs. Nicht umsonst ist *Visual Studio Code* das beliebteste Tool im Bereich der Webentwicklung.⁹

1.4 Software-Entwicklungstools einrichten

Ich möchte mit *PhpStorm* beginnen, meinem Tool der Wahl, das seit Jahren mein treuer Begleiter ist. Ich gehe davon aus, dass ein Webserver lokal auf Ihrem Rechner läuft (siehe Abschnitt 1.2) und dass Sie WordPress installiert haben (siehe dazu Abschnitt 1.1).

Wenn Sie eine App wie *Local* benutzen, haben Sie beim Erstellen einer neuen Testseite den Speicherort der Website bestimmt. Falls Sie ihn nicht angegeben haben

⁹ Zum Beispiel laut der StackOverflow-Umfrage von 2019: <https://insights.stackoverflow.com/survey/2019#development-environments-and-tools>

oder nicht wissen, wo die Dateien liegen, haben Sie in den meisten Programmen die Möglichkeit, sich die Dateien im Finder (bzw. im Explorer unter Windows) anzeigen zu lassen. In *Local* auf dem Mac geht das zum Beispiel über den Eintrag REVEAL IN FINDER (siehe Abbildung 1.11). Sie finden es im aufklappenden Menü, wenn Sie einen Rechtsklick auf der jeweiligen Seite durchführen. Merken Sie sich, wo die Dateien liegen. Denn Sie benötigen den Pfad beim Einrichten der nachfolgend genannten Programme.

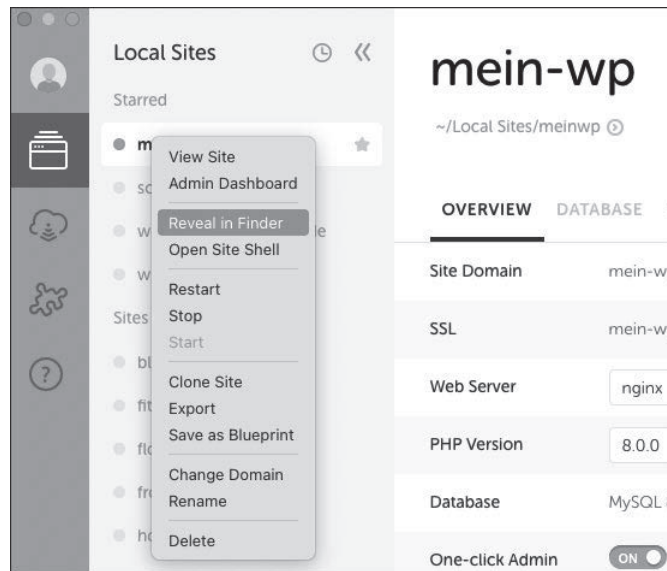


Abbildung 1.11 Ein Klick auf »Reveal in Finder« bringt Sie zu den WordPress-Dateien auf dem Laufwerk.

1.4.1 PhpStorm einrichten

Bei PhpStorm handelt es sich um eine IDE, also um eine integrierte Entwicklungsumgebung. Der Funktionsumfang ist sehr hoch. Das Programm lässt sich also nicht nur zum Schreiben von Code einsetzen. Sie können PHP-Programme debuggen, Code versionieren und vieles mehr. Aber am besten ist es, wenn Sie sich das einmal live ansehen.

Alle Screenshots entstanden in der Version 2021.2.3. Es ist davon auszugehen, dass neuere Versionen ähnlich aufgebaut sein werden. Erfahrungsgemäß ändert sich an der optischen Aufmachung von PhpStorm eher wenig.

PhpStorm herunterladen

Sie können das Programm unter <https://www.jetbrains.com/de-de/phpstorm/download> kostenfrei herunterladen und für 30 Tage ausgiebig testen. Die Installation

erfolgt so, wie Sie auch andere Programme unter Ihrem Betriebssystem installieren. Die oben genannte Website hält jedoch auch Informationen zur Installation bereit.

Projekt öffnen

Nach dem Öffnen des Programms öffnet sich ein kleines Fenster mit dem Titel WELCOME TO PHPSHORM. Dort können Sie auf den Button OPEN klicken (siehe Punkt ❶ in Abbildung 1.12). Alternativ gelangen Sie über das Menü unter FILE • OPEN... zum gleichen Kommando. Wählen Sie dann das Verzeichnis auf Ihrem Laufwerk aus, in dem Sie Ihre WordPress-Installation finden. Das ist das Verzeichnis, in dem auch die *wp-config.php* zu finden ist. In Falle von Local ist das der Unterordner */app/public*.

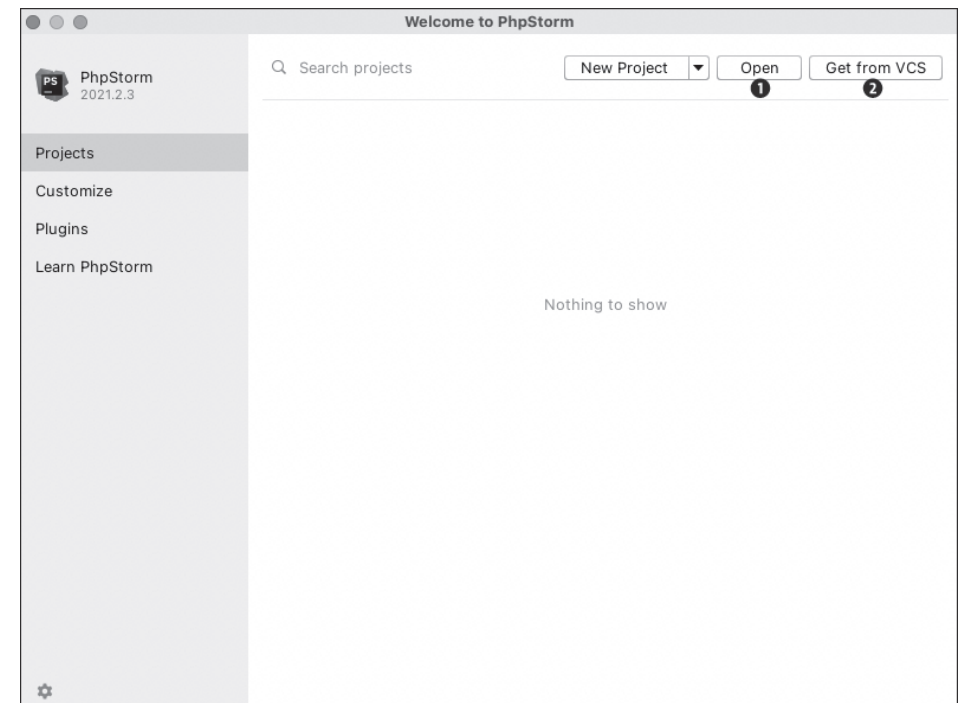


Abbildung 1.12 Das Willkommensfenster von PhpStorm

Selbstverständlich können Sie auch jederzeit ein Projekt aus einem VCS (Version-Control-System), also einem Versionierungssystem, auschecken ❷.

Erstmaliges Öffnen von PhpStorm

Erst jetzt öffnet sich das Hauptfenster von PhpStorm. Zuallererst wird es die Dateien im Verzeichnis indexieren. Das kann, je nach Größe des Projekts, etwas dauern. Bei einer neuen WordPress-Installation geschieht dies aber innerhalb von Sekunden.

Der EVENT LOG (in der Regel rechts unten im Bild) wird Sie über einige Dinge informieren, die PhpStorm automatisch eingestellt hat (dass die PHP-Version zum Beispiel auf 7.1 gestellt wurde). Das geschieht, weil die Software nach der Indexierung intelligent erkennt, dass sich PHP-Code in den Dateien befindet, der ausschließlich in dieser Version vorkommen kann. Das heißt nicht, dass Sie sich auch daran halten müssen. Im Grunde können Sie diese Meldungen erst einmal getrost ignorieren. Wir werden die Anwendung nämlich im nächsten Schritt manuell einrichten.

Der Aufbau der Anwendung ist relativ einfach. Am linken Bildschirmrand sehen Sie die Dateien und Verzeichnisse des Projekts, während Sie rechts die jeweilige geöffnete Datei mit deren Inhalt sehen.

PhpStorm für WordPress vorbereiten

Das Geniale an PhpStorm ist, dass es eine direkte WordPress-Unterstützung hat. Über den Menüeintrag **PHPSTORM • PREFERENCES** auf dem Mac (oder **FILE • SETTINGS** unter Windows) gelangen Sie zum Einstellungsfenster aus Abbildung 1.13. Im linken Menüeintrag klicken Sie auf den kleinen Pfeil neben **PHP** ❶. Daraufhin klappt ein Menü aus. Dort wählen Sie **FRAMEWORKS** ❷ aus und klicken schließlich auf **WORDPRESS** ❸.

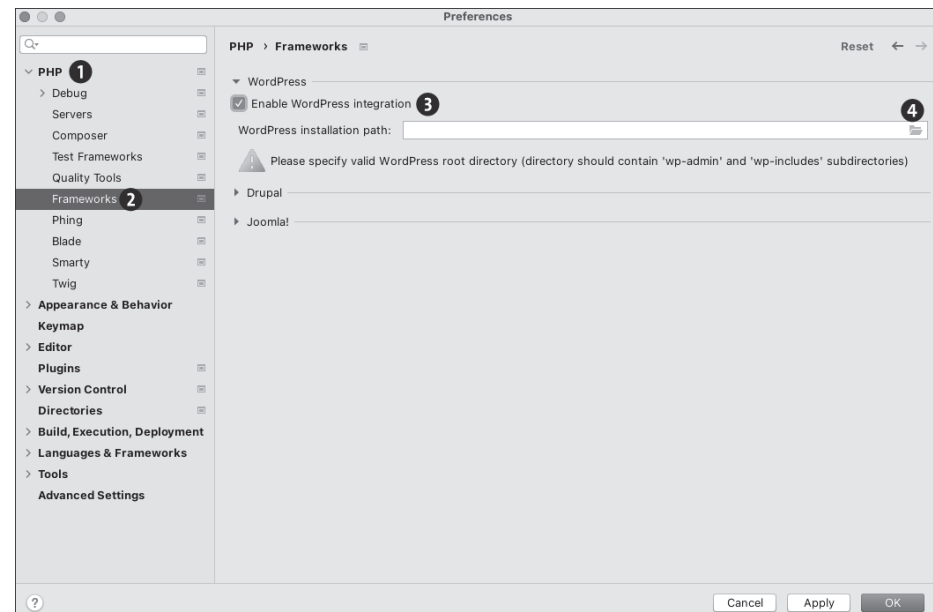


Abbildung 1.13 PhpStorm bietet eine direkte Unterstützung von WordPress.

Über das kleine Ordner-Symbol ❹ wählen Sie schließlich das Verzeichnis der aktuellen WordPress-Installation aus. Ein Klick auf **OK** schließt das Fenster. PhpStorm wird daraufhin die Dateien noch einmal mit diesen neuen Einstellungen indexieren.

Von nun an bietet PhpStorm Ihnen eine umfassende Programmierunterstützung für die Entwicklung von WordPress:

- ▶ Code-Vervollständigung
- ▶ Suche nach Hook-Registrierungen (siehe dazu Kapitel 3)
- ▶ Suche nach Funktionen, die als Parameter für die Hook-Registrierung angegeben werden
- ▶ Navigation zwischen Hook-Registrierungen und Hook-Aufrufen
- ▶ die Möglichkeit, den Codierungsstil in Übereinstimmung mit den WordPress-Coding-Standards zu konfigurieren
- ▶ Anzeige der offiziellen WordPress-Dokumentation direkt aus der Anwendung und so weiter

Abbildung 1.14 bis Abbildung 1.16 zeigen einige dieser Funktionen in Aktion.

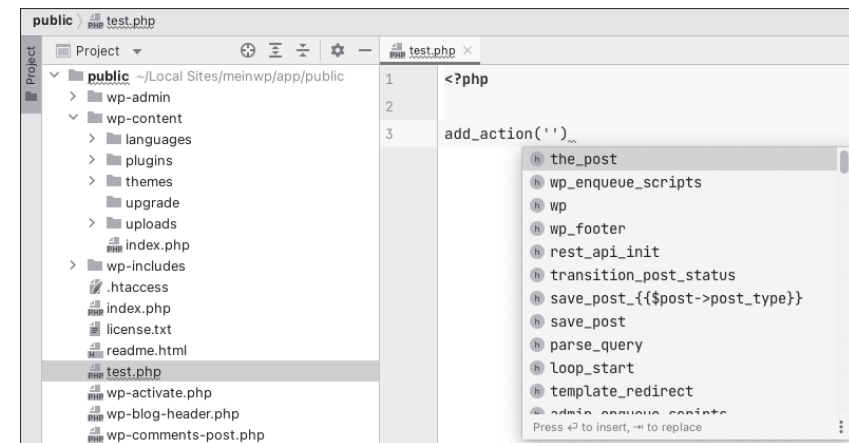


Abbildung 1.14 Die Code-Vervollständigung in Aktion: PhpStorm schlägt Hooks vor.



Abbildung 1.15 Durch einen Klick auf den blauen Kreis mit dem Pfeil ❶ springen Sie in den Quellcode von WordPress an die Stelle, an der der Hook definiert wurde. Die farbige Hervorhebung in ❷ zeigt, dass WordPress die Verbindung zwischen dem String und der Funktionsdefinition kennt.

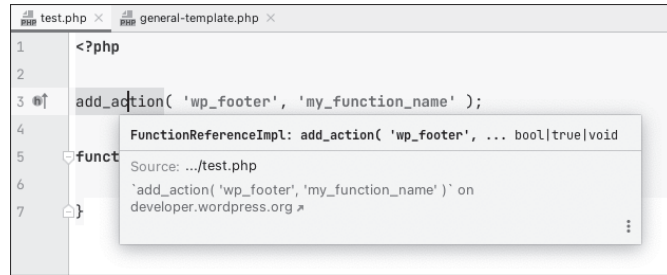


Abbildung 1.16 Lassen Sie den Mauszeiger etwas länger über den WordPress-Funktionen verweilen, öffnet sich eine kleine Funktionsreferenz mit der Möglichkeit, eine Suche bei »developer.wordpress.org« auszulösen.

Tipp

Wenn Sie Interesse an weiterführenden Themen in Zusammenhang mit PhpStorm haben, besuchen Sie die Adresse <https://wp-typ.de/buch/phpstorm-artikel>. Dort habe ich einige Artikel aufgelistet, die beschreiben, wie Sie mit der Software umgehen und wie Sie sie mit anderer Software verbinden.

1.4.2 Visual Studio Code einrichten

Visual Studio Code, kurz *VS Code*, versteht sich nicht als IDE. Es ist ein kostenloser Quelltext-Editor von Microsoft, der aber Grundzüge einer IDE aufweist, weil er sich durch Erweiterungen entsprechend konfigurieren lässt. Bis auf die Ähnlichkeit im Namen hat das Programm nichts mit *Visual Studio* gemeinsam. In VS Code gibt es keine Projektdateien wie etwa bei Visual Studio. Deswegen arbeitet Letzteres mit sogenannten *Workspaces*. Das sind Arbeitsumgebungen, in denen der Bearbeitungs-zustand und die Reihenfolge der geöffneten Dateien bis zum nächsten Öffnen gespeichert werden – ganz ähnlich wie bei PhpStorm.

Ich habe Visual Studio Code ausgewählt, weil es zum einen kostenlos erhältlich ist und weil es sich zum anderen immer größerer Beliebtheit erfreut. Aus der StackOver-flow-Umfrage aus dem Jahre 2021 ging hervor, dass 71,06 % der Befragten das Pro-gramm nutzen.¹⁰ Das ist schon ein gewaltiger Marktanteil, wenn man bedenkt, dass es über 80.000 Teilnehmer gab.

VS Code herunterladen

Sie können VS Code unter <https://code.visualstudio.com/> kostenlos herunterladen. Die Installation erfolgt so, wie Sie jedes andere Programm auch unter Ihrem Betriebs-

¹⁰ Vgl. StackOverflow: 2021 Developer Survey, in: StackOverflow, 2021, <https://insights.stackover-flow.com/survey/2021#integrated-development-environment> (abgerufen am 07.12.2021).

system aufsetzen. Unter <https://code.visualstudio.com/docs/setup/setup-overview> finden Sie weitere Unterstützung, sollten Sie bei der Installation nicht weiterkom-men.

Ein Projekt öffnen

Nach dem Öffnen des Programms werden Sie vom GET STARTED-Bildschirm begrüßt (siehe Abbildung 1.17). Hier können Sie unter anderem auswählen, welches Theme Sie nutzen möchten (hell, dunkel, hoher Kontrast) ①. Darüber hinaus folgen mit weite-ren Klicks auf NEXT SECTION mehr Einstellmöglichkeiten (②, ③ usw.).

Nachdem Sie VS Code zum ersten Mal geöffnet haben, bietet das Tool Ihnen an, ein Sprachpaket zu installieren ④. Das ist ein kleiner Vorteil gegenüber PhpStorm, das nur auf Englisch verfügbar ist. Ich werde das deutsche Sprachpaket installieren und nachfolgend die deutschen Begriffe nutzen, falls das möglich ist.

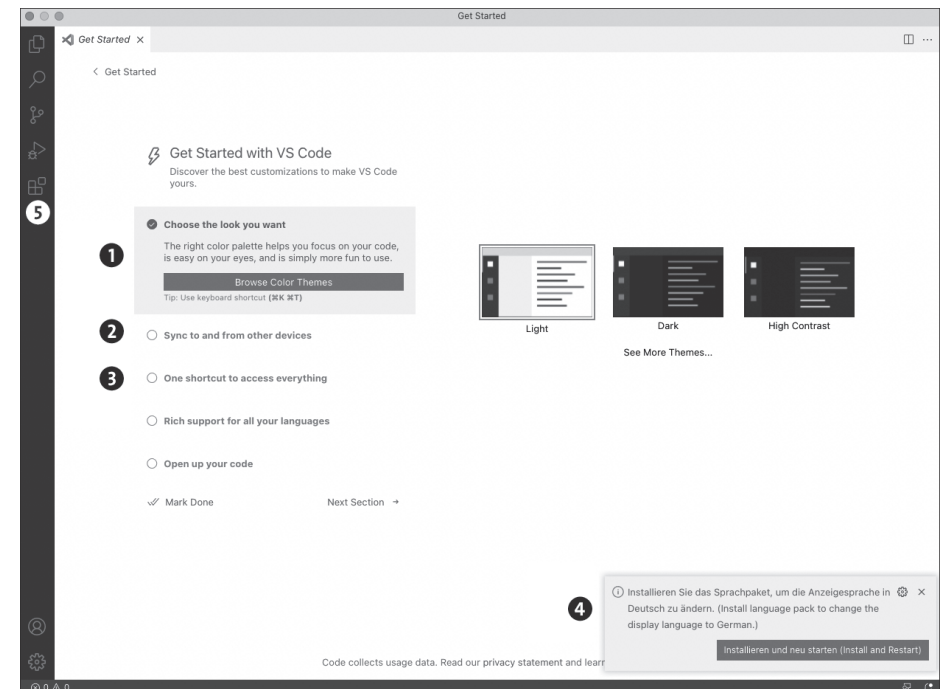


Abbildung 1.17 Der Welcome-Screen von VS Code

Im letzten Abschnitt in der Einstellungsliste (OPEN UP YOUR CODE) lässt sich über den blauen Button ein Verzeichnis auswählen. Alternativ gelangen Sie über FILE • OPEN zum gleichen Kommando. Wählen Sie dann das Verzeichnis auf Ihrem Laufwerk aus, in dem Ihre WordPress-Installation liegt. Das ist das Verzeichnis, in dem auch die *wp-config.php* zu finden ist. In Falle von *Local* ist das der Unterordner */app/public*.

Telemetriedaten

Ich möchte noch darauf hinweisen, dass Visual Studio Code standardmäßig Telemetriedaten an Microsoft übermittelt. Diese Funktion ist automatisch aktiv, kann aber über den Menüpunkt DATEI • EINSTELLUNGEN • TELEMETRIEEINSTELLUNGEN abgeschaltet werden.

Erstmaliges Öffnen von VS Code

Nun öffnet sich das Hauptfenster von VS Code. Auch dieses Programm wird die Dateien im WordPress-Verzeichnis indexieren, damit Sie diese durchsuchen können. Im Gegensatz zu PhpStorm bringt VS Code aber keine weiteren Meldungen, weist Sie also nicht auf etwaige PHP-Versionen oder Ähnliches hin.

Aufgebaut ist VS Code (wie wahrscheinlich jede App dieser Art) wie folgt: Links befindet sich der Datei- und Verzeichnisbaum, rechts die jeweilige geöffnete Datei mit ihrem Inhalt.

VS Code für WordPress vorbereiten

VS Code wirkt im Vergleich zu PhpStorm recht schlank. Es hat daher keine eingebauten WordPress- oder gar PHP-Funktionen. Das alles lässt sich aber, wie bereits mehrfach erwähnt, mit Erweiterungen nachrüsten.

In Abbildung 1.17 sehen Sie links im Bild die schwarze Menüleiste. Das letzte Icon in der oberen Hälfte ist der Menüpunkt für die Erweiterungen. Nach einem Klick öffnet sich ein Reiter mit einem Suchfeld. Dort können Sie die in Tabelle 1.3 gelisteten Pakete suchen und installieren. In Abbildung 1.18 und Abbildung 1.19 sehen Sie, dass die Funktionen denen in PhpStorm ähneln.

Erweiterungspaket	Beschreibung
PHP Intelephense*	Code-Vervollständigung, Symbolnavigation etc.
WordPress Snippets**	Code-Vervollständigung für den WordPress-Kern
WordPress Hooks IntelliSense***	Vervollständigung für WordPress-Hooks

* Zum Download unter: <https://marketplace.visualstudio.com/items?itemName=bmewburn.vscode-intelephense-client> oder mit der ID `wordpresstoolbox.wordpress-toolbox`

** Zum Download unter: <https://marketplace.visualstudio.com/items?itemName=wordpresstoolbox.wordpress-toolbox> oder mit der ID `bmewburn.vscode-intelephense-client`

*** Zum Download unter: <https://marketplace.visualstudio.com/items?itemName=johnbillion.vscode-wordpress-hooks> oder mit der ID `johnbillion.vscode-wordpress-hooks`

Tabelle 1.3 Erweiterungen für VS Code

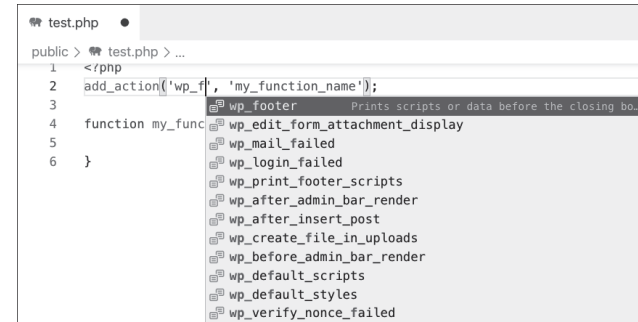


Abbildung 1.18 Code-Vervollständigung für WordPress-Hooks

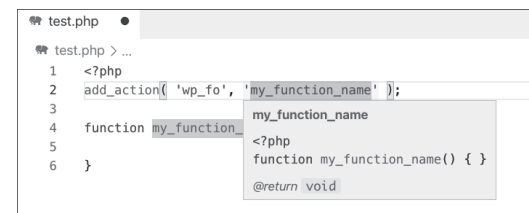
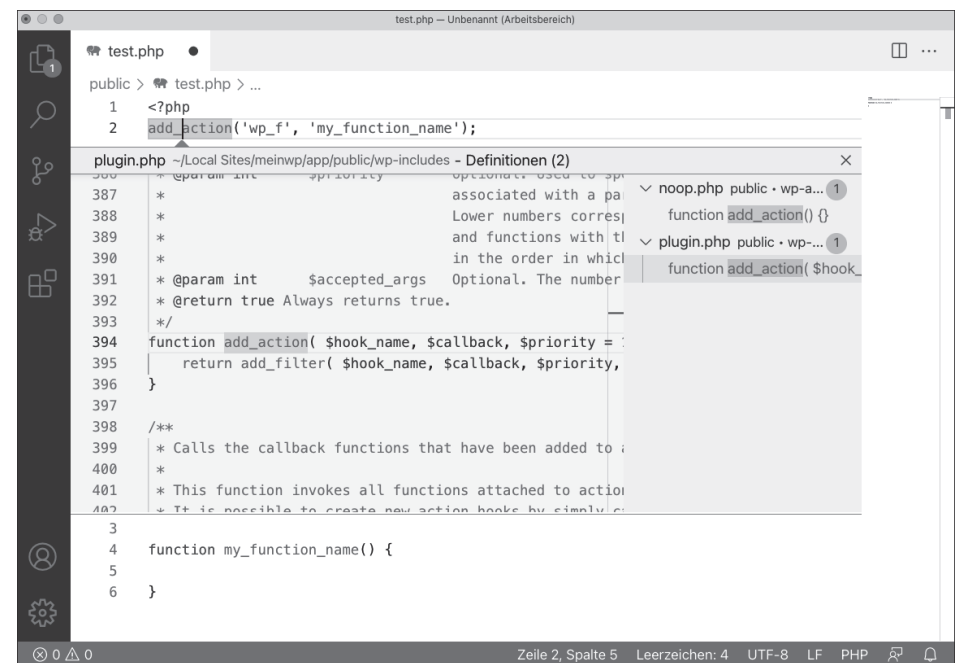


Abbildung 1.19 Quellcode-Navigation: VS Code erkennt den String als Funktionsverweis. Mit einem Klick auf den String gelangen Sie direkt zur Funktionsdefinition.



Jetzt können wir eigentlich loslegen, nicht wahr? Fast. Denn im unmittelbar folgenden Kapitel 2 möchte ich Ihnen die bewährten Konzepte der WordPress-Entwicklung vorstellen.

- ▶ Schriften sollten möglichst leserlich sein. Dazu trägt auch ein guter Kontrast zwischen Text- und Hintergrundfarbe bei.
- ▶ Gehen Sie davon aus, dass die meisten Besucher über ein Smartphone auf Ihre Website kommen werden. Es gilt der Leitsatz: *Mobile first!* Das heißt, dass sich das Layout an alle möglichen Endgeräte und Displaygrößen anpassen können muss (*Responsiveness*).
- ▶ Bauen Sie, wo es geht, weitere Medien (Bilder, Icons, Videos, Animationen) ein, um den inhaltlichen Text zu ergänzen oder um Emotionen zu erzeugen.

5.3 Ein Classic-Theme erstellen

Nun haben Sie sich schon umfangreiches Wissen angeeignet. Sie haben Kenntnis darüber, was ein WordPress-Theme ist und wie Themes von Plugins und Blöcken abzugrenzen sind. Sie wissen, wo Sie Themes finden, und haben dadurch eine große Inspirationsquelle. Bei den beliebtesten Themes können Sie sich Anregungen für Ihr eigenes Theme holen. Falls das nicht reicht, haben Sie in Abschnitt 5.2, »Crashkurs: Gutes Webdesign«, gelesen, wie Sie erprobte Prinzipien einsetzen können, um eine Website zu gestalten.

In diesem Abschnitt geht es an die technische Umsetzung, also an das Schreiben von Code für HTML-, CSS- und PHP-Dateien. Ich lege die empfohlene Verzeichnisstruktur aus Abschnitt 2.7.2 zugrunde. Die zwei wichtigsten Dateien sind dabei

- ▶ *index.php* und
- ▶ *style.css*.

Denn sie sind die einzigen Dateien, die in einem WordPress-Theme unbedingt erforderlich sind. Ohne die *index.php* kann WordPress nichts anzeigen. Es lässt sich gar nicht erst aktivieren. Und ohne die *style.css* weiß WordPress nicht, um welches Theme es sich handelt.

In Ihrer Entwicklungsumgebung der Wahl legen Sie also ein neues Verzeichnis unter */wp-content/themes/* an und erstellen die zwei oben genannten Dateien. In meinem Beispiel sähe das so wie in Abbildung 5.20 aus.

Download des Beispiel-Themes

Das Classic-Theme mit allen Beispielen bis zu Abschnitt 5.6 finden Sie in den Download-Materialien zum Buch.

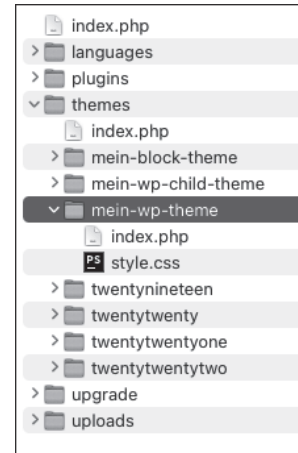


Abbildung 5.20 Meine Theme-Dateien im »themes«-Verzeichnis von WordPress

5.3.1 Die Datei *style.css*

Meiner Meinung nach ist es etwas unglücklich gelöst, dass ausgerechnet eine Stylesheet-Datei so ziemlich die wichtigste Datei eines Themes ist. Zwar existiert das JSON-Format schon seit 1997, aber selbst im Jahr 2003 kam niemand auf die Idee, das Format zur Konfiguration von Webprojekten zu benutzen. Als Entschuldigung dient vielleicht, dass die Funktion `json_decode()` erst im Jahr 2006 Einzug in PHP hielt. Dementsprechend war es technisch und ohne größeren Aufwand nicht möglich, eine solche Datei zu *parsen* (also zu zerlegen und entsprechend einzulesen). Die *theme.json*-Datei hielt erst 2021 Einzug in WordPress. Und zwar mit dem Voranschreiten der Entwicklung des Block-Editors (siehe Abschnitt 5.3.3, »Design und Layout festlegen: theme.json«).

Wir müssen es also hinnehmen, dass die Metadaten eines Themes (trotz *theme.json*) wohl oder übel in einer CSS-Datei angegeben werden. Deswegen muss diese sich auch zwingend im Stammverzeichnis eines Themes befinden. Die Datei dient zwar auch zur Angabe von individuellem CSS, aber dazu gleich mehr.

Theme-Metadaten

Ähnlich wie bei einem Plugin muss sich im Kopfbereich der *style.css* ein Kommentar-Block befinden, der bestimmte Daten über das Theme enthält. Im einfachsten Fall sähe das so wie in Listing 5.3 aus:

```
/*
Theme Name: Mein WP Theme
*/
```

Listing 5.3 Die einfachste Variante von Theme-Metadaten

Wurden die zwei Dateien (*index.php* und *style.css*) erstellt und die *style.css* mit diesem Inhalt befüllt, wird das Theme im WordPress-Backend unter DESIGN • THEMES gelistet und steht zur Aktivierung bereit. Ein Bild wird dort allerdings noch nicht angezeigt. Das können Sie jedoch nachholen, indem Sie eine Datei mit den Namen *screenshot.png*, *.jpg*, *.jpeg*, *.gif* oder *.webp* in das Stammverzeichnis des Themes laden. Die Größe sollte dabei 1200 × 900 px nicht überschreiten. Da es derzeit noch kein ansehnliches Layout gibt, hinterlege ich statt eines Screenshots eine einfache Grafik (siehe Abbildung 5.21).

Nun kann die Style-Datei noch um weitere Metadaten ergänzt werden. Die folgenden Angaben stimmen mit denen bei Plugins überein (siehe Abschnitt 4.2.1 und dort unter der Überschrift »Die Kopfzeilen der Stammdatei«) und können deswegen ebenfalls verwendet werden:

Description	Requires PHP
Version	License
Author	License URI
Author URI	Text Domain
Requires at least	Domain Path

Die zwei Angaben Network und Update URI hingegen funktionieren bei Themes nicht. Darüber hinaus gibt es noch drei weitere Meta-Angaben, die so nur bei Themes, jedoch nicht bei Plugins zum Einsatz kommen können:

Tested up to

Gibt die letzte WordPress-Hauptversion an, bis zu der das Theme getestet wurde.

Tags

Wörter oder Ausdrücke, die es Nutzern ermöglichen, das Theme mithilfe des Tag-Filters zu finden. Eine vollständige Liste der Tags listet das *Theme Review Handbuch* unter <https://make.wordpress.org/themes/handbook/review/required/theme-tags/> auf.

Template

Falls es sich bei dem Theme um ein sogenanntes Child-Theme handelt, muss hier der Slug des Eltern-Themes angegeben werden. Mehr dazu lesen Sie in Abschnitt 5.7, »Wenn Themes Kinder bekommen: Child-Themes«.

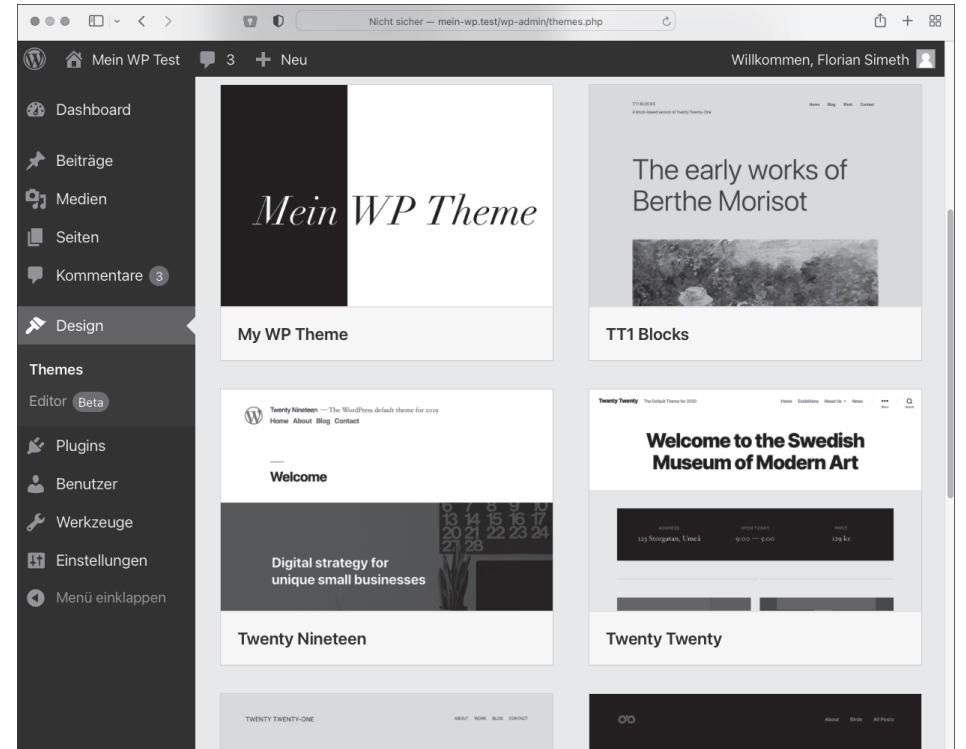


Abbildung 5.21 »Mein WP Theme« wird angezeigt.

Inkonsistenzen zwischen Theme- und Plugin-Meta-Informationen

Der Meta-Wert Tested up to existiert bei Plugins nicht. Zumindest nicht in der Stammdatei. Er existiert aber sehr wohl in der *readme.txt*-Datei¹⁴, die jedem Plugin beigelegt werden muss, wenn es bei wordpress.org hochgeladen wird.

Mir ist nicht bekannt, warum es diese Inkonsistenz gibt. Ein Grund könnte sein, dass beispielsweise der Wert Tested up to im WordPress-Kern nicht vorkommt. Tatsächlich wird nicht geprüft, ob das vorliegende Theme mit der aktuellen WordPress-Version kompatibel ist oder nicht. Es wird jedoch ein Hinweis angezeigt, der angibt, dass ein Theme (oder ein Plugin) mit der derzeit genutzten WordPress-Version nicht getestet wurde oder nicht funktioniert. Und zwar erscheint er, kurz bevor ein Theme oder ein Plugin installiert wird. Insofern kommt der Wert zur Anwendung, aber nicht im Kern, sondern seitens der wordpress.org-API¹⁵.

¹⁴ Siehe <https://developer.wordpress.org/plugins/wordpress-org/how-your-readme-txt-works/>

¹⁵ Siehe https://codex.wordpress.org/WordPress.org_API

Lizenzangaben

Wie bei Plugins sollte auch bei Themes eine Lizenzangabe nicht fehlen. Weitere Informationen dazu finden Sie unter der Überschrift »Inline-Lizenzangaben« in Abschnitt 4.2.1.

Eine Beispiel-style.css-Datei

Ein vollständiger Kopfbereich einer *style.css* könnte also aussehen wie in unserem Beispiel aus Listing 5.4. Die meisten Angaben sehen Sie, wenn Sie im WordPress-Backend unter DESIGN • THEMES Ihr Theme anklicken. Daraufhin öffnet sich das Pop-up-Fenster aus Abbildung 5.22.

```
/**
Theme Name: Mein WP Theme
Theme URI: https://wp-typ.de/themes/mein-wp-theme/
Description: A Starter-Theme for WordPress.
Version: 1.0.0
Requires at least: 5.0.0
Tested up to: 5.4.0
Requires PHP: 7.0.0
Author: floriansimeth
Author URI: https://florian-simeth.de
Tags: blog, one-column, custom-background, custom-colors, custom-logo,
custom-menu, editor-style, featured-images, ...
Text Domain: mein-wp-theme
License: GNUGPLv3
License URI: https://www.gnu.org/licenses/gpl-3.0.html

/**
 * Copyright (C) 2022 Florian Simeth
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see https://www.gnu.org/licenses/.
 */
```

Listing 5.4 »style.css«-Header mit Lizenzangaben

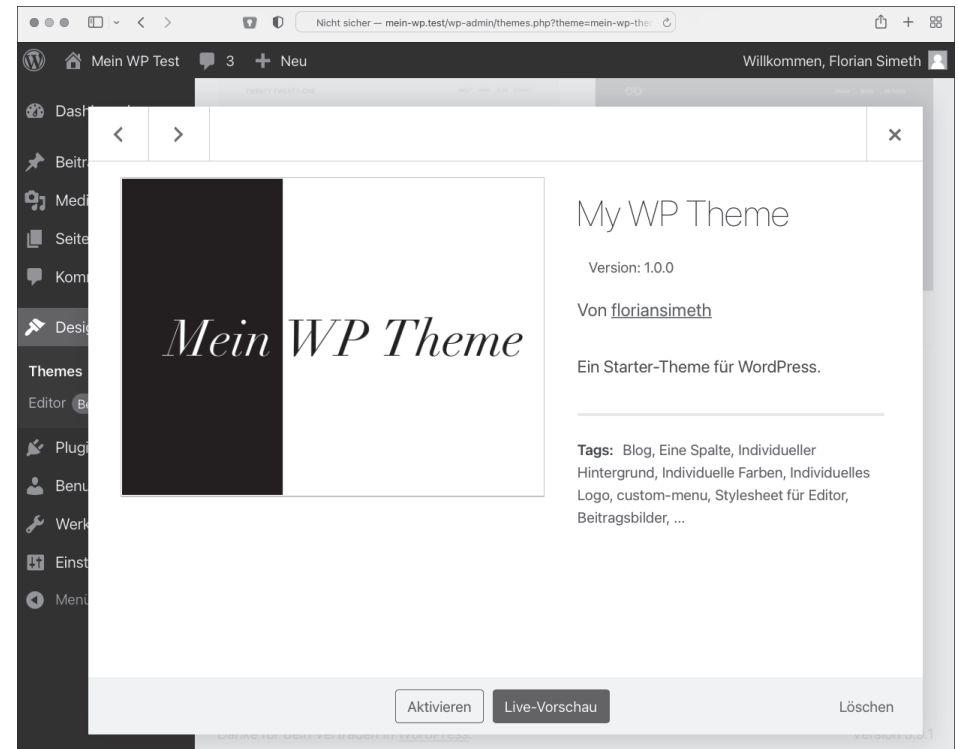


Abbildung 5.22 Theme-Details werden angezeigt, wenn auf sie geklickt wird.

Individuelle Styles

Neben der Angabe der Metadaten zu einem Theme erfüllt die *style.css* natürlich noch eine weitere Aufgabe: Sie dient zum Styling des Themes durch CSS-Styles.

Normalerweise müsste eine Stylesheet-Datei mit `wp_enqueue_style()` eingebunden werden, so wie es in Kapitel 8, »CSS und JavaScript in WordPress«, beschrieben ist. In heutigen Entwicklungsumgebungen kommen allerdings immer öfter alternative Stylesheet-Sprachen wie *Sass* und *SCSS* zum Einsatz. Diese erlauben das Arbeiten mit Variablen, Schleifen und vielen anderen Funktionen, die CSS nicht mitbringt. Die Ausgabe erfolgt dabei über einen *CSS-Präprozessor* in eine separate oder mehrere einzelne Dateien. Den entstandenen Code lässt man in der Regel nicht in die *style.css* schreiben, sondern legt ihn in einem Build-Verzeichnis ab. Letztlich würde Ersteres auch den Kopfbereich der *style.css* überschreiben, und das Theme von WordPress würde nicht mehr erkannt. Aus diesen oder anderen Gründen kann sie (bis auf den Kopfbereich) leer sein. Wir werden später noch sehen, wie weitere CSS-Dateien zum Einsatz kommen, die genutzt werden, um bestimmte Teile des Beispiel-Themes entsprechend zu designen (siehe Abschnitt 5.4, »CSS für Themes«).

5.3.2 Theme-Funktionen festlegen: functions.php

Die *functions.php* ist eine optionale Datei, die im Stammverzeichnis eines Themes abgelegt werden kann. Sie dient dazu, dem Theme individuelle Funktionen hinzuzufügen.

Im Großen und Ganzen verhält sich die Datei wie ein WordPress-Plugin mit dem Unterschied, dass es dort keine Kopfzeilen gibt. Denn diese werden stattdessen in der *style.css* angegeben (siehe Abschnitt 5.3.1).

Tipp!

Auch hier wieder der Hinweis: Wenn Sie neue Funktionen erstellen, die unabhängig vom Aussehen der Website verfügbar sein sollen, empfiehlt es sich, diese stattdessen in ein Plugin zu integrieren. Denn nur dann bleiben die Funktionen auch nach der Deaktivierung des Themes vorhanden.

Was muss in die functions.php?

Es stellt sich also die Frage, wofür eine *functions.php* benötigt wird. Und die Antwort ist ganz einfach: Es gibt unzählige Hooks für Themes. Ein Beispiel wäre `excerpt_length`. Dieser Hook filtert die maximale Anzahl von Wörtern in einem Beitragsauszug. Der Code dafür sollte nicht in ein Plugin, sondern in das Theme.

Außerdem benötigt fast jedes Theme eine bestimmte Grundkonfiguration, die sich mit einer Setup-Funktion festlegen lässt. Sie wird über den Action-Hook `after_setup_theme` getriggert. Dort werden beispielsweise Übersetzungsdateien geladen. Darüber hinaus wird mit der Funktion `add_theme_support()` festgelegt, welchen Funktionsumfang ein Theme haben soll. Vieles stellt WordPress schon direkt ohne zusätzlichen Programmieraufwand zur Verfügung. Vorstellen kann man sich die Funktion deshalb als Ein- oder Ausschaltknopf für bestimmte Eigenschaften. Sie erfahren mehr dazu in Abschnitt 5.3.4 unter »Theme-Funktionen aktivieren und deaktivieren«.

Was gehört also in die *functions.php*?

- ▶ das Laden von Übersetzungsdateien für das Theme
- ▶ das Ein-/Ausschalten bestimmter Features mittels `add_theme_support()`
- ▶ die Registrierung von Navigationsmenüs
- ▶ das Einstellen der Inhaltsbreite des Editors
- ▶ das Registrieren von sogenannten *Sidebars* (Seitenleisten, siehe Abschnitt 5.4, »CSS für Themes«).
- ▶ Funktionen für Hooks, die das Theme betreffen
- ▶ weitere eigene PHP-Funktionen, falls notwendig

Legen wir gemeinsam nun die Datei *functions.php* im Stammverzeichnis des Themes an. Bei mir heißt die Datei also `/wp-content/themes/mein-wp-theme/functions.php`. Als Erstes definiere ich eine Setup-Funktion wie in Listing 5.5 angegeben:

```
<?php
namespace floriansimeth\mein_wp_theme;

function setup_theme() {
    $GLOBALS['content_width'] = 750;

    add_theme_support( 'title-tag' );
    add_theme_support( 'post-thumbnails' );

    register_nav_menus(
        [
            'primary' => 'Hauptmenü',
            'footer'  => 'Fußzeilenmenü',
        ]
    );
}

add_action( 'after_setup_theme', 'floriansimeth\mein_wp_theme\setup_theme' );
```

Listing 5.5 Die »functions.php« des Beispiel-Themes

Zur Variable `$content_width` habe ich ein paar Hinweise im Kasten »Überbleibsel aus alten Zeiten: Die globale Variable `$content_width`« in diesem Abschnitt zusammengefasst.

Wie erwähnt wird durch `add_theme_support()` jeweils ein Schalter umgelegt. `title-tag` bedeutet, dass WordPress das `<title>`-Tag im HTML-Quellcode des Frontends selbstständig erstellen soll. Sie müssen sich also nicht mehr damit beschäftigen. Ähnliches gilt für `post-thumbnails`. Es schaltet die Möglichkeit frei, dass ein Artikelbild mit einem Beitrag und mit einer Seite verknüpft werden kann. Eine Liste von Schaltern finden Sie in Tabelle 5.8 weiter unten in Abschnitt 5.3.4.

Die Funktion `register_nav_menus()` registriert schließlich zwei Menüs, die später in den Template-Dateien Anwendung finden werden (siehe dazu Abschnitt 5.6, »Menüs erstellen«).

Wir werden die *functions.php*-Datei gleich noch einmal benötigen. Zuvor möchte ich Ihnen aber zeigen, wie Sie das Design und Layout bestimmter Elemente festlegen können.

Überbleibsel aus alten Zeiten: Die globale Variable `$content_width`

Seit Version 2.6 von WordPress existiert die globale Variable `$content_width`, die an diversen Stellen im WordPress-Kern genutzt wird. Damit lässt sich die maximale Breite für bestimmte Inhalte im Theme festlegen, zum Beispiel für oEmbeds und Bilder, die zu Beiträgen hinzugefügt werden. Da der Wert noch immer Anwendung findet, kann er nicht einfach ignoriert werden. Immerhin wird er für den neuen Block-Editor nicht mehr genutzt und verschwindet daher irgendwann aus dem Kern.

5.3.3 Design und Layout festlegen: `theme.json`

Die `theme.json`-Datei ist eine Konfigurationsdatei für Theme- und Block-Styles. Damit gehen wir immer weiter in Richtung Block-Editor. Die Möglichkeit, eine solche Datei erstellen zu können, gibt es erst seit Version 5.8 von WordPress. In älteren WP-Versionen wird die Datei schlicht ignoriert. Es sei denn, das Gutenberg-Plugin wird installiert (siehe dazu die Informationen im Kasten »Gutenberg und der Block-Editor« in diesem Abschnitt).

Die Datei ist nur dann wichtig, wenn der Einsatz des Block-Editors explizit erwünscht ist. Denn mit ihr lassen sich viele Einstellungen und das Styling einzelner Blöcke einstellen. Wird der Classic-Editor¹⁶ oder ein Page-Builder eines Drittherstellers benutzt, wird die Datei jedoch nicht benötigt. Ich gehe in diesem Beispiel davon aus, dass der Block-Editor genutzt wird. Immerhin ist er mittlerweile der Standard.

Obwohl eine `theme.json`-Datei für ein Classic-Theme nicht unbedingt benötigt wird, können Sie trotzdem damit arbeiten. Achten Sie nur darauf, dass keine doppelten CSS-Regeln erzeugt werden. Denn aus der Konfiguration der Datei entstehen automatisch CSS-Regeln, wie Sie in Abschnitt 5.8.2, »Hauptkonfiguration mit der `theme.json`«, noch sehen werden.

Darüber hinaus sollten Sie sich bewusst sein, dass einige der Konfigurationsmöglichkeiten der Datei möglicherweise einen oder mehrere Aufrufe von `add_theme_support()` ersetzen. Zum Beispiel lässt sich in der Datei eine Farbpalette hinterlegen, die den Aufruf von `add_theme_support('editor-color-palette', ...)` ersetzt. Im Fall, dass beide vorhanden sind, erhält die Konfiguration aus der `theme.json`-Datei den Vorrang.

Gutenberg und der Block-Editor

Gutenberg ist der interne Codename für den Block-Editor. Da gegenwärtig sehr intensiv an ihm gearbeitet wird, gibt es ein Plugin dazu. Sie finden es unter <https://>

¹⁶ Zum Download unter: <https://wordpress.org/plugins/classic-editor/>

wordpress.org/plugins/gutenberg/. Zwar ist der Editor bereits im Kern von WordPress integriert, aber mit dem Plugin erhalten Sie stets die aktuellen Features, die allerdings nicht immer final sind. Es kann also sein, dass das Plugin Funktionen enthält, die später wieder entfernt werden. Das kann beim integrierten Block-Editor nicht passieren. Das Plugin ist also nichts anderes als ein Test-Plugin, das immer den aktuellen Stand der Entwicklung bereithält.

Es gibt derzeit fünf Bereiche, in die die Datei, die nur Schlüssel-Wert-Paare enthalten kann, unterteilt ist. Die zwei Bereiche, mit denen wir uns im Folgenden beschäftigen werden, sind `settings` und `styles`. Zusätzlich muss eine Versionsnummer angegeben werden. Aktuell befinden wir uns in Version 2. Der initiale Code einer `theme.json` sieht also so aus wie in Listing 5.6:

```
{
  "version": 2,
  "templateParts": [],
  "patterns": [],
  "customTemplates": [],
  "settings": {},
  "styles": {}
}
```

Listing 5.6 »`theme.json`«-Datei

Die Angaben zu `customTemplates` und `templateParts` ermöglichen die Konfiguration von Template-Dateien für reine Block-Themes. `patterns` dient dazu, Block-Vorlagen aus dem Block-Verzeichnis von wordpress.org hinzuzufügen (siehe »Vorlagen (Patterns)« in Abschnitt 5.8.2).

Innerhalb von `settings` und `styles` gibt es zahlreiche Möglichkeiten, wie sich bestimmte Dinge des Themes steuern und stylen lassen. wordpress.org stellt dafür eine JSON-Schema-Datei zur Verfügung, die sich in die meisten Entwickler-Tools (wie PhpStorm und Visual Studio Code) laden lässt. Oft geschieht dies vollautomatisch, weil die Programme die Schemas von der Website <https://SchemaStore.org> herunterladen.¹⁷ Das ist sehr praktisch, weil Sie sich in den Programmen dann Vorschläge machen lassen können (siehe Abbildung 5.23).

Die Schema-Datei beschreibt, welche Einstellmöglichkeiten es gibt. Die jeweils aktuelle Fassung finden Sie unter <https://schemas.wp.org/trunk/theme.json>.

¹⁷ Das passiert bei PhpStorm automatisch. Für Visual Studio Code gibt es hier eine Anleitung: <https://make.wordpress.org/themes/2021/11/30/theme-json-schema/>

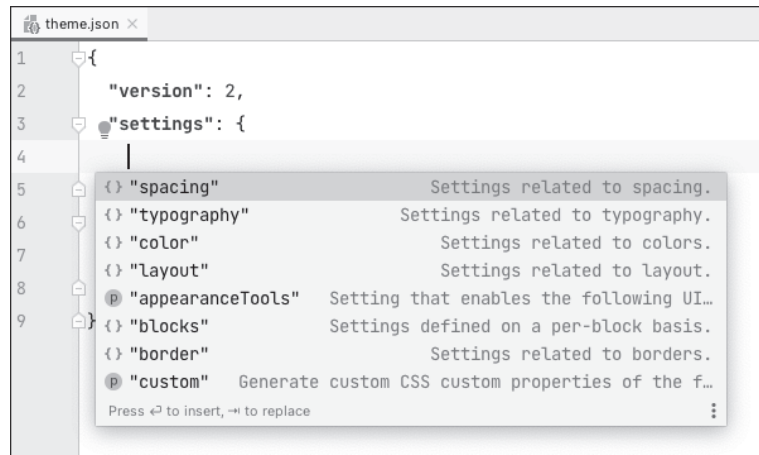


Abbildung 5.23 PhpStorm zeigt mögliche Werte für die »theme.json«-Datei

Listing 5.7 zeigt einen Ausschnitt der derzeit potenziellen Werte der *theme.json*-Datei. Ein vollständiges Beispiel mit allen möglichen Blöcken finden Sie bei den Download-Materialien zu diesem Abschnitt. Wie die Datei konfiguriert wird, erfahren Sie in Abschnitt 5.8.2, »Hauptkonfiguration mit der theme.json«. In Listing 5.8 ist ein kleiner Ausschnitt der Konfiguration für das Beispiel-Theme zu sehen. Abbildung 5.24 zeigt vorweggegriffen einen Screenshot von einem Beitrag, der zu diesem Zeitpunkt nicht funktioniert, weil dafür noch kein Template existiert. Ich werde in den folgenden Abschnitten noch genauer darauf eingehen. Nichtsdestotrotz sind beispielhaft die verschiedenen Schriftgrößen zu sehen, so wie ich sie in der *theme.json* konfiguriert habe.

```
{
  "version": 2,
  "settings": {
    "typography": {
    },
    "color": {
    },
    "border": {
    },
    "spacing": {
    },
    "blocks": {
      "core/paragraph": {
```

```
    },
    "core/separator": {
    }
    ...
  },
  "layout": {
  },
  "appearanceTools": false,
  "custom": true
},
"styles": {
  "blocks": {
    "core/paragraph": {
    },
    "core/separator": {
    }
    ...
  },
  "spacing": {
  },
  "border": {
  },
  "color": {
  },
  "typography": {
  },
  "elements": {
    "link": {
    },
    "h1": {
    },
    ...
    "h6": {
    }
  }
},
"customTemplates": [
  {
    "postTypes": [],
    "name": "",
```

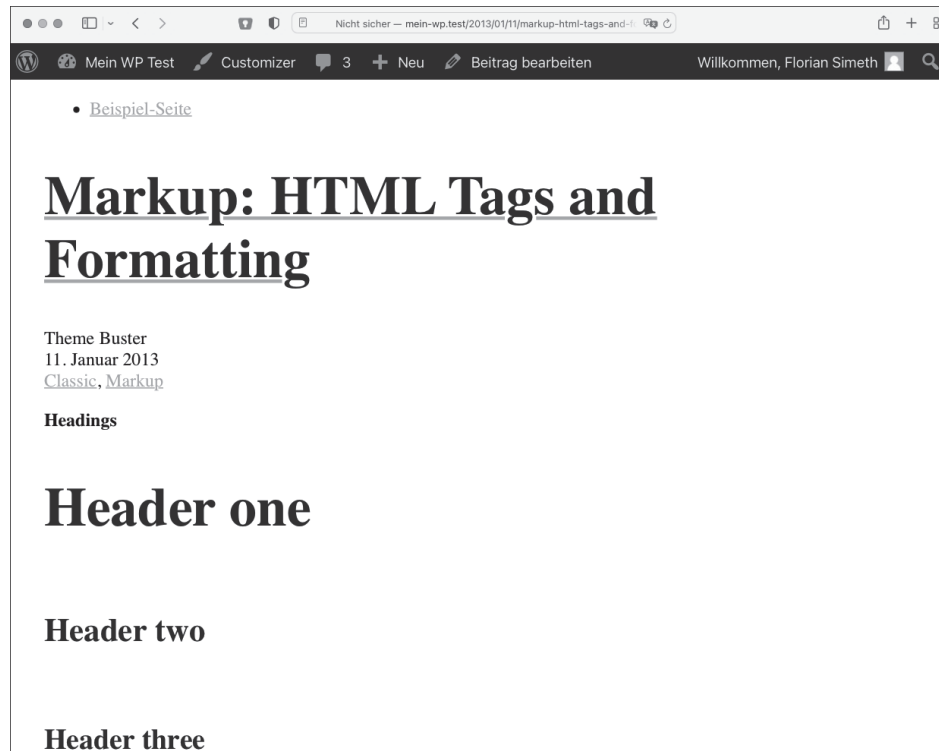



Abbildung 5.24 Gut zu erkennen: die einzelnen Schriftgrößen, wie sie in der »theme.json« angegeben wurden

5.3.4 Einstellungen für Themes festlegen

Nun kommen wir zur Themes-API. Wie bei fast allen APIs in WordPress handelt es sich nicht um eine direkte Schnittstelle, sondern vielmehr um eine Sammlung von Funktionen. Tatsächlich existiert nicht einmal der Begriff *Themes-API*. Aber ich habe sie so genannt, um alle Funktionen zusammenzufassen, die für die Theme-Entwicklung relevant sind. Meines Erachtens spaltet sie sich in drei Teile auf:

- ▶ in die *Theme-Customize-API*,
- ▶ in die *Theme-Modification-API* (beide existieren in WordPress) und
- ▶ in weitere Funktionen, wie z. B. die zuvor oft erwähnte Funktion `add_theme_support()`

All diese Funktionen werden dazu verwendet, Daten innerhalb des Themes zu verwalten oder Einstellungen zu aktivieren und zu deaktivieren, während weitere Funktionen existieren, die rein für das Frontend (also für die Anzeige von Daten) zuständig sind. Hier wären unter anderem folgende Themen genannt, denen ich einzelne Abschnitte gewidmet habe:

- ▶ Abschnitt 5.3.7, »Funktionen für Templates und der Loop«
- ▶ Abschnitt 5.3.8, »Konditionale Abfragen«
- ▶ Abschnitt 5.5, »Seitenleisten und Widgets erstellen«
- ▶ Abschnitt 5.6, »Menüs erstellen«

Theme-Optionen speichern: Die Theme-Modification-API

Die älteste API für Theme-Einstellungen ist die Theme-Modification-API. Sie kann von Theme-Autoren verwendet werden, um Änderungen an Themes als WordPress-Optionen zu speichern und abzurufen. Sie ist relativ einfach gestrickt, denn sie besteht aus lediglich zwei praktischen Funktionen (siehe Listing 5.9):

```
<?php
function set_theme_mod( string $name, mixed $value ): bool {
    ...
}
function get_theme_mod( string $name, mixed $default = false ): mixed {
    ...
}
```

Listing 5.9 Funktionen zum Speichern und Lesen von Theme-Einstellungen

Beide nutzen die Options-API in WordPress (siehe Abschnitt 6.3.1), weswegen sie ähnlich aufgebaut sind. Sie erwarten jeweils zwei Parameter und haben einen Rückgabewert:

`set_theme_mod()`:

- ▶ `$name`: Eindeutiger Name der Einstellung
- ▶ `$value`: Ein Wert, der gespeichert werden soll
- ▶ Gibt einen booleschen Wert zurück, der angibt, ob die Einstellung ordnungsgemäß in der Datenbank gespeichert werden konnte.

`get_theme_mod()`:

- ▶ `$name`: Wieder der eindeutige Name
- ▶ `$default`: Ein Standardwert, falls kein Wert in der Datenbank hinterlegt ist
- ▶ Zurückgegeben wird der zuvor gespeicherte Wert.

Aufgrund der Ähnlichkeit zur Options-API gehe ich nicht näher darauf ein, da diese in Abschnitt 6.3.1, »Werte speichern und abrufen mit der Options-API«, noch ausführlich behandelt wird. Sie können die dort gezeigten Beispiele nehmen und statt `get_option()` und `set_option()` die jeweilige `..._theme_mod()`-Funktion nutzen.

Der Customizer: Ein Theme vom Benutzer anpassen lassen

Die Customize-API ist ein Framework für die Live-Vorschau aller Umgestaltungen an WordPress. Auch sie speichert Änderungen in die Datenbank, erlaubt aber auch die Anpassung verschiedener Aspekte eines Themes und der Website – von Farben und Layouts bis hin zu Widgets, Menüs und mehr.

Der Customizer ist der klassische Weg, um Optionen zu einem Theme hinzuzufügen, obwohl auch diese API immer mehr vom Block-Editor und seinen Zusatzfunktionen verdrängt werden wird.

Im WordPress-Backend unter dem Menü DESIGN • CUSTOMIZER lässt sich der Customizer öffnen. Nach dem Klick wird die Startseite der Website angezeigt (2 in Abbildung 5.25). Am linken Rand befindet sich eine Seitenleiste (1) und darin der Customizer mit all seinen Einstellmöglichkeiten (4). Freilich variiert die Menge an Einstellungen je nach Theme. Über das Stift-Symbol (5) gelangen Sie direkt zur entsprechenden Einstellmöglichkeit im Customizer.

Verändern Sie eine Einstellung, so wird diese live im rechten Teil des Bildschirms angezeigt. Die Änderungen werden allerdings erst nach einem Klick auf VERÖFFENTLICHEN (3) übernommen und sind dann für alle Nutzer der Website sichtbar.

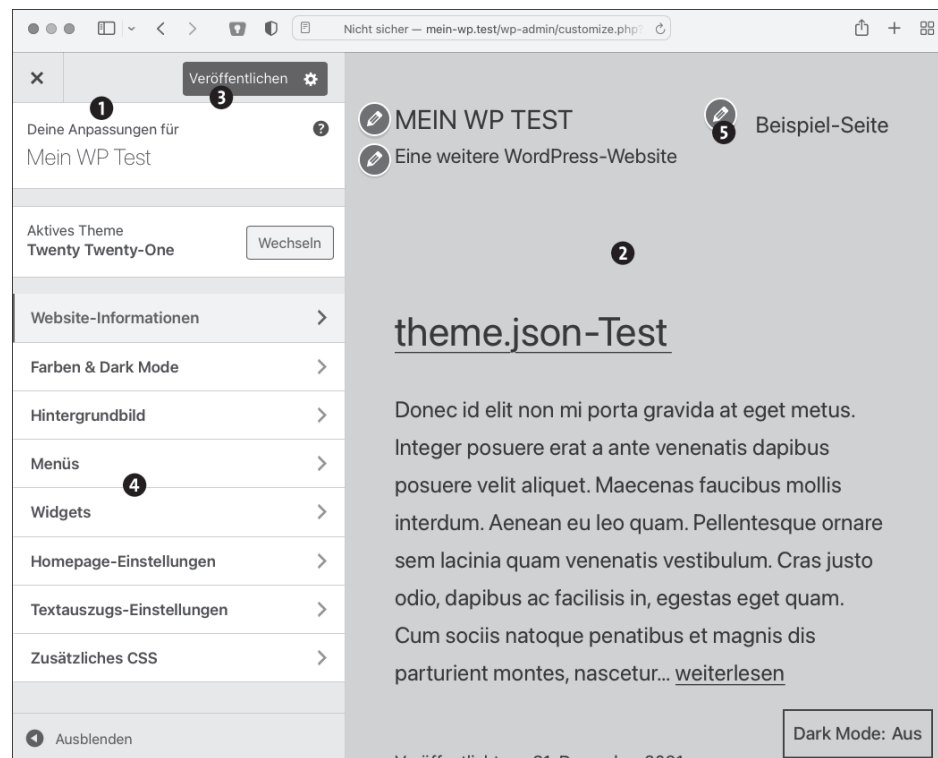


Abbildung 5.25 Geöffneter Customizer

Interessant ist, dass sich über das Einstellungsrad (3) beim Button eine Änderung terminieren, aber auch teilen lässt. Sie erhalten dazu einen Link zur entsprechenden Unterseite, die gerade geöffnet ist. Jemand mit dem Link kann sich die Änderungen dann anzeigen lassen, obwohl sie noch nicht final veröffentlicht wurden. Möglich ist dies, weil die API die Zustände in sogenannten *Customizer Objects* speichert. Diese sind PHP-Objekte und funktionieren, weil die API objektorientiert entwickelt wurde. Abbildung 5.26 zeigt, dass es vier Haupttypen von Customizer-Objekten gibt:

1. Panels,
2. Sections,
3. Settings und
4. Controls.

Settings assoziieren UI-Elemente (*Controls*) mit Einstellungen, die in der Datenbank gespeichert werden. *Sections* sind UI-Container für Controls, um deren Organisation zu verbessern. *Panels* sind Container, in denen mehrere Sections gruppiert werden können. Abbildung 5.26 zeigt auch das Zusammenspiel von PHP im Backend und JavaScript im Frontend.

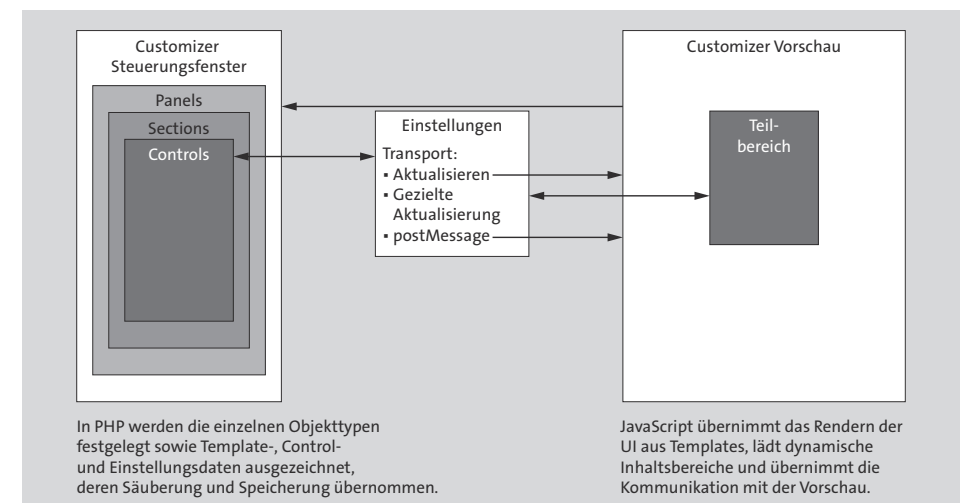


Abbildung 5.26 Zusammenstellung der einzelnen Objekte in der Customizer-API¹⁸

Im Folgenden zeige ich an einem Beispiel, wie Sie eine Einstellung zum Customizer hinzufügen können. Dazu wird eine neue Datei im Theme-Verzeichnis erstellt: `/includes/customizer.php`. Die Datei wird, wie in Listing 5.10 gezeigt, in der `functions.php` eingebunden. Den Inhalt der `customizer.php`-Datei sehen Sie in Listing 5.11.

¹⁸ Quelle: Customize-Object-Hierarchy-Graphic von WordPress.org, lizenziert unter GPLv2. <https://developer.wordpress.org/themes/customize-api/customizer-objects> Übersetzung.


```
<?php
require_once __DIR__ . '/includes/customizer.php';
```

Listing 5.10 Auszug aus der Datei »functions.php«

```
<?php
namespace floriansimeth\mein_wp_theme;

add_action( 'customize_register', 'floriansimeth\mein_wp_theme\custom_
customizer_settings' );

function custom_customizer_settings( \WP_Customize_Manager $manager ) {
    $my_panel = $manager->add_panel(
        'mwp',
        [
            'title'      => 'Mein WP Theme Einstellungen',
            'description' => 'Optionen für Beiträge',
        ]
    );

    $my_section = $manager->add_section(
        'mwp_post_settings',
        [
            'panel'      => $my_panel->id,
            'title'      => 'Beitragseinstellungen',
            'description' => 'Optionen für Beiträge',
        ]
    );

    $my_setting = $manager->add_setting(
        'mwp_post_settings_display_author',
        [
            'default'      => false,
            'sanitize_callback' => function ( $value, $setting ) {
                return (bool) $value;
            },
        ]
    );

    $my_control = $manager->add_control(
        $my_setting->id,
        [
            'type'          => 'checkbox',
```

```
'section'      => $my_section->id,
'label'        => 'Autor anzeigen?',
'description'  => 'Autor bei den Beiträgen anzeigen?',
'active_callback' => function () {
    return is_single();
}
] );
}
```

Listing 5.11 Die »customizer.php«-Datei

Über den Hook `customize_register` wird eine Funktion registriert. Sie ist danach zuständig für das Hinzufügen des Panels, der Section, der Settings und schließlich des Controls. Das geschieht nacheinander durch den Aufruf der entsprechenden Methoden der Klasse `WP_Customize_Manager`, deren Objektinstanz mit dem Hook zurückgegeben wird. Am Ende sieht der Customizer so aus wie in Abbildung 5.27 gezeigt. Dort ist von links nach rechts abgebildet, wie der Benutzer über den Menüpunkt **MEIN WP THEME EINSTELLUNGEN • BEITRAGSEINSTELLUNGEN • OPTIONEN FÜR BEITRÄGE** zur Option **AUTOR ANZEIGEN?** gelangt.

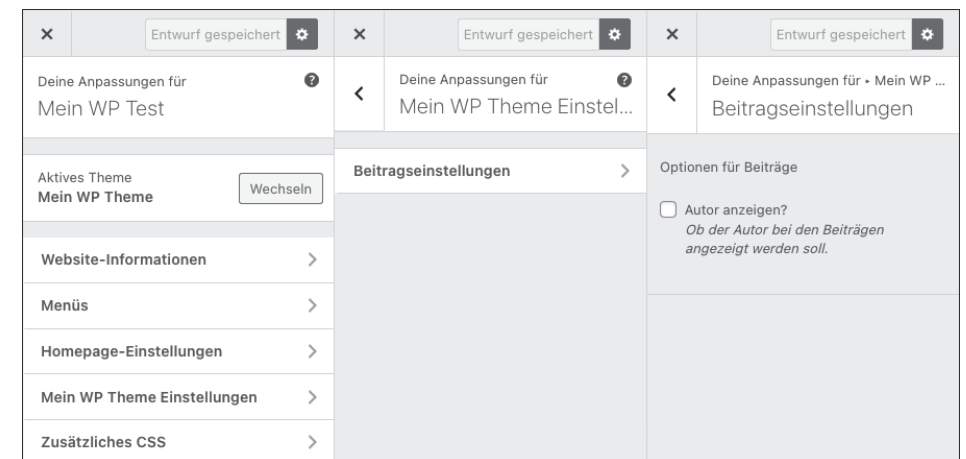


Abbildung 5.27 Von links nach rechts sehen Sie hier die einzelnen Fenster bis hin zur eigentlichen Option.

Hinweis

Das neue Panel ist erst sichtbar, wenn Sie sich auf einer Beitragsseite befinden (siehe die folgenden Erklärungen). Wenn im Customizer die Startseite angezeigt wird, wechseln Sie am besten von dort zu einem einzelnen Beitrag, um sie zu sehen.

Mit der Methode `add_panel()` (siehe Listing 5.12) wird als Erstes das Panel zum Customizer hinzugefügt:

```
<?php
public function add_panel(
    WP_Customize_Panel|string $id,
    array $args = array(
        int $priority = 160,
        string $capability = 'edit_theme_options',
        string|string[] $theme_supports,
        string $title,
        string $description,
        string $type,
        callable $active_callback
    )
): WP_Customize_Panel {
    ...
}
```

Listing 5.12 Funktionsbeschreibung von »`add_panel()`«

- ▶ Mit `$id` wird eine eindeutige ID erwartet. Das ist in der Regel ein String, kann aber auch ein `WP_Customize_Panel`-Objekt sein.
- ▶ `$args` ist ein Array mit zusätzlichen Angaben:
 - `$priority` ist eine Ganzzahl, die die Sortierung bestimmt. Je höher die Nummer ist, desto weiter unten wird das Panel angezeigt.
 - `$capability` ist das Recht, das ein Benutzer haben muss, damit er die Einstellung sehen kann. Mehr dazu erfahren Sie in Kapitel 16, »Benutzer- und Rechte-management«.
 - `$theme_supports` ist ein einzelner String oder ein Array von Strings mit Angaben zu Theme-Funktionen, die vorhanden sein müssen, damit das Panel angezeigt wird.
 - `$title` ist der Panel-Titel.
 - `$description` ist eine Beschreibung des Panels. Der Text wird angezeigt, wenn auf das Fragezeichen geklickt wird.
 - `$type` ist der Typ des Panels. Er muss nur angegeben werden, wenn eine eigene Subklasse eines Panels erstellt werden soll.
 - `$active_callback` gibt die Funktion an, die überprüft, wann das Panel sichtbar sein soll. Wird nichts angegeben, ist das Panel immer sichtbar (siehe dazu auch Abschnitt 5.3.8, »Konditionale Abfragen«).

Eine Section wird mit der Methode `add_section()` erstellt (siehe Listing 5.13):

```
<?php
public function add_section(
    WP_Customize_Section|string $id,
    array $args = array(
        int $priority = 160,
        string $panel = '',
        string $capability = 'edit_theme_options',
        string|string[] $theme_supports,
        string $title,
        string $description,
        string $type,
        callable $active_callback,
        bool $description_hidden = false
    )
): WP_Customize_Section {
    ...
}
```

Listing 5.13 Funktionsbeschreibung von »`add_section()`«

Sie erwartet ebenfalls eine eindeutige ID als ersten Parameter. Als zweiter wird ein Array mit weiteren Argumenten übergeben. Die Liste ist fast identisch mit der aus der Methode `add_panel()`. Jedoch gibt es noch zwei zusätzliche Argumente:

- ▶ `$panel` erwartet die ID des Panels, zu dem die Section hinzugefügt werden soll. Es kann die ID des eigenen oder eines bereits bestehenden Panels angegeben werden.
- ▶ Wird `$description_hidden` auf `true` gesetzt, so wird die Beschreibung hinter einem Fragezeichen-Symbol versteckt. Ein Klick darauf gibt sie dann frei. Bei `false` wird sie sofort angezeigt.
- ▶ Nun benötigen wir noch eine Einstellung (siehe Listing 5.14):

```
<?php
public function add_setting(
    WP_Customize_Setting|string $id,
    array $args = array(
        string $type = 'theme_mod',
        string $capability = 'edit_theme_options',
        string|string[] $theme_supports,
        string $default = '',
        string $transport = 'refresh',
        callable $validate_callback,
    )
): WP_Customize_Setting {
    ...
}
```

```

        callable $sanitize_callback,
        callable $sanitize_js_callback,
        bool $dirty
    )
): WP_Customize_Setting {
    ...
}

```

Listing 5.14 Funktionsbeschreibung von »add_setting()«

Ebenso wie bei den vorangegangenen Methoden wird hier zuerst eine eindeutige ID benötigt. Danach folgt die Argumentenliste. Einige Argumente sind ähnlich wie bei den zuvor genannten Methoden. Ich liste sie daher nicht mehr separat auf.

- ▶ `$type` gibt den Speicherort an. Wählen Sie zwischen `theme_mod` oder `option` (siehe dazu auch den Abschnitt »Theme-Optionen speichern: Die Theme-Modification-API« weiter oben).
- ▶ `$default` ist der Standardwert der Einstellung.
- ▶ `$transport` ist eine Option für die Darstellung der Live-Vorschau von Änderungen im Customizer. Die Verwendung von `refresh` macht sie sichtbar, indem die gesamte Vorschau neu geladen wird. Mit `postMessage` kann ein benutzerdefiniertes JavaScript das Laden umgehen.
- ▶ `$validate_callback` und `$sanitize_callback` sind Sicherheitsfunktionen. Sie stellen sicher, dass nur bestimmte Werte angenommen werden und in die Datenbank gelangen (lesen Sie dazu auch Abschnitt 7.1, »WordPress-Projekte absichern«).
- ▶ `$sanitize_js_callback` erwartet eine Callback-Funktion, die zur Konvertierung eines PHP-Einstellungswerts dient, falls dieser nicht in das JSON-Format umgewandelt werden kann.
- ▶ `$dirty` wird verwendet, um sicherzustellen, dass eine Einstellung beim Laden des Customizers vom Einstellungskasten an die Vorschau gesendet wird. Normalerweise wird sie nur dann mit der Vorschau synchronisiert, wenn sie geändert wurde. Mit `$dirty = true` wird die Einstellung von Anfang an übertragen.

Zum Schluss benötigt die Einstellung noch ein Formularfeld. Dies gelingt mit der Methode `add_control()` aus Listing 5.15:

```

<?php
public function add_control(
    WP_Customize_Control|string $id,
    array $args = array(
        int $instance_number,
        WP_Customize_Manager $manager,

```

```

        string $setting = 'default',
        array $settings,
        string $section = '',
        string $type = 'text',
        string $label = '',
        array $choices = [],
        array $input_attrs = [],
        bool $allow_addition = false,
        callable $active_callback,
        string $description = '',
        string $capability,
        int $priority = 10
    )
): WP_Customize_Control {
    ...
}

```

Listing 5.15 Funktionsbeschreibung von »add_control()«

Es gilt:

- ▶ `$id` ist die eindeutige ID des Feldes. Falls die gleiche ID wie bei `add_setting()` benutzt wird, so kann der Wert automatisch abgerufen werden.
- ▶ `$args` sind weitere Parameter:
 - `$instance_number` ist die Reihenfolge, in der diese Instanz im Verhältnis zu anderen Instanzen erstellt wurde.
 - `$manager` ist die `WP_Customize_Manager`-Instanz. Sie wird benötigt, weil der Wert der Settings ausgelesen werden muss. Sie muss nicht angegeben werden, wenn `$id` die gleiche ID hat wie bei `add_setting()`.
 - `$setting` ist die ID, die bei `add_setting()` verwendet wurde und die zu diesem Control gehört. Damit wird dann der Wert aus der Datenbank gelesen.
 - Benötigt das Control mehrere Werte von der Datenbank, lässt sich ein Array mit Setting-Ids übergeben, nämlich das Argument `$settings`.
 - Mit `$section` wird festgelegt, zu welcher Section das Control gehört.
 - `$type` ist der HTML-Typ des Formulars. Er kann die Werte `text`, `checkbox`, `text-area`, `radio`, `select` und `dropdown-pages` annehmen. Feldtypen wie `email`, `url`, `number`, `hidden` und `date` werden implizit unterstützt.
 - `$label` ist das HTML-Label zum Formular-Element, das erzeugt wird.
 - `$choices` ist ein Array mit möglichen Werten, die für die HTML-Elemente `radio` und `select` benötigt werden. Dabei ist der Schlüssel des Arrays der Wert und der Wert des Arrays das Label für das erzeugte HTML-Element.

- `$input_attrs` ist ein Array für benutzerdefinierte HTML-Attribute der Formularfelder. Es findet keine Anwendung, wenn bei `$type` einer der folgenden Werte angegeben wurde: `checkbox`, `radio`, `select`, `textarea` oder `dropdown-pages`.
- Falls unter `$type` der Wert `dropdown-pages` angegeben wurde, so können Sie mit `$allow_addition = true` festlegen, ob weitere Inhalte hinzugefügt werden dürfen.
- `$capability`, `$priority`, `$description` und `$active_callback` sollten selbsterklärend sein (Sie finden die Beschreibungen weiter oben).

Nun können Sie den Customizer eigenständig mit Einstellungen befüllen, die von Ihrem Theme benötigt werden. Abgerufen werden die Werte dann je nach dem Namen, den Sie während `add_setting()` gewählt haben. Wurde dort das Argument `$type` wie im Beispiel auf `theme_mod` gesetzt, so erfolgt der Abruf wie in Listing 5.16 dargestellt. Wurde stattdessen `option` gewählt, so erfolgt der Abruf wie in Listing 5.17. Wie das im Beispiel funktioniert, erfahren Sie in Abschnitt 5.3.7 und dort unter der Überschrift »Meta-Angaben«.

```
<?php
$display_author = get_theme_mod(
    'mwp_post_settings_display_author',
    false
);
```

Listing 5.16 Auslesen einer Theme-Einstellung

```
<?php
$display_author = get_option(
    'mwp_post_settings_display_author',
    false
);
```

Listing 5.17 Auslesen einer Option

Wie Sie sehen, ist der Customizer ein sehr mächtiges Werkzeug und ging bei seiner Einführung schon ein klein wenig in Richtung *Full-Site-Editing*. So richtig gelang dieser Schritt aber erst mit dem Block-Editor, weshalb der Customizer in Zukunft an Bedeutung verlieren wird.

Theme-Funktionen aktivieren und deaktivieren

Nun kommen wir zum letzten Teil der Themes-API, und zwar zur Funktion `add_theme_support()`. Diese schon mehrfach erwähnte Funktion dient während der Initialisierungsphase dazu, WordPress mitzuteilen, welche Funktionen das Theme unterstützt. Sehen Sie sich dazu noch einmal Listing 5.5 aus Abschnitt 5.3.2, »Theme-

Funktionen festlegen: `functions.php`«, an. In WordPress sind sehr viele Funktionen eingebaut, die das Theme sofort verwenden kann. Im Beispiel nutzten wir lediglich zwei von ihnen.

```
<?php
function add_theme_support( string $feature, mixed ...$args ): void|bool {
    ...
}
```

Listing 5.18 Funktionsbeschreibung von »`add_theme_support()`«

In Listing 5.18 ist `$feature` der Name der zu unterstützenden Funktion und `$args` ist ein Platzhalter für weitere Argumente. WordPress prüft mit `get_theme_support()`, ob der Schalter umgelegt wurde und damit eine Funktion unterstützt wird. Mögliche Werte sind in Tabelle 5.8 dargestellt.

admin-bar	align-wide	automatic-feed-links
core-block-patterns	custom-background	custom-header
custom-line-height	custom-logo	customize-selective-refresh-widgets
custom-spacing	custom-units	dark-editor-style
disable-custom-colors	disable-custom-font-sizes	editor-color-palette
editor-gradient-presets	editor-font-sizes	editor-styles
featured-content	html5	menus
post-formats	post-thumbnails	responsive-embeds
starter-content	title-tag	wp-block-styles
widgets	widgets-block-editor	

Tabelle 5.8 Mögliche Werte von »`add_theme_support()`«

Wenn Sie wissen möchten, was jedes einzelne dieser Features im Detail macht, empfehle ich Ihnen eine Suche im WordPress-Kern. Für den Wert `custom-header` können beispielsweise mehrere Argumente übergeben werden, wie Listing 5.19 zeigt. Der Abruf der Textfarbe erfolgt mit der Funktion `get_header_textcolor()` (siehe Listing 5.20). Beachten Sie, dass einige dieser Schalter Einstellungsfelder im Customizer aktivieren. Abbildung 5.28 zeigt blau hervorgehoben die Option zum Verwalten des Headerbildes. Im Menüpunkt FARBEN gibt es dann zusätzlich die Möglichkeit, die Textfarbe im Header anzupassen.

In Abschnitt 5.3.7 und dort unter »Der Header« sehen Sie im Detail, wie mit dem Custom-Header gearbeitet wird.

```
<?php
$args = array(
    'default-image'      => get_template_directory_uri()
                        . 'public/images/default-background.jpg',
    'default-text-color' => '000',
    'width'              => 1000,
    'height'             => 250,
    'flex-width'         => true,
    'flex-height'        => true,
);
add_theme_support( 'custom-header', $args );
```

Listing 5.19 Zusatzargumente für einen individuellen Kopfbereich

```
<?php
function get_header_textcolor() {
    return get_theme_mod(
        'header_textcolor',
        get_theme_support( 'custom-header', 'default-text-color' )
    );
}
```

Listing 5.20 Beispiel aus dem WordPress-Kern

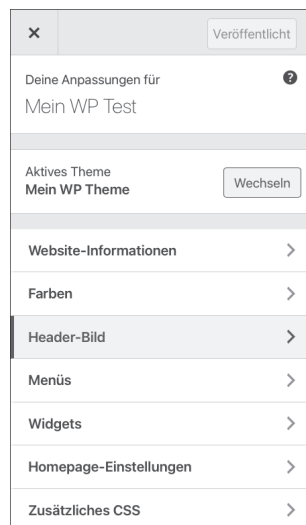


Abbildung 5.28 Wird die Custom-Header-Funktion aktiviert, so gibt es eine neue Option »Header-Bild« im Customizer.

5.3.5 Das WordPress-Template-System

Sie haben jetzt gelernt, wie Sie Theme-Metadaten angeben, und Sie wissen, was in eine *functions.php*-Datei gehört, und können Daten des Themes verändern. Letzteres geschieht mit dem Customizer oder der *theme.json*-Datei. In diesem Abschnitt geht es nun darum, mit diesen Daten zu arbeiten. Wir sehen uns das Template-System in WordPress an, und Sie lernen, wie und wo welche Inhalte angezeigt werden.

Durch den Einblick, den ich bereits geben konnte, wissen Sie, dass ein Theme aus mindestens zwei Dateien bestehen muss: aus der *style.css* und aus einer *index.php*. Ansonsten lässt es sich nicht aktivieren. Sie werden aber rasch feststellen, dass es mit diesen zwei Dateien nicht getan ist. Denn WordPress wird Sie über eine PHP-Warnung (siehe Kapitel 11, »Fehler finden und beheben: Das Debugging«) schnell darauf hinweisen, dass weitere Dateien fehlen – etwa eine *header.php* und eine *footer.php*. Darüber hinaus existieren noch weitere Template-Dateien, die wir uns im Folgenden näher ansehen.

Was sind Template-Dateien?

Bevor Sie die einzelnen Template-Dateien kennenlernen, möchte ich noch die Frage beantworten, was Template-Dateien sind. Das ist schnell erklärt: Es sind PHP-Dateien, die eine Mischung aus HTML, Template-Tags (siehe Abschnitt 5.3.7, »Funktionen für Templates und der Loop«) und PHP-Code enthalten. Sie bilden damit Vorlagendateien, um das Layout und Design verschiedener Teile einer Website zu beeinflussen. Wenn jemand eine Seite auf einer Website besucht, lädt WordPress eine Template-Datei auf der Grundlage der Anfrage. Die Template-Hierarchie (siehe Abbildung 5.29) beschreibt, welche Template-Datei betroffen ist. Der Server führt dann den PHP-Code in der Vorlage aus und gibt das daraus entstandene HTML an den Browser des Besuchers zurück.

Inhaltstypen

WordPress bietet Ihnen mehrere unterschiedliche Inhaltstypen. Die zwei offensichtlichen sind *Beiträge* und *Seiten*. Über spezielle Funktionen lassen sich weitere Inhaltstypen hinzufügen (siehe Kapitel 13, »Mit eigenen Inhaltstypen arbeiten«). Im Englischen spricht man von *Post Types*. Der Begriff *Post* ist der Oberbegriff für einen einzelnen Beitrag, eine Seite oder einen Artikel eines eigenen Inhaltstyps. Im weiteren Verlauf werde ich der Einfachheit halber ebenfalls den Begriff *Post* verwenden.

Der Unterschied zwischen Beiträgen und Seiten ist, dass Beiträge eine chronologische Reihenfolge haben. Sie dienen also zum Schreiben von Blogbeiträgen oder News, während Seiten eher statisch sind. Zum Beispiel werden sie fast immer für Landingpages¹⁹, Kontaktseiten oder das Impressum verwendet.

¹⁹ Eine Landingpage ist eine Zielseite, die ein spezielles Ziel verfolgt, den Verkauf eines Produkts zum Beispiel. Sie ist daher meist für eine bestimmte Zielgruppe optimiert.

Welche Inhaltstypen sind noch in WordPress integriert?

Weitere integrierte Inhaltstypen sind *Medien*, *Revisionen* und *Navigationsmenüs*. Revisionen sind im WordPress-Backend erst einmal nicht sichtbar. Erst wenn ein Post verändert und aktualisiert wird, entsteht eine Revision. So ist es möglich, alle Änderungen nachzuvollziehen und zu einem späteren Zeitpunkt zu einer bestimmten Version zurückzukehren.

Beiträge können verschlagwortet und kategorisiert werden. Jedes Schlagwort und jede Kategorie erzeugt eine eigene Unterseite. Diese werden zusammengefasst als *Archive* bezeichnet.

Da der Inhaltstyp Beiträge also deutlich mehr Arten von Seiten erzeugen kann als der Inhaltstyp Seiten, dreht sich in der Template-Hierarchie fast alles um Beiträge.

Die Template-Hierarchie

Die *index.php*-Datei ist bei der Template-Hierarchie so etwas wie eine *Auffangvorlage*. Sie wird immer dann aufgerufen, wenn für die Anfrage keine spezifische Template-Datei existiert. In der Regel wird ein Entwickler aber die meisten Dateien nutzen, um eine individuelle Gestaltung hinterlegen zu können.

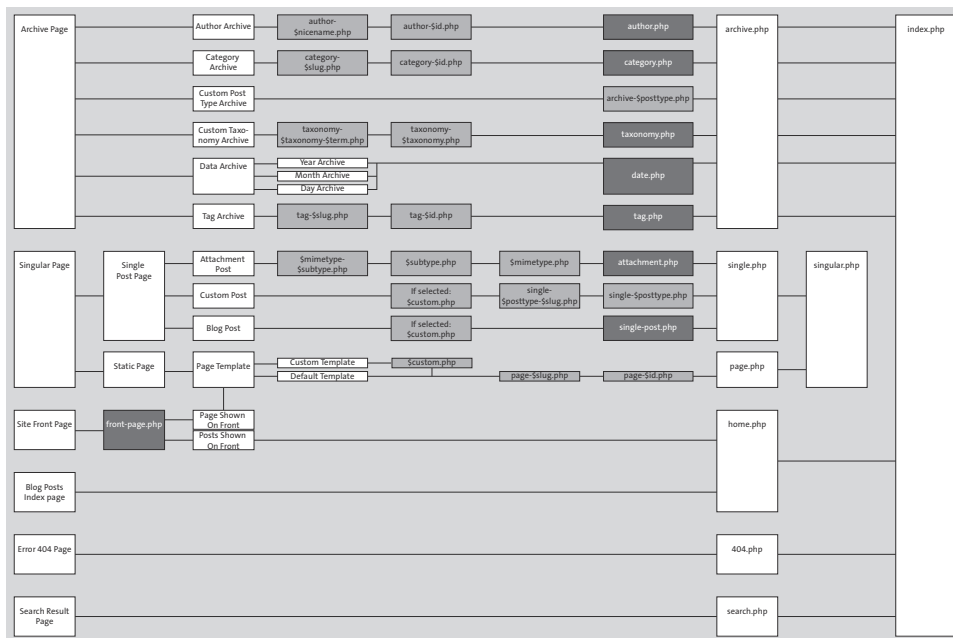


Abbildung 5.29 Die komplette Template-Hierarchie²⁰

²⁰ Quelle: Screenshot der Website <https://wphierarchy.com>. Übersetzung, Abgerufen am 03.01.2022.

Abbildung 5.29 zeigt einen Screenshot von der Website <https://wphierarchy.com>. Dort können Sie mit der Karte sehr leicht interagieren. Die komplette Template-Hierarchie in WordPress wird ebendort abgebildet. Ganz links ist die Einstiegsseite zu sehen. Je nach Anfrage landet der Besucher also auf einer der in Tabelle 5.9 dargestellten Seiten mit den entsprechenden Template-Dateien.

Seite	Template-Datei	Beschreibung
Archive Page	<i>archive.php</i>	Eine Archivseite ist eine Seite, die mehrere Beiträge anzeigt.
Singular Page	<i>singular.php</i>	Ein einzelner Beitrag, eine Seite oder ein Post eines anderen Inhaltstyps
Site Front Page	<i>front-page.php</i>	Die Startseite
Blog Posts Index Page	<i>home.php</i>	Seite mit der chronologischen Reihenfolge aller Beiträge
Error 404 Page	<i>404.php</i>	Eine Fehlerseite, die erscheint, falls nichts gefunden werden konnte.
Search Result Page	<i>search.php</i>	Listet alle Suchtreffer auf.

Tabelle 5.9 Trefferseite mit den zugehörigen Template-Dateien

Eine Anfrage endet jedoch nicht bei diesen Templates. WordPress checkt vorher, ob noch spezifischere Template-Dateien existieren. Landet ein Besucher auf einer Archivseite, wird geprüft, um welche es sich handelt. Handelt es sich um eine Kategorie-seite, so wird geprüft, ob die Datei *category.php* vorhanden ist. Falls ja, wird sie genutzt und eingebunden. Das geschieht jedoch erst, wenn nicht noch etwas Spezifischeres existiert. Laut Abbildung 5.29 wäre das die Datei *category-\$id.php*, wobei *\$id* für die Datenbank-ID steht, die WordPress für diese Kategorie vergeben hat. Somit erlaubt das Template-System, für jedes einzelne Objekt im System ein Template anzulegen. Ob das sinnvoll ist, muss jedoch jeder für sich selbst entscheiden. Da Sie nur selten jede einzelne Kategorie umgestalten werden, genügt in den meisten Fällen die Nutzung der *category.php* oder der *archive.php*.

Seiten-Templates

Es gibt noch ein weiteres Template-System in WordPress, nämlich Seiten-Templates. Der Name ist wieder etwas ungünstig gewählt, weil damit nicht unbedingt der Inhaltstyp *Seiten* gemeint ist. Vielmehr können alle Inhaltstypen, die eine entsprechende Freischaltung erhalten haben, Seiten-Templates nutzen. WordPress durchsucht dazu das Unterverzeichnis */page-templates/* im Theme und stellt sie als Auswahl zur Verfügung.

Seiten-Templates haben eine Besonderheit: Sie benötigen eine Meta-Angabe in Form eines DocBlocks. Als Beispiel erstelle ich die Datei `/page-templates/full-width.php` mit dem Inhalt aus Listing 5.21:

```
<?php
/**
 * Template Name: Full Width
 */
```

Listing 5.21 Anfänglicher DocBlock einer Seiten-Template-Datei

Anhand dieser Meta-Angabe wird das Seiten-Template erkannt und kann anschließend im Block-Editor **1** angezeigt und ausgewählt werden (siehe auch Abbildung 5.30).

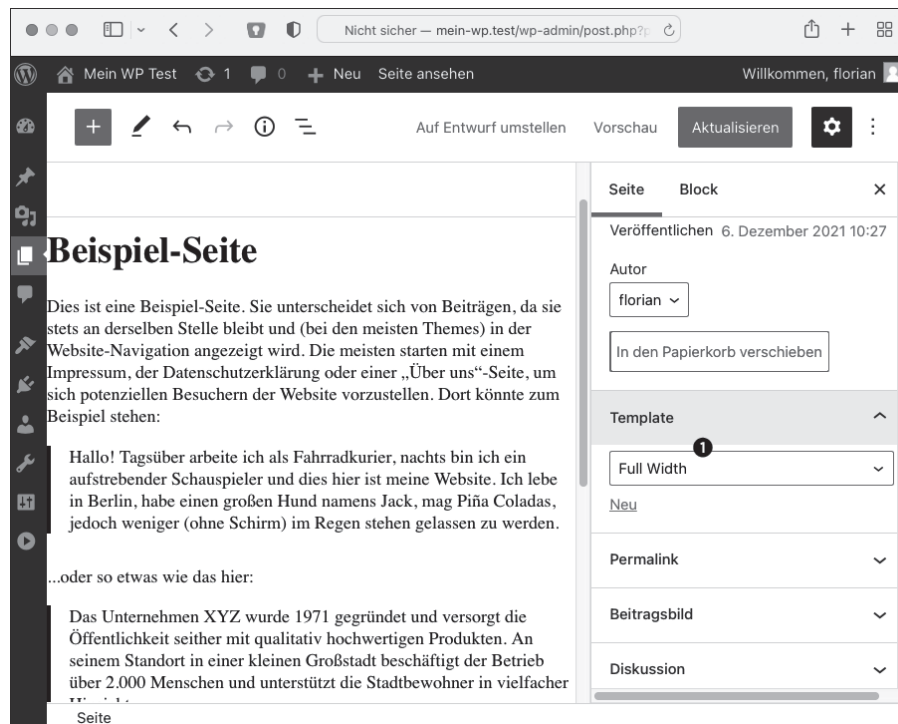


Abbildung 5.30 Auswahlmenü des Seiten-Templates

Tipp

Seiten-Templates müssen nicht zwingend im Verzeichnis `/page-templates/` abgelegt werden. Sie können sie auch im Stammverzeichnis des eigenen Themes speichern. Dann aber benötigt der Dateiname das Präfix `page-`. In meinem Fall müsste ich die Datei also `page-full-width.php` nennen.