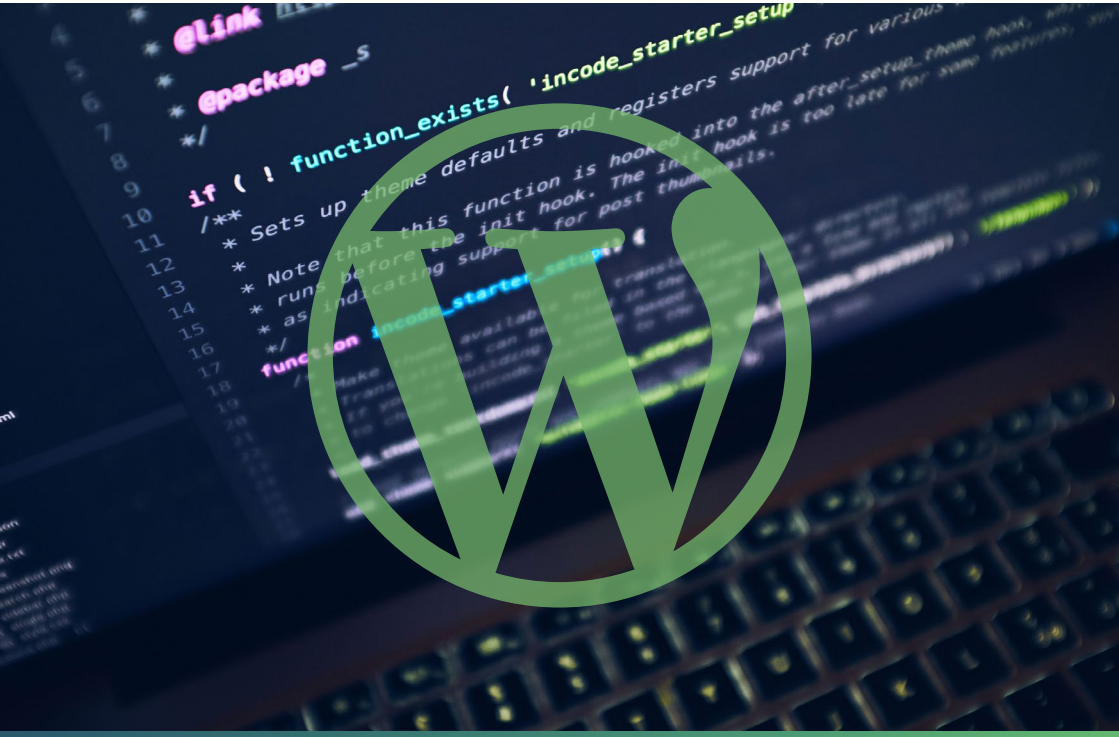
Host Europe



WordPress Gutenberg

Experten-Tipps und Techniken
rund um den neuen Block-Editor

E-Book

Inhaltsverzeichnis

Der Redakteur

Oliver Lindberg

Oliver Lindberg ist ein unabhängiger Redakteur, Content-Consultant, und Gründer von Pixel Pioneers, einer Konferenz für Frontend-Entwickler und UX/UI Designer. Ehemals Chefredakteur der wegweisenden Zeitschrift 'net magazine', beschäftigt Oliver sich inzwischen seit mehr als 15 Jahren mit Webdesign und -entwicklung und hilft internationalen Unternehmen bei der Umsetzung von erfolgreichen Content(-Marketing)-Strategien.

Vorwort

Als WordPress 5.0 im Dezember 2018 veröffentlicht wurde, war die Einführung des neuen Block-Editors – auch als Gutenberg bekannt – eine der gravierendsten Änderungen in der bis dahin fünfzehnjährigen Geschichte der Plattform.

Gutenberg bringt viele neue Möglichkeiten mit sich, Inhalte in WordPress-Seiten einzufügen und zu gestalten. Mittlerweile sind wir bei Version 5.6 angelangt, und der Block-Editor etabliert sich langsam als Standard. In diesem E-Book möchten wir Ihnen daher veranschaulichen, wie Sie Ihre eigenen Projekte an den Block-Editor anpassen und Kunden die Arbeit mit Gutenberg erleichtern können.

Das Autoren-Line-Up liest sich dabei wie ein Who's Who der deutschsprachigen WordPress-Community: Zunächst gibt Frontend-Entwicklerin Jessica Lyschik, die auch hinter der Gutenberg-Fibel steckt und aktiv an dem neuen WordPress-Standard-Theme Twenty Twenty-One mitgearbeitet hat, einen Überblick über den aktuellen Stand des Block-Editors sowie die Vor- und Nachteile, mit ihm zu arbeiten. Dann erläutert Webdesignerin Maja Benke, wie man den Block-Editor individualisieren kann, um Kunden zu ermöglichen, Seiten im Stil ihrer Corporate Identity zu erstellen. Die WordPress-Trainerin Britta Kretschmer geht auf einen speziellen Aspekt des Block-Editors ein: die Gestaltung von Inhalten anhand der neuen Inhaltsbreiten alignfull und alignwide. Der Plugin-Autor Bernhard Kau erklärt Ihnen, wie man Plugins dem Block-Editor anpasst, und beschreibt den Entwicklungsprozess eines Custom-Post-Types. Abschließend zeigt der ebenfalls sehr aktive WordPress-Entwickler Simon Kraft (u. a. zuständig für WP Letter und KrautPress) die Vorteile von WordPress-Support und wie man diesen effektiv anbieten kann, um nachhaltige Kundenbeziehungen aufzubauen sowie verlässliche Einnahmen zu generieren.

Viel Spaß beim Lesen!

Die Autorin

Jessica Lyschik

Jessica Lyschik ist Frontend-Entwicklerin bei der result gmbh in Köln. Als gelernte Fachinformatikerin für Anwendungsentwicklung hat sie bereits über 14 Jahre Erfahrung in der Webentwicklung gesammelt. Seit 2015 liegt der Fokus auf WordPress, das System kennt sie schon seit etwa 2006. Sie ist aktives Mitglied der WordPress-Community und teilt ihr Wissen regelmäßig auf Meetups und Konferenzen. Im November 2018 veröffentlichte sie die Gutenberg-Fibel, das deutsche „Missing Manual“ zum neuen Block-Editor von WordPress.

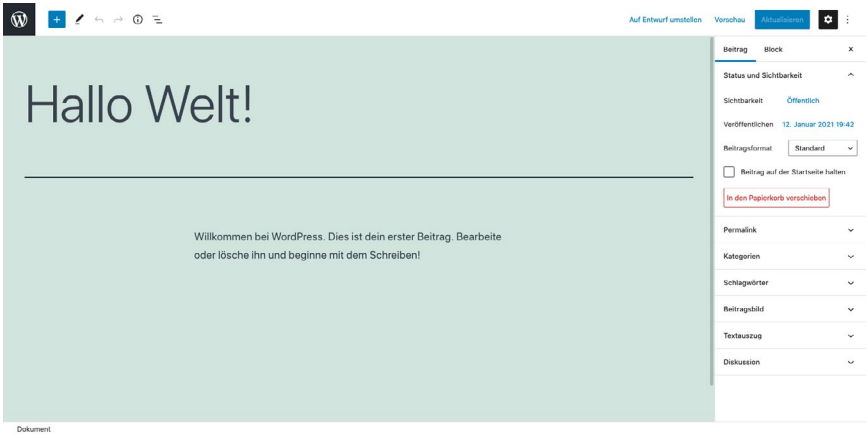
Der Block-Editor – die Zukunft von WordPress

Kein anderes Projekt innerhalb von WordPress hat dermaßen für kontroverse Diskussionen gesorgt wie der Block-Editor, a.k.a. Gutenberg. Der Mensch ist ein Gewohnheitstier und Veränderungen sind oftmals unangenehm, besonders im größeren Ausmaß. Die Veröffentlichung in Version 5.0 wurde von vielen hitzigen Diskussionen begleitet, entsprechend verhalten war die Begeisterung für den neuen Editor. Dieses Minimum Viable Product erntete viel Kritik wie „es kann ja nichts". Doch wie sieht es eigentlich heute aus?

Inzwischen sind zwei Jahre vergangen, und der Block-Editor hat sich kontinuierlich weiterentwickelt. Auch wenn verglichen mit anderen Page-Buildern der Block-Editor scheinbar noch „nicht so viel kann", kann man dies auch als Stärke sehen: der auf der Webseite ausgegebene Code ist schlanker und die dazugehörigen Assets (CSS- und JavaScript-Dateien) weitaus leichtgewichtiger. Immer öfter liest man von Vergleichen zwischen den Platzhirschen unter den Page-Buildern und wie der Block-Editor diese in Sachen Performance ziemlich alt aussehen lässt.

Zu Beginn des Jahres 2021 steht der Block-Editor an einem weiteren wichtigen Punkt in seiner Entwicklung: nachdem nun die Editor-Funktionen selbst ausgereift sind, soll mit dem Block-Editor zukünftig die gesamte Seite editierbar sein. Dieses Projekt nennt sich „Full Site Editing" und soll im Laufe diesen Jahres den Weg in die anstehenden WordPress-Versionen finden. Zukünftig sind also auch Header, Footer und Sidebars aus dem Editor heraus anpassbar, was bisher nur über die Theme-Dateien möglich war. Dies wird eine weitere, tiefgreifende Veränderung mit sich bringen: Themes werden nicht mehr ein Konglomerat aus Template-

Dateien und Funktionalitäten der Website sein, sondern sich viel stärker auf das Bereitstellen der Templates konzentrieren, wie es ursprünglich einmal gedacht war.



Screenshot des Beispielbeitrags aus WordPress mit Version 5.6.1.

Das ist auch noch nicht das Ende der Entwicklung, die der Block-Editor durchmachen wird. Die Phase 2 mit dem Full Site Editing ist zwar noch im Gange, aber bereits von Beginn an hat Matt Mullenweg, Mitbegründer von WordPress und CEO von Automattic, die grobe Roadmap vorgegeben: in Phase 3 sollen die Möglichkeiten zur Kollaboration verbessert werden. Vieles ist hier mit Plugins möglich. Am Ende der Phase 3 soll es möglich sein, dass mehrere Personen an einem Beitrag oder einer Seite gleichzeitig arbeiten können. Im letzten Schritt, der Phase 4, soll der Block-Editor von Haus aus mehrsprachig werden. Auch diese Funktionalität kann aktuell nur mit zusätzlichen Plugins umgesetzt werden.

Der Block-Editor ist also weiterhin ein sehr dynamisches Projekt, dass kontinuierlich Veränderungen erfährt und Weiterentwicklungen erhält. Es ist deshalb aber nicht schlechter geeignet als Page-Builder, um die eigene Website oder Kunden-Websites damit zu gestalten. Denn auch die Konkurrenz schläft nicht und entwickelt ihre Produkte weiter. Wer bisher mit einem Page-Builder gearbeitet hat und gut damit zurechtkommt, muss jetzt nicht auf Biegen und Brechen umsteigen. Wenn aber ein Relaunch der Seite ansteht, ist der Wechsel zum Block-Editor durchaus eine valide Option.

Die Vor- und Nachteile des Block-Editors im Überblick

Steht ein Relaunch einer vorhandenen Website oder ein neuer Kundenauftrag an, ist es hilfreich, die Vor- und Nachteile des Block-Editors zu kennen. Diese wollen wir uns im Folgenden genauer ansehen.

Performance durch schlanken HTML-Code und wenig Assets

Eine besondere Stärke des Editors ist die Performance. Durch den schlanken HTML-Code der Blöcke und einer einzigen CSS-Datei für die Core-Blöcke schlägt der Block-Editor jeglichen anderen Page-Builder.

Ein Beispiel dafür ist die Darstellung eines einfachen Absatzes, wenn wir den Block-Editor mit dem beliebten Page-Builder Elementor vergleichen. Während in Elementor neun verschachtelte Div-Elemente und ein Paragraph-Element benötigt werden, um einen einfachen Absatz anzuzeigen, wird im Block-Editor der Text einfach nur in ein Paragraph-Element gepackt. Diese Differenz findet man nicht nur beim Absatz, sondern auch bei allen anderen Standard-Elementen, die beide Editoren zu bieten haben. Hochgerechnet auf eine komplett gestaltete Seite, wird allein die Differenz des ausgegebenen HTML-Codes zwischen dem Block-Editor und Elementor immer größer.

```
<div class="entry-content">
  ::before
 <p>
    Willkommen bei WordPress. Dies ist dein erster Beitrag. Bearbeite oder lösche ihn und beginne mit dem Schreiben!
  </p>
  ::after
</div>
```

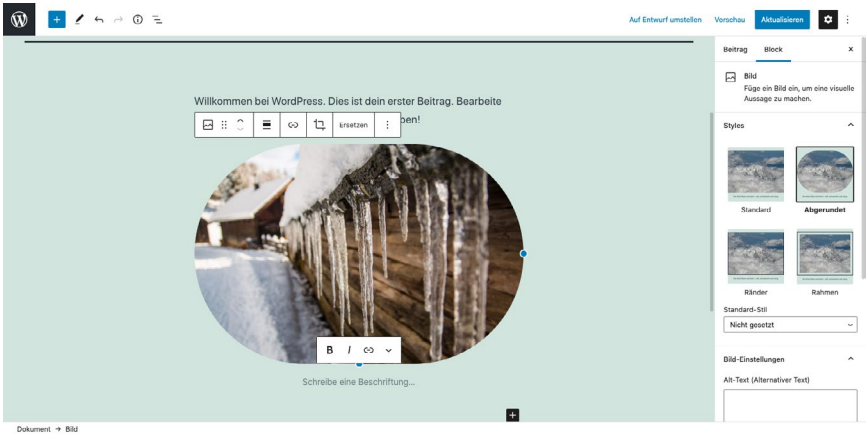
Ebenso schlank wie das HTML ist auch das mitgelieferte CSS. Elementor benötigt insgesamt sechs CSS-Dateien, die kombiniert um die 169 KB betragen. Der Block-Editor hingegen kommt mit einer CSS-Datei von 54 KB aus. Noch stärker fällt der Unterschied im Bereich JavaScript aus: Elementor benötigt acht Dateien von fast 350 KB Dateigröße. Der Block-Editor kommt im Core komplett ohne JavaScript aus. Auf meiner Testseite mit einer Überschrift, einem Absatz und einem Bild lädt Elementor knapp über 500 KB an CSS- und JavaScript-Dateien. Der Block-Editor lädt eine CSS-Datei, die gerade mal etwas mehr als 10 Prozent der Datenmenge ausmacht.

Dieser Vergleich soll Elementor nicht schlechter darstellen, als der Page-Builder eigentlich ist. Es zeigt aber sehr gut, was bei diesen „All-In-One" Page-Buildern eine große Schwäche ist: sie können viel, aber um das zu können, müssen sie auch viel Ballast mitbringen. Dem gegenüber steht der Block-Editor mit seiner schlanken Basis, die – wenn nötig – auch noch erweitert werden kann.

Gute Erweiterbarkeit für zusätzliche Funktionen

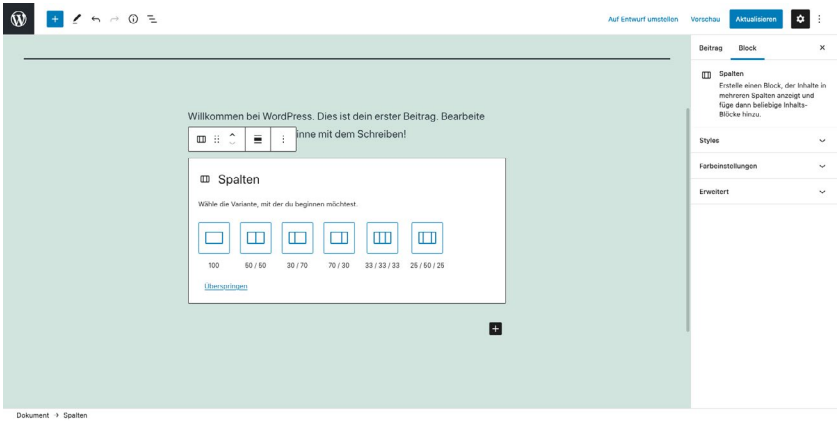
Seit der Einführung des Block-Editors sind die Möglichkeiten zur Erweiterung und Anpassung durch APIs immer weiter gewachsen. Neben der Erweiterung durch eigene, selbstgeschriebene Blöcke gibt es inzwischen verschiedene Möglichkeiten, die Standard-Blöcke anzupassen, ohne gleich einen ganz neuen Block mit ähnlichen Funktionen schreiben zu müssen.

Die einfachste Möglichkeit der Erweiterung sind die Block-Stile (engl. Block Styles). Diese findet man z. B. beim Bild-Block oder beim Button-Block: In der Sidebar werden ganz oben die verschiedenen Stile angezeigt. Bei den Buttons gibt es die Stile für „Füllen“ und „Kontur“ beim Bild-Block finden sich die Stile „Standard“ und „Abgerundet“. Die Stile ändern also nur das optische Erscheinungsbild des Blocks.



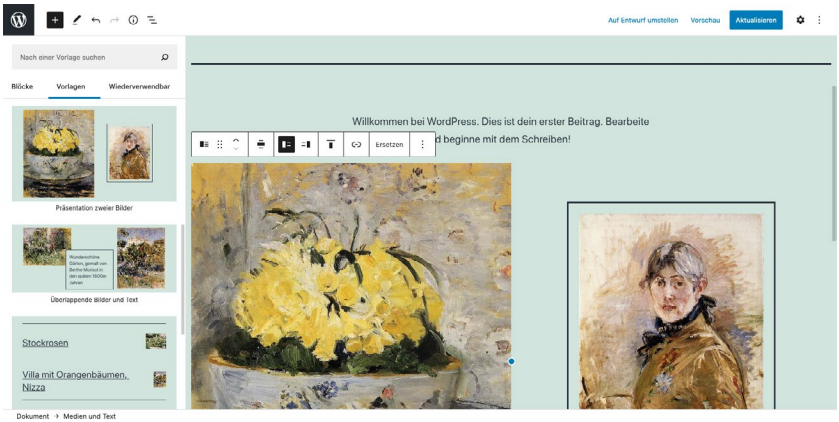
Die Block-Stile beim Bild-Block im Twenty Twenty-One Theme.

Als weitere Möglichkeit wurden die Block-Variationen (engl. Block Variations) in WordPress 5.4 eingeführt. Dabei lassen sich innerhalb eines Blocks verschiedene Möglichkeiten der Gestaltung per Klick auswählen. Der vermutlich bekannteste Anwendungsfall dafür ist der Spalten-Block. Fügt man diesen ein, werden verschiedene Optionen für die Spalten angezeigt: ein, zwei oder drei Spalten mit unterschiedlichen Aufteilungen. Wählt man eine der Optionen aus, wird innerhalb des Spalten-Blocks die entsprechende Anzahl der Spalten und mit der gewählten Breite hinzugefügt.



Beispiel der Block-Variation beim Spalten-Block.

Ein anderer Anwendungsfall wäre beispielsweise ein Formular-Block, bei dem verschiedene Arten von Formularen bereits fertig konfiguriert über die Variationen per Klick eingefügt werden können. Eine Block-Variation ist dann sinnvoll, wenn man einen bereits vorhandenen Block nur minimal abwandeln möchte und auf das Erstellen eines eigenen Blocks verzichten möchte.



Beispiele der Block-Vorlagen von Twenty Twenty-One.

Die dritte Möglichkeit zur Anpassung sind Block-Vorlagen (engl. Block Patterns), die seit WordPress 5.5 verfügbar sind. Im Prinzip ist die Block-Vorlage ein aus mehreren einzelnen Blöcken zusammengestelltes Template, dass beliebig oft eingesetzt werden kann. Die Inhalte können alle individuell angepasst werden und nach Belieben auch ausgetauscht oder gelöscht werden. Während eine Block-Variation immer einem bestimmten Block zugeordnet sein muss, sind die Block-Vorlagen völlig unabhängig von anderen Blöcken.

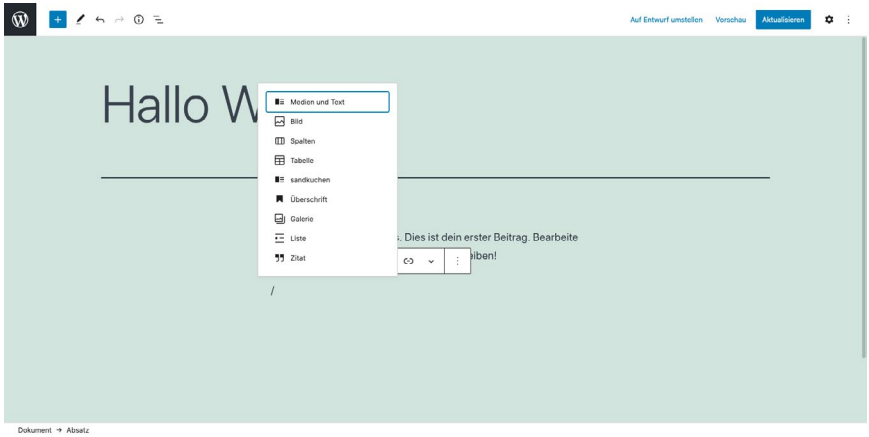
Bereits von Beginn an war die Möglichkeit gegeben, einen wiederverwendbaren Block anzulegen. Wie der Name schon sagt, lässt sich dieser Block auf mehreren Seiten wiederverwenden. An einer Stelle geändert, wirkt sich die Änderung auch auf alle anderen Instanzen aus. Bevor die Block-Vorlagen dazu kamen, konnte man die Funktion über die wiederverwendbaren Blöcke abbilden – man musste diese nach dem Einfügen nur „in normale Blöcke“ umwandeln, also quasi entkoppeln.

Wenn all diese Optionen nicht reichen, um die gewünschte Funktionalität abzubilden, dann kann natürlich noch ein eigener Block erstellt werden. Dabei kann man auf verschiedene Möglichkeiten zurückgreifen, wie das *create-block Package*, den WP-CLI-Befehl *wp scaffold block*, eine Boilerplate wie *create-guten-block*, oder auch Blöcke mittels dem Advanced Custom Fields Plugin erstellen. Dies ist natürlich um einiges aufwendiger und wartungsintensiver, als die vorhandenen APIs wie Block-Stile, Block-Variationen und Block-Vorlagen zu nutzen. Es ist daher sinnvoll, einen eigenen Block nur dann zu erstellen, wenn dieser nicht durch die Nutzung der gegebenen APIs umgesetzt werden kann.

Einfache Erstellung der Inhalte

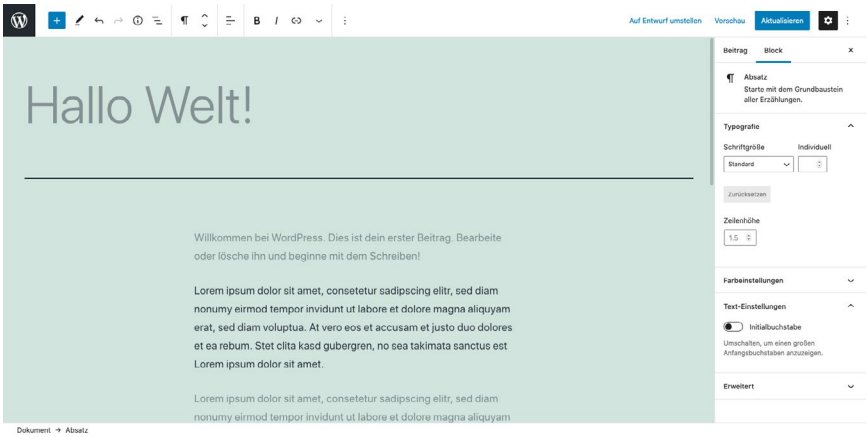
Die Inhaltserstellung im Block-Editor ist recht unkompliziert: soll ein Text hinzugefügt werden? Einfach drauflos tippen. Braucht man eine Überschrift, ein Bild, ein Trennzeichen? Man sucht sich den passenden Block

im Block-Inserter aus oder tippt einfach ein / und den Namen des entsprechenden Blocks, z. B. /überschrift. Durch die visuelle Darstellung der Inhalte, die fast zu 100 Prozent identisch zum Frontend dargestellt werden kann, lassen sich die Inhalte auch von Personen ohne große technische Kenntnisse umsetzen.



Editor-Ansicht ohne rechte Seitenleiste. In einem neuen Absatz-Block wurde ein Slash eingetippt und die vorgeschlagenen Blöcke erscheinen.

Die Oberfläche ist aufgeräumt und legt den Fokus auf den Inhalt. Die rechte Seitenleiste kann bei Bedarf auch ausgeblendet werden, um dem Inhalt noch mehr Raum zu geben. Über die Einstellungen lassen sich zwei weitere Ansichts-Optionen auswählen, die obere Werkzeugleiste und der Spotlight-Modus. Bei der oberen Werkzeugleiste wird die sonst direkt am Block angezeigte Werkzeugleiste in die obere Editorleiste verschoben. Der Spotlight-Modus graut alle nicht aktiven Blöcke aus und stellt nur den Block in den Vordergrund, in dem sich gerade der Cursor befindet.



Angepasster Editor mit aktiviertem Spotlight-Modus und oberer Werkzeugleiste.

Wenn man Blöcke neu anordnen möchte, können diese über die Pfeile nach unten und oben in der Werkzeugleiste jeweils um eine Position verschoben werden. Darüber hinaus kann man sie noch mit der Maus an die entsprechende Stelle ziehen. Dies braucht ein klein wenig Übung, gerade bei verschachtelten Blöcken wie dem Spalten-Block, wenn man Elemente hinzufügen oder aus dem genannten entfernen möchte. Dennoch ist die Arbeit mit dem Block-Editor weitaus bequemer, als sie es je mit dem Classic Editor war.

Der Block-Editor ist nicht barrierefrei

Der von Beginn an größte Kritikpunkt am Block-Editor ist die fehlende Barrierefreiheit bzw. Barrierearmut. Im Frontend kann die Website mit dem passenden Theme barrierefrei gestaltet werden, gute Beispiele sind hierfür die Standard-Themes Twenty Twenty und Twenty Twenty-One. Dennoch gibt es weiterhin Hürden für Menschen, die z. B. auf einen Screenreader angewiesen sind oder ausschließlich mit der Tastatur arbeiten.

Anfang 2019 wurde ein professioneller Accessibility Audit von der Community mittels Crowdfunding gestartet, auch Automattic beteiligte sich an der Finanzierung. Das Ergebnis war ein 329-seitiger Audit, bei dem die Version 5.0 getestet wurde. Seitdem hat sich bereits viel getan, auch wenn es noch immer offene Punkte gibt. Durch den Audit wurde auch der Fokus für Barrierefreiheit gestärkt.

Es gibt eine Lernkurve, und die muss ab und an aufgefrischt werden

Wer schon mit WordPress gearbeitet hat und einen eigenen Workflow etabliert hat, der tut sich vermutlich schwer damit, den Einstieg in den Block-Editor zu finden. Die Arbeitsweise ist definitiv anders und bringt eine gewisse Lernkurve mit sich, die erstmal bewältigt werden muss. Wie kann ich neue Blöcke hinzufügen? Wie kann ich die Einstellungen ändern? Wo finde ich meine zusätzlichen Felder oder Metaboxen meiner Plugins, z. B. für SEO? Welche Möglichkeiten habe ich mit den Standard-Blöcken? Mit welchen Blöcken und Einstellungen kann ich mein Wunschlayout nachbauen?

Wenn man sich dann mit dem Block-Editor beschäftigt und die Antworten auf die beispielhaften Fragen oben gefunden hat, kann es passieren, dass man erneut nach Antworten suchen muss. Durch die stetige Weiterentwicklung und Verbesserung passiert es immer wieder, dass sich das Aussehen ändert oder Funktionen an eine andere Stelle wandern oder sich gänzlich verändern. Der Block-Editor von heute ist in vielen Punkten anders als der Block-Editor in WordPress 5.0. Auch wenn dies bei anderen Page-Buildern ebenfalls vorkommt, beim Block-Editor passieren die Änderungen noch etwas häufiger.

Auch APIs und der Code verändert sich

Auch für Entwickler ist die Arbeit mit dem Block-Editor nicht immer einfach. Gerade wer sich erst neu damit beschäftigen möchte, der schaut sich die Dokumentation an und sucht nach Tutorials und Anleitungen. Beim Ausprobieren muss man dann allerdings feststellen, dass die Dokumentation nicht unbedingt vollständig ist und die Tutorials irgendwie nicht mehr so richtig funktionieren, gerade wenn sie aus der Anfangszeit des Editors sind.

Dabei kann es durchaus passieren, dass selbstentwickelte Blöcke nach einer gewissen Zeit nicht mehr richtig oder gar nicht mehr funktionieren, weil sich Änderungen am Code oder an den APIs ergeben haben. Ich habe bereits von einigen Entwicklern gehört, die die Arbeit an ihren eigenen Erweiterungen eingestellt haben, weil sich einfach zu viel zu schnell geändert hat. Gerade wenn solche Eigenentwicklungen nur Hobby oder ein kleines Seitenprojekt sind, ist die Frustrationsgrenze schnell erreicht.

Warum am Ball bleiben wichtig ist
Gerade durch die schnell voranschreitende Entwicklung ist es nötig, am Ball zu bleiben. Dabei muss man nicht unbedingt jeden Commit lesen und jede Issue im Blick behalten, die im Gutenberg-Repository hinzugefügt werden.

Was final in einer neuen WordPress-Version landet, wird vorher im Gutenberg-Plugin zur Verfügung gestellt. Zu jeder neuen Plugin-Version gibt es ein ausführliches Update mit den Änderungen im Core-Blog, dass die Änderungen in verschiedene Kategorien aufteilt: Features, Enhancements, New APIs, Security, Breaking Changes, Bug Fixes, Performance, Experiments, Documentation, Code Quality, Tools, Various. Nicht immer sind alle Kategorien enthalten. Ebenso gibt es Blogposts, die zum Monatsanfang einen Überblick darüber geben, was als nächstes in der Entwicklung ansteht. Es lohnt sich, dort regelmäßig einen Blick drauf zu werfen und sich die interessantesten Dinge genauer anzusehen.

Und natürlich hilft das Arbeiten mit dem Block-Editor auch dabei, Veränderungen wahrzunehmen. Ein für Änderungen offenes Mindset hilft dabei, nicht sofort in Frust zu verfallen, wenn sich mal wieder ein Button an eine andere Stelle verschoben hat, oder die bisher genutzte Funktion irgendwie nicht mehr zu finden ist. Denn Software ist kontinuierlicher Veränderung unterworfen – manchmal in kleinen Schritten und manchmal in großen Sprüngen, wie bei dem Block-Editor.

Photo by jose aljovin on Unsplash

Die Autorin

Maja Benke

Maja Benke arbeitet als Webdesignerin und hat eine Leidenschaft für UX und Barrierefreiheit. Sie hält regelmäßig Vorträge und engagiert sich in der WordPress-Community, z. B. durch die Co-Organisation von WordCamps und des WordPress Meetups in Berlin und im Bereich Barrierefreiheit. Zudem erstellt sie Tutorials, über WordPress auf ihrem Blog „WP1x1.de" und über digitale Barrierefreiheit auf maja-benke.de.

Den Block-Editor für Kundenprojekte individualisieren

Dieser Artikel richtet sich vor allem an Webdesigner*innen mit Grundkenntnissen in HTML, CSS und PHP. Ziel ist es, über ein Theme und selbstentwickeltes Child-Theme eine Webseite zu erstellen und das Corporate Design in den Block-Editor (Gutenberg) zu integrieren. Auf diese Weise können die Kund*innen neuen Content im Design ihrer Corporate Identity erstellen. Die Code-Beispiele gelten für ein Child-Theme namens „Starter-Child“. Das Child-Theme bezieht sich auf das Twenty Twenty Theme als Parent-Theme. Alle Code-Beispiele sind als einzelne Gists verfügbar. Am Ende des Artikels befindet sich eine GitHub-Repo mit dem kompletten Code des Child-Themes.

Warum der Block-Editor

Der Vorteil des Block-Editors (auch Gutenberg genannt) ist, dass dieser im WordPress-Core integriert ist. Das heißt, man benötigt kein weiteres Plugin oder Theme, um den Block-Editor nutzen zu können. Weil er zu WordPress gehört, wird er auch von der Open Source Community entwickelt. Daher ist man nicht von einer Agentur abhängig, wie das zum Beispiel bei einigen Page-Buildern der Fall wäre.

Ein weiterer wichtiger Punkt ist die Kompatibilität. Viele Themes und Plugins, die neu auf den Markt kommen, bzw. ein Update enthalten, sind mit dem Block-Editor kompatibel, da dies der neue Status Quo von WordPress ist. Der alte Classic-Editor wird bald nicht mehr unterstützt. Daher ist es eine gute Zeit, nun auf den Block-Editor umzusteigen, wenn man dies bisher nicht getan hat.

Welche Themes eignen sich

Generell sollte bei der Themenauswahl darauf geachtet werden, dass das Theme aktuell und sauber geschrieben ist. Vor allem für Nicht-Entwickler*innen ist es wichtig, dass der Code gut strukturiert und verständlich ist. Nun ist es natürlich nicht immer einfach zu beurteilen, wie gut die Code-Basis ist. Als einen guten Anhaltspunkt achte ich immer darauf, dass das Theme übersetzbar und möglichst barrierearm ist. Gut wäre es zudem, wenn das Theme bereits Block-Stile unterstützt. Auf diese Weise muss man nicht die gesamte Block-Editor-Unterstützung ins Child-Theme schreiben.

Folgende Themes eignen sich dafür (kein Anspruch auf Vollständigkeit):

- Theme: Twenty Twenty
- Theme: Twenty Twenty One
- Theme: Aino
- Theme: Inclusive
- Theme: Go
- … und weitere Themes

Diese Themes eignen sich super als Grundlage für ein Child-Theme. Bei der Auswahl kann man schauen, welches Theme bereits dem gedachten Design am nächsten kommt, damit man nicht allzu viel anpassen muss. Die Code-Beispiele werden sich auf das Theme „Twenty Twenty“ beziehen. Ich habe es ausgewählt, weil es sich super als Starter-Theme eignet.

Twenty Twenty hat bereits einige Stile für die Blöcke, aber auch nicht zu viele, so dass der Code immer noch sehr schlank ist. Es bringt drei Templates für Seiten und Beiträge mit. Das Layout und die Optionen eignen sich sehr gut für Kunden-Webseiten.

Das neuere Theme „Twenty Twenty One“ ist ebenfalls sehr gut für den Block-Editor geeignet. Es ist jedoch ein klein wenig komplexer. Daher ist

das Arbeiten am Code damit etwas vielseitiger. Dieser Beitrag soll sich in erster Linie an Personen mit Basiswissen im Coding wenden. Twenty Twenty wird weiterhin aktualisiert und kann natürlich auch weiterhin aktiv genutzt werden.

Child-Theme erstellen

Nachdem man sich für ein Theme entschieden hat, wird als Nächstes ein Child-Theme erstellt. Bei der Erstellung des Child-Themes sollte man neben den Dateien style.css und functions.php auch die style-editor.css anlegen.

Im Falle von Twenty Twenty sieht der erste Code für das Erstellen des Child-Themes und das Einbinden der style.css und der style-editor.css wie folgt aus (Details im Gist auf GitHub):

style.css

Mit der style.css wird das Child-Theme erstellt und definiert, welches das Parent-Theme ist. Zudem bestimmt die Datei die Styles des Frontends.

```
/*
Theme Name:   Starter Child
Theme URI:    Domain of the Theme
Description:  Child of Twenty Twenty
Author:       Your Name
Author URI:   Your Domain
Template:     twentytwenty
Version:      1.0
License:      GNU General Public License v2 or later
License URI:  http://www.gnu.org/licenses/gpl-2.0.html
*/

/* == Add your own styles below this line ==
--------------------------------------------*/
```

Zum Code →

style.css

style-editor.css

Die Datei style-editor.css definiert das Aussehen der Inhalte im Block-Editor. Diese kann je nach Theme anders heißen und in einem Unterordner sortiert sein. In diesem Fall heißt sie „editor-style-block.css“ und ist in dem Ordner assets/css sortiert. Hier sollte man sich immer an der Struktur des Parent-Themes orientieren.

Die style-editor.css kann optional gestaltet werden, da sie nicht das Aussehen der Webseite selbst beeinflusst. Bei Kundenprojekten kann es sich anbieten, die Designs für die Kunden auch im Backend zu verankern, um den Eindruck von „What you see is what you get“ zu erhöhen.

```
1  /* ---------------------------------------------- */
2
3  /*      Starter Child Editor Styles – Block Editor
4  /* ---------------------------------------------- */
```

Zum Code →

style-editor.css

functions.php

In der functions.php werden die style.css und die style-editor.css aufgerufen und ausgegeben (enqueued). Diese Datei wird später mit weiteren Funktionen ergänzt.

```php
<?php
/**
 * This file is part of a child theme called Starter Child.
 * Functions in this file will be loaded before the parent theme's functions.
 * For more information, please read
 * https://developer.wordpress.org/themes/advanced-topics/child-themes/
 */

/**
 * Load the frontend parent and child theme styles
 */
function starter_child_enqueue_styles() {
	$parent_style = 'twentytwenty-style';

	wp_enqueue_style(
		$parent_style,
		get_template_directory_uri() . '/style.css',
		array(),
		wp_get_theme()->parent()->get( 'Version' )
	);

	wp_enqueue_style(
		'starter-child-style',
		get_stylesheet_directory_uri() . '/style.css',
		array( $parent_style ),
		wp_get_theme()->get( 'Version' )
	);
}
add_action( 'wp_enqueue_scripts', 'starter_child_enqueue_styles' );

/**
 * Load the block editor parent and child theme styles
 */
function starter_child_block_editor_styles() {
	wp_enqueue_style(
		'twentytwenty-block-editor-styles',
		get_theme_file_uri( '/assets/css/editor-style-block.css' ),
		array(),
		wp_get_theme()->get( 'Version' )
	);

	wp_enqueue_style(
		'starter-child-block-editor-styles',
		get_theme_file_uri( '/assets/css/editor-style-block.css' ),
		array(),
		wp_get_theme()->get( 'Version' )
	);
}
add_action( 'enqueue_block_editor_assets', 'starter_child_block_editor_styles', 1 );
```

Zum Code →

functions.php

screenshot.png

Das Bild screenshot.png wird im Backend unter Design -> Themes als Vorschau des Themes ausgegeben.

Hinweis: Es gibt weitere Style-Dateien, die importiert werden können. Dies betrifft vor allem Sprachen, die von rechts nach links gelesen werden, und andere Non-Latin-Sprachen, sowie Styles für den Classic-Editor. Diese habe ich weggelassen, um es überschaubarer zu machen.

Das Starter-Theme auf GitHub:
github.com/00travelgirl00/starter-child-basic.

Das komplette Starter-Theme mit dem Code aus den folgenden Abschnitten ist am Ende dieses Kapitels zu finden.

Child-Theme anpassen

Nun ist das Child-Theme erstellt und kann weiter angepasst werden. Als erstes füge ich immer eine Gliederung in die style.css ein, um den Code besser zu strukturieren.

Zum Code →

Block-Editor-Optionen personalisieren

Der nächste Schritt ist, den Block-Editor zu personalisieren. Dazu gehören Farben und Farbverläufe, Schriftgrößen, Block-Styles und Block-Patterns.

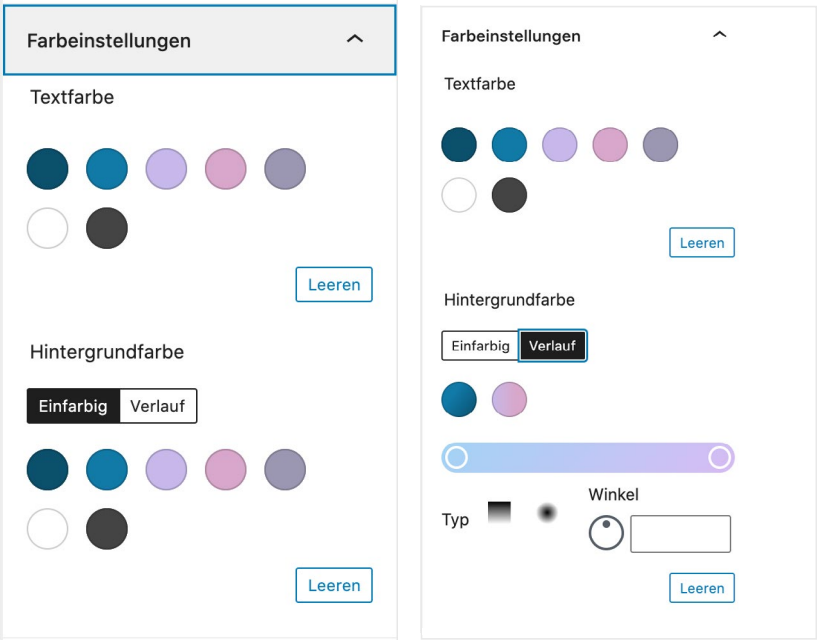
Farben und Farbverläufe

Vor allem die Farben spielen eine große Rolle im Look and Feel einer Webseite. Das Tolle ist, dass Farben und auch Farbverläufe im Block-Editor hinterlegt werden können.

Die Farben werden dafür im Child-Theme über die functions.php registriert und stehen dann für jeden Block zur Auswahl, der eine Farbwahl-Option hat. In einem nächsten Schritt muss man in der style.css die registrierten Farbklassen stylen.

Folgende Farboptionen kann man registrieren:

- Farben
- Farbverläufe
- Möglichkeit eines Custom Color Pickers (aktivieren oder deaktivieren)



Farben und Farbverlauf

Farben

Farben in der functions.php registrieren:

```php
//Add Gutenberg Support
/**
 * Registers support for Gutenberg Color Palette
 */
function starter_child_gutenberg_support() {

	// Add theme support for custom color palette.
	add_theme_support( 'editor-color-palette', array(
		array(
			'name'  => esc_html__( 'Blue Sapphire', 'starter-child' ),
			'slug'  => 'blue-sapphire',
			'color' => '#0b4f6c',
		),
		array(
			'name'  => esc_html__( 'Celadon Blue', 'starter-child' ),
			'slug'  => 'celadon-blue',
			'color' => '#117aa7',
		),
		array(
			'name'  => esc_html__( 'Blue Purple', 'starter-child' ),
			'slug'  => 'blue-purple',
			'color' => '#c7b8ea',
		),
		array(
			'name'  => esc_html__( 'Pink Lavender', 'starter-child' ),
			'slug'  => 'pink-lavender',
			'color' => '#d8a7ca',
		),
		array(
			'name'  => esc_html__( 'Cool Grey', 'starter-child' ),
			'slug'  => 'cool-grey',
			'color' => '#9b97b2',
		),
		array(
			'name'  => esc_html__( 'White', 'starter-child' ),
			'slug'  => 'white',
			'color' => '#ffffff',
		),
		array(
			'name'  => esc_html__( 'Dark Grey', 'starter-child' ),
			'slug'  => 'dark-grey',
			'color' => '#444444',
		),
	) );
}
add_action( 'after_setup_theme', 'starter_child_gutenberg_support', 11 );
```

Zum Code →

Farbklassen in der style.css definieren:

Es gibt insgesamt drei CSS-Klassen pro Farbe, die gestylt werden sollten. Diese sind (das xxx steht für den Namen der Farbe, die registriert wurde):

- **.has-xxx-color**
- **.has-xxx-background-color** (hier sollte neben der Hintergrundfarbe auch die Textfarbe definiert werden, um z. B. bei Buttons eine gute Lesbarkeit zu gewährleisten)
- **.has-xxx-background-color:hover** (diese sollte natürlich anders sein, als die background-color ohne Hover State)

Zudem sollte auch der Default-Style für Buttons und Co definiert werden, wenn keine Farbe gewählt wurde. Dafür werden folgende Selektoren verwendet:

```
.wp-block-button__link:not(.has-background)
.wp-block-button__link:not(.has-background):hover
```

Der Code für die Farben sieht dann wie folgt aus:

Zum Code →

Damit die Styles auch im Block-Editor angezeigt werden, empfiehlt es sich, die Styles aus der style.css auch in die editor-style-block.css zu laden.

Zwar werden die Farben aus dem Inline Style geladen und könnten auch weggelassen werden, aber für Farbverläufe bietet es sich an, diese zu übertragen.

Zum Code →

Wenn man sich noch etwas unsicher ist, welche Selektoren und welche Werte definiert werden sollen, kann man sich auch anschauen, wie es im Parent-Theme gelöst wurde.

Nachdem man alles ins Child-Theme geschrieben hat, sollten nun die Farben getestet werden.

Dazu kann man mehrere Buttons mit den verschiedenen Farben definieren und prüfen, ob diese korrekt gestylt, die Hover-Farben gut erkennbar und die Textfarben gut lesbar sind.

Es gibt bei dem Cover-Block zudem manchmal die Eigenheit, dass bei einem farbigen Hintergrund (anstatt eines Bildes) der Cover-Block schwarz erscheint, weil die Klasse .has-background-dim geladen wird. Diese müsste man unter Umständen ebenfalls anpassen.

Farbverläufe registrieren

Die Farbverläufe werden wie die Farben registriert und gestylt. Wichtig ist, bei dem Code darauf zu achten, dass er innerhalb der folgenden Funktion geschrieben ist:

```
function starter_child_gutenberg_support() { }
```

Um den Code für den Farbverlauf zu erstellen, gibt es verschiedene Online-Tools. Dort kann man über eine Einstellungsmaske die Farben, die Richtung des Farbverlaufs und die prozentualen Anteile der Farben einstellen.

Der Code für den Farbverlauf wird dann erstellt und man kann ihn kopieren, zum Beispiel mit dem Farbverlauf-Tool auf der Seite cssgradient.io.

Farbverläufe in der functions.php registrieren:

```
        // Add theme support for custom color gradients
        add_theme_support( 'editor-gradient-presets', array(
                        array(
                                'name'     => __( 'Blue Sapphire to Celadon Blue', 'starter-child' ),
                                'gradient' => 'linear-gradient(307deg, rgba(11,79,108,1) 0%, rgba(17,122,167,1) 66%)',
                                'slug'     => 'blue-sapphire-to-celadon-blue',
                        ),
                        array(
                                'name'     => __( 'Blue Purple to Pink Lavender', 'starter-child' ),
                                'gradient' => 'linear-gradient(90deg, rgba(199,184,234,1) 0%, rgba(216,167,202,1) 69%)',
                                'slug'     => 'blue-purple-to-pink-lavender',
                        ),
                )
        );
```

Zum Code →

Nun müssen die Farbverläufe noch in der style.css definiert werden. Hier reicht es aus, .has-background und den Hover State zu definieren.

Farbverläufe in der style.css definieren:

```
.has-blue-sapphire-to-celadon-blue-gradient-background {
    background: linear-gradient(307deg, rgba(11,79,108,1) 0%, rgba(17,122,167,1) 66%);
    color: #fff;
}
.has-blue-sapphire-to-celadon-blue-gradient-background:hover {
    background: linear-gradient(90deg, rgba(199,184,234,1) 0%, rgba(216,167,202,1) 69%);
}

.has-blue-purple-to-pink-lavender-gradient-background {
    background: linear-gradient(90deg, rgba(199,184,234,1) 0%, rgba(216,167,202,1) 69%);
    color: #000;
}
.has-blue-purple-to-pink-lavender-gradient-background:hover {
    background: linear-gradient(307deg, rgba(11,79,108,1) 0%, rgba(17,122,167,1) 66%);
}
```

Zum Code →

Farbverläufe in der editor-style-block.css definieren:

```
/*style color gradient*/
.has-blue-sapphire-to-celadon-blue-gradient-background {
    background: linear-gradient(307deg, rgba(11,79,108,1) 0%, rgba(17,122,167,1) 66%);
    color: #fff;
}
.has-blue-sapphire-to-celadon-blue-gradient-background:hover {
    background: linear-gradient(90deg, rgba(199,184,234,1) 0%, rgba(216,167,202,1) 69%);
}

.has-blue-purple-to-pink-lavender-gradient-background {
    background: linear-gradient(90deg, rgba(199,184,234,1) 0%, rgba(216,167,202,1) 69%);
    color: #000;
}
.has-blue-purple-to-pink-lavender-gradient-background:hover {
    background: linear-gradient(307deg, rgba(11,79,108,1) 0%, rgba(17,122,167,1) 66%);
}
```

Zum Code →

Custom Color Picker

Der Custom Color Picker ist eine Option, mit der eine beliebige Farbe ausgewählt werden kann, unabhängig von den registrierten Farben. Je nach Projekt und Block kann es Sinn machen, den Custom Color Picker zu erlauben oder zu unterbinden.

Für den Button-Block würde es sich zum Beispiel anbieten, den Color Picker zu unterbinden, weil das Stylen der Textfarbe sowie die Veränderung des Hover States recht schwierig sein kann.

Auch wenn gewährleistet sein soll, dass nur die definierten Farben des Styleguides von den Kunden ausgewählt werden können, kann es nützlich sein, den Color Picker zu deaktivieren.

Für den Fall, dass man den Custom Color Picker ausschaltet, sollte man außerdem sicherstellen, dass auch neutrale Farben registriert wurden. Dies ist wichtig, damit bei Buttons und Hintergründen ein guter Farbkontrast möglich ist. Dabei bietet es sich an, vor allem Weiß und Schwarz zu registrieren. Als Alternative oder zusätzlich auch gerne ein gebrochenes Weiß und ein Dunkelgrau.

Der Code zum Deaktivieren des Custom Color Pickers sollte ebenfalls innerhalb der Funktion stehen:

Befehl: function starter_child_gutenberg_support() { }

```
1          //Disable the custom color picker.
2          add_theme_support( 'disable-custom-colors' );
```

Zum Code →

Andere Stile wie Schriftgrößen werden ebenfalls auf diese Weise ins Theme geschrieben. Zuerst registriert man diese in der functions.php und danach stylt man sie in der style.css und in der editor-style-block.css.

Es gibt noch weitere Optionen, die man über add-theme-support in der functions.php hinzufügen kann. Dazu gehören z. B. Schriftgrößen, Full-Width-Unterstützung und einige mehr.

Anleitungen und Code-Beispiele gibt es im Entwicklungs-Handbuch von WordPress (auf Englisch).

Block-Stile

Neben den Farben kann auch über die Block-Stile eine Personalisierung des Designs erfolgen. Hier kann es Sinn machen, sich bereits im Vorfeld bei der Erstellung des Styleguides und der Mockups zu überlegen, welche Block-Stile die Webseite benötigt.

Folgende Blöcke bieten sich für Block-Stile an (dies ist nur eine Auswahl mit möglichen Beispielen. Die Stile sollten immer zum Design der Seite passen):

- Buttons
  - Voller Button
  - Outline Button
  - Button mit Farbverlauf
- Trennlinie
  - Breite Linie
  - Schmale Linie
  - Symbole
- Zitat
  - verschiedene Schriftarten, Schriftstile, Schriftgrößen, ...
- Überschrift
  - verschiedene Schriftarten, Schriftstile, Schriftgrößen, ...

Wie man Block-Stile anlegt, kann man in den folgenden Artikeln nachlesen:

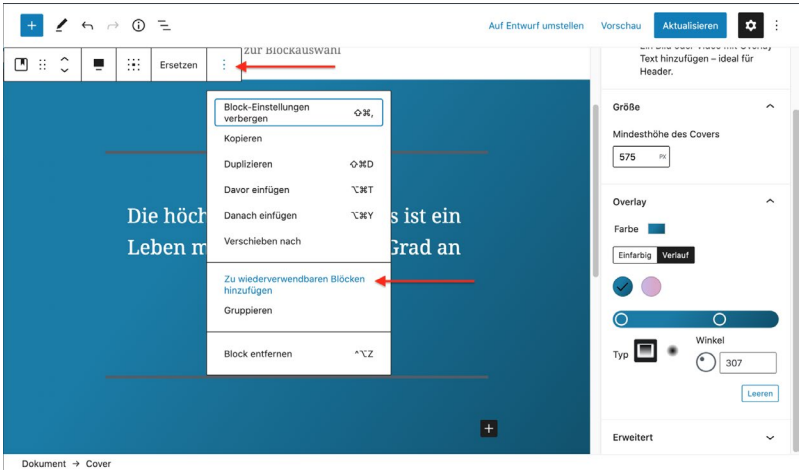
- Das Block Filters-Kapitel im Developer-Handbuch von WordPress (auf Englisch)
- So legst du eigene Gutenberg Block Styles an (von Webtimiser)
- Alternativen Stil für Gutenberg-Block erstellen (von KrautPress)
- Block-Styles für Gutenberg-Blöcke ergänzen (von den Netzialisten)

In den Beispielen wird zumeist ein Plugin verwendet. Man kann dies aber auch direkt über das Child-Theme machen.

Wiederverwendbare Blöcke

Wiederverwendbare Blöcke sind Inhalte, die über die verschiedenen Unterseiten oder Beiträge hinweg immer gleich sind. Ganz ähnlich wie bei den Widgets. Dies hat den Vorteil, dass Informationen, die sich ändern, nur an einer Stelle geändert werden müssen und diese Änderung auf der gesamten Webseite aktualisiert wird.

Um einen wiederverwendbaren Block anzulegen, erstellt man den gewünschten Block im Block-Editor und fügt dort die Inhalte ein. Dann markiert man den Block und über die drei Punkte in der Einstellungsleiste findet man die Option „zu wiederverwendbaren Blöcken hinzufügen“. Anschließend gibt man dem Block einen Namen und speichert ihn ab. Jetzt kann man diesen wiederverwendbaren Block auf jeder anderen Seite oder Beitrag einfügen und verwenden.



Einen wiederverwendbaren Block benennen.

Mit dem Plugin „Reusable Blocks Extended“ bekommt man einen Menüpunkt „Reusable Blocks“. Dort kann man alle wiederverwendbaren Blöcke sehen und bearbeiten.

Block-Pattern

Seit WordPress 5.5 gibt es die Block-Patterns direkt im Core. Diese sind Design-Vorlagen, die aus mehreren Blöcken bestehen. Anders als wiederverwendbare Blöcke synchronisiert sich der Inhalt nicht. Einige Themes kommen bereits mit Block-Patterns.

Es können zudem eigene Block-Patterns im Theme hinterlegt werden. Dies bietet sich vor allem bei häufiger vorkommenden Designs an. Zum Beispiel ein Team-Block, eine Customer-Review oder ähnliches. So gibt man den Kunden bereits kleine Layout-Bausteine an die Hand.

Die Block-Patterns können auch in Kategorien hinterlegt werden, um so eine bessere Übersicht zu gewährleisten. Das Hinterlegen der Block-Pattern ist recht simpel. Zuerst wird ganz normal im Block-Editor das Layout erstellt, das später ein Block-Pattern werden soll. Dabei sollte man direkt alle Designs und Einstellungen vornehmen. Dann werden die Blöcke markiert und kopiert. In der functions.php kann man dann den folgenden Code schreiben (siehe Gist).

Der Code der Blöcke kommt dann an die Stelle, wo „copy code from block editor here“ steht. Die Anführungszeichen müssen dabei erhalten bleiben.

Beispiel Code:

```
/* Register Block Patterns */
register_block_pattern(
        'unique-pattern-name',
        array(
                'title' => __( 'Name of Pattern', 'starter-child' ),
                'description' => _x( 'Here should be a description of the pattern', 'Block pattern description', 'starter-child
                'categories'  => array('buttons'),
                'content'     => "copy code from block editor here",
        )
);
```

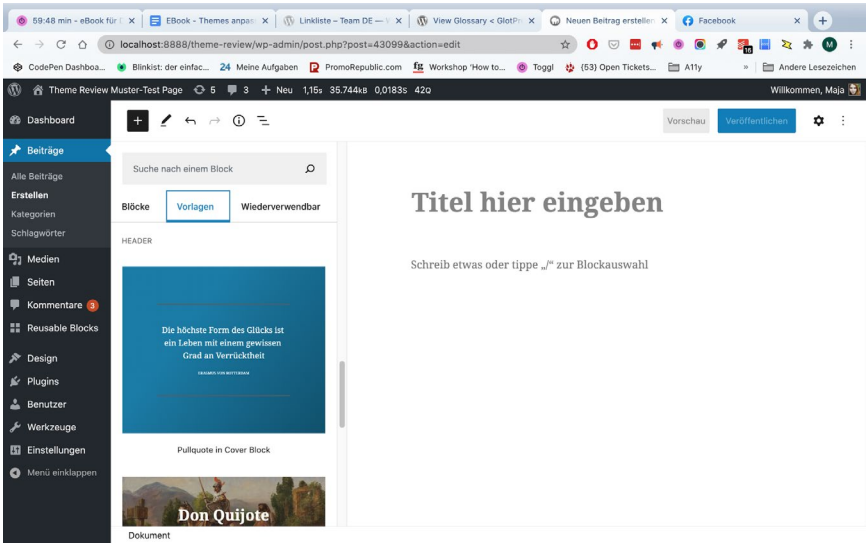
Zum Code →

Konkretes Beispiel mit einem Block-Pattern:

```
/* Register Block Patterns */

register_block_pattern(
        'pullquote-cover-block',
        array(
                'title' => __( 'Pullquote in Cover Block', 'starter-child' ),
                'description' => _x( 'Pullquote over a cover block with color gradient', 'Block pattern description', 'starter-
                'categories'  => array('header'),
                'content'     => "<!-- wp:cover {\"minHeight\":575,\"minHeightUnit\":\"px\",\"gradient\":\"blue-sapphire-to-cel
<div class=\"wp-block-cover alignfull has-background-dim has-background-gradient has-blue-sapphire-to-celadon-blue-gradient-bac
<div class=\"wp-block-columns alignwide\"><!-- wp:column {\"width\":12} -->
<div class=\"wp-block-column\" style=\"flex-basis:12%\"><!-- wp:spacer -->
<div style=\"height:100px\" aria-hidden=\"true\" class=\"wp-block-spacer\"></div>
<!-- /wp:spacer --></div>
<!-- /wp:column -->

<!-- wp:column -->
<div class=\"wp-block-column\"><!-- wp:pullquote {\"textColor\":\"white\",\"align\":\"full\",\"className\":\"is-style-default\"
<figure class=\"wp-block-pullquote alignfull is-style-default\"><blockquote class=\"has-text-color has-white-color\"><p>Die höc
<!-- /wp:pullquote --></div>
<!-- /wp:column -->

<!-- wp:column {\"width\":12} -->
<div class=\"wp-block-column\" style=\"flex-basis:12%\"><!-- wp:spacer -->
<div style=\"height:100px\" aria-hidden=\"true\" class=\"wp-block-spacer\"></div>
<!-- /wp:spacer --></div>
<!-- /wp:column --></div>
<!-- /wp:columns --></div></div>
<!-- /wp:cover -->",
        )
);
```

Zum Code →

Sobald das in der functions.php eingefügt wurde, findet man den Block-Pattern im Block-Editor:



Übersetzungen

Um die Namen der registrierten Farben und die Namen der Block-Stile übersetzen zu können, muss im ersten Schritt der Code übersetzbar sein. Dies ist in den Code-Beispielen bereits vorhanden.

Als nächsten Schritt muss man dann die Strings in die .po-Datei übertragen. Dort kann man dann die Übersetzungen vornehmen und daraus die .mo-Datei erstellen. Dies kann man mit dem Programm Poedit erstellen.

Danach müssen dann die Übersetzungsdateien im Theme geladen werden.

Konkret sieht das wie folgt aus:

1. Zunächst den Ordner „languages“ im Child-Theme erstellen.
2. Dort die .po-Datei aus dem Parent-Theme kopieren (diese liegt im Ordner wp-content/languages/themes) und den Namen des Parent-Themes löschen, sodass nur noch das Sprachkürzel übrigbleibt.
3. Dann die Übersetzungs-Dateien mit folgendem Code laden:

```php
// Load translation files from your child theme instead of the parent theme
function starter_child_locale() {
        load_child_theme_textdomain( 'starter-child', get_stylesheet_directory() . '/languages' );
}
add_action( 'after_setup_theme', 'starter_child_locale', 11 );
```

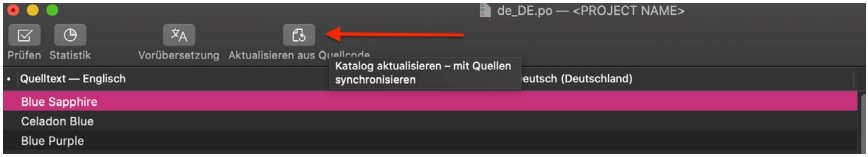
Zum Code →

4. Damit nun noch die Strings aus dem Child-Theme geladen werden, folgenden Code am Anfang der .po-Datei einfügen (den Header ersetzen) und speichern (über einen Text-Editor):

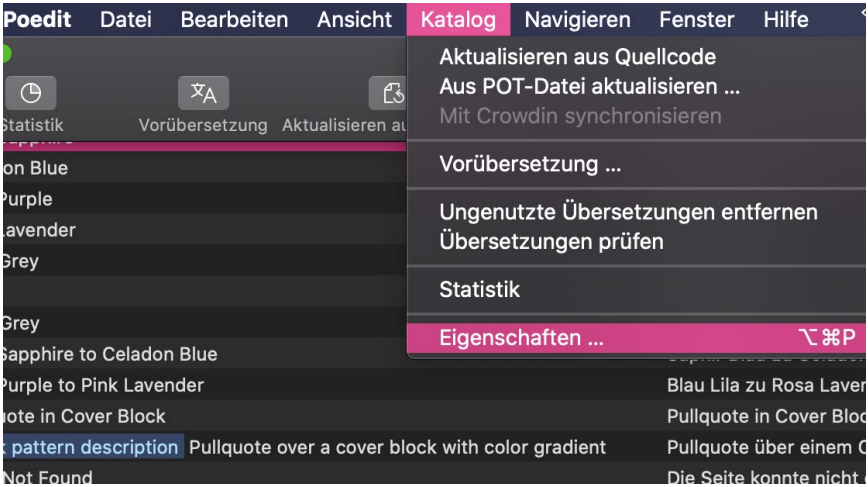
```
# Translation of Themes - Twenty Twenty in German
# This file is distributed under the same license as the Themes - Twenty Twenty package.
msgid ""
msgstr ""
"Project-Id-Version: <PROJECT NAME>\n"
"POT-Creation-Date: 2020-11-16 13:13+0100\n"
"PO-Revision-Date: \n"
"Last-Translator: \n"
"Language-Team: \n"
"Language: de_DE\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
"Plural-Forms: nplurals=2; plural=n != 1;\n"
"X-Poedit-SourceCharset: UTF-8\n"
"X-Poedit-KeywordsList: __;_e;__ngettext:1,2;_n:1,2;__ngettext_noop:1,2;"
"_n_noop:1,2;_c,_nc:4c,1,2;_x:1,2c;_nx:4c,1,2;_nx_noop:4c,1,2;_ex:1,2c;"
"esc_attr__;esc_attr_e;esc_attr_x:1,2c;esc_html__;esc_html_e;esc_html_x:1,2c\n"
"X-Poedit-Basepath: ../..\n"
"X-Textdomain-Support: yes\n"
"X-Generator: Poedit 2.2.1\n"
"X-Poedit-SearchPath-0: starter-child\n"
"X-Poedit-SearchPath-1: twentytwenty\n"
```

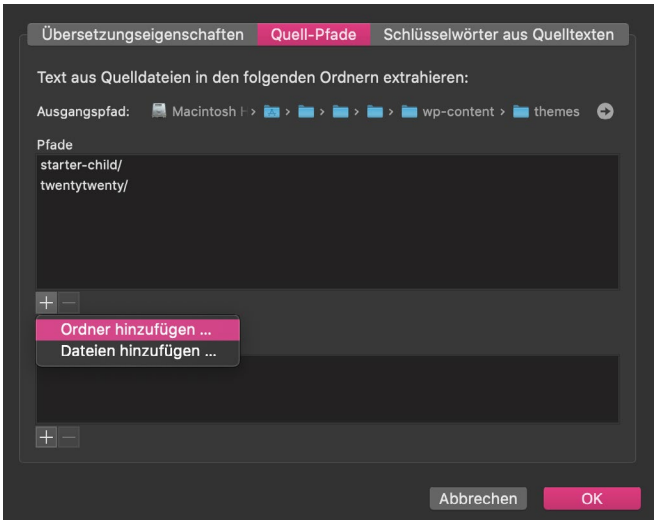
Zum Code →

5. Die .po-Datei mit dem Programm Poedit öffnen. Hier sieht man nun die Strings und eventuell auch die Übersetzungen dazu, falls bereits vorhanden. Auf das Symbol „Aktualisieren aus Quellcode“ klicken.

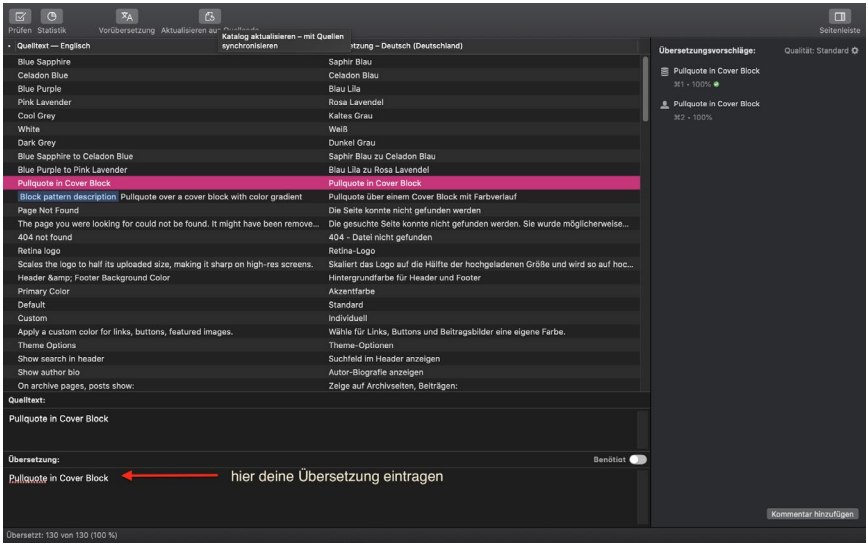


6. Für den Fall, dass man die Strings aus dem Parent-Theme bearbeiten möchte, kann man diese ebenfalls in die .po-Datei laden. Dazu auf „Katalog -> Eigenschaften -> Quellpfade“ gehen. Hier im oberen Feld „Pfade“ den kompletten Ordner des Parent-Themes hinzufügen. Die .po-Datei abspeichern.





7. Nun die Strings übersetzen und die .po-Datei speichern. Dabei wird automatisch die .mo-Datei erstellt.



8. Jetzt kann man überprüfen, ob die Übersetzung korrekt geladen wird, zum Beispiel indem man in den Block-Editor geht und über eine Farbe hovert. Wenn man die Farbnamen übersetzt hat, erscheint hier die Übersetzung.

Kompletter Code des Starter-Themes

Alle Code-Beispiele aus diesem Artikel, gesammelt im Starter Theme, befinden sich auf GitHub: github.com/00travelgirl00/starter-theme.

Welche Plugins können eine sinnvolle Ergänzung darstellen?

Es gibt sehr viele Plugins, die den Block-Editor ergänzen, vor allem Kollektionen an Blocks, die man hinzufügen kann. Um einen besseren Überblick zu bekommen, gibt es mittlerweile die Kategorie „Blocks“ im Plugin-Verzeichnis.

Es soll demnächst auch die Möglichkeit geben, dass man über den Block-Editor direkt Plugins installieren kann. Etwas ärgerlich ist, dass es recht wenig Einzel-Blöcke gibt, die man per Plugin ergänzen kann. Leider sind auch nicht alle Block-Kollektionen barrierefrei, was es erschwert, Inhalte zu erstellen, die von allen Personen konsumiert werden können. Dies betrifft vor allem Blöcke wie Accordions, Slider, Tabellen usw.

Daher lohnt es sich, etwas Zeit zu investieren und sich mehrere Block-Kollektionen genauer anzuschauen. Dann kann man sich ein, zwei oder auch drei Plugin-Kollektionen heraussuchen, mit denen man gerne arbeitet. Je nach Projekt kann man dann schauen, welche Block-Kollektion sich für das jeweilige Projekt eignet.

Gute Erfahrungen habe ich mit den Block-Kollektionen „Gutenberg Blocks by Kadence Blocks“ und „Gutenberg Blocks – Ultimate Addons for Gutenberg“ gemacht.

Weitere Plugins, die sinnvoll sein können:

Find my Blocks

„Find my Blocks“ gibt eine Übersicht, wo die Blöcke verwendet wurden. Dies kann sehr hilfreich sein, wenn man zum Beispiel eine Block-Kollektion ersetzen möchte oder wenn ein Relaunch ansteht.

Block Manager

Es gibt viele Plugins, die Blöcke in Gutenberg verwalten können. Ich kann zum einem den Gutenberg Block Manager und den Disable Gutenberg Blocks – Block Manager empfehlen. Mit diesen Plugins kann man Blocks für den Block-Editor deaktivieren. Dies kann Sinn machen, um die Verwendung des Block-Editors zu vereinfachen und um alle Blöcke, die für die jeweilige Webseite nicht benötigt werden, zu deaktivieren.

EditorsKit

Mit dem Plugin „Editors Kit“ kann man ebenfalls Blöcke deaktivieren. Zusätzlich kann man aber auch weitere Formatierungsoptionen ein und ausschalten. Für den Cover-Block werden außerdem weitere Block-Styles hinzugefügt.

Alles in allem ist dieses Plugin recht umfangreich und kann eine gute Ergänzung für Kundenprojekte sein. Es empfiehlt sich, das Plugin direkt zu Beginn des Projekts hinzuzufügen, um die entsprechenden CSS-Klassen mit zu stylen.

Reusable Blocks Extended

„Reusable Block Extended“ legt einen Menüpunkt im Dashboard an, um die wiederkehrenden Blöcke anzuzeigen.

Fazit

Der Block-Editor bietet viele Möglichkeiten, die Webseite und den Editor für Kunden zu personalisieren. Es dauert am Anfang vielleicht ein wenig, um sich einzuarbeiten und für sich selbst einen Workflow mit passenden Themes, Block-Kollektionen und weiteren Plugins zu erarbeiten, aber dann kann man recht flexibel und individuell die Seiten bauen.

Ich arbeite mit dem Block-Editor wesentlich lieber als mit einem Page-Builder und freue mich sehr auf die neue Funktion des Full-Site-Editings, mit dem man als Designer*in noch viel flexibler wird.

Foto: Ali Mokhtari, Köln

Die Autorin

Britta Kretschmer

Britta Kretschmer ist Online-Redakteurin, Autorin und Ex-Wissenschaftlerin, die lange Jahre am Kölner Universitätsklinikum in Lehrprojekten gearbeitet hat. Sie gestaltet mittels WordPress Blogs und Websites und bietet mit Internetkurse Köln WordPress Coachings und Tutorials an.

Britta lebt und arbeitet seit vielen Jahren in Köln. Sie mag Bücher, Filme und Rockmusik und liebt ihre Katze Luz.

Die Gutenberg-Breiten alignfull und alignwide – und wie man damit Seiten gestalten kann

Als WordPress 5.0 im Dezember 2018 mit Gutenberg als neuem Editor an den Start ging, brachte der nicht nur eine Menge Blöcke mit, sondern auch die beiden neuen Gutenberg-Breiten alignfull und alignwide.

Blöcke, das haben wir schnell gelernt, repräsentieren einzelne Elemente einer Webseite, also zum Beispiel Absätze, Überschriften, Bilder, Galerien, Listen, Zitate etc. Mittels der beiden neuen Breiten ist es seither möglich, einzelne Blöcke breiter auszugeben, als die Inhaltsbreite dies vorgibt. Zu Beginn galt dies vor allem Bildern und Galerien. Mittlerweile ist es aber durchaus möglich, mit ihnen Seiten zu gestalten. Doch das ist leider nicht immer so einfach, wie es auf den ersten Blick erscheinen mag. Der Grund dafür hat viel damit zu tun, dass für die Darstellung aller Gutenberg-Breiten, also alignfull und alignwide genauso wie alignnone, alignleft und alignright, das aktive Theme verantwortlich ist. Will man darauf Einfluss nehmen, muss man sich die Vorgehensweise des Themes anschauen und sie mehr oder weniger individuell modifizieren.

Vorausschau auf das Kapitel

- Im folgenden Kapitel geht es zuerst darum, zu verstehen, welche Voraussetzungen für die beiden Gutenberg-Breiten nötig sind, damit WordPress-Benutzer*innen sie überhaupt einsetzen können.
- Im zweiten Schritt schauen wir uns an, was genau alignfull und alignwide bedeuten und wie sich einzelne Elemente mit ihnen gestalten lassen.

- Im dritten Teil kommt die Gruppierung von Elementen ins Spiel. Damit lassen sich Elemente verschachteln und somit ganze Seiten gestalten. Dafür habe ich einen Seitenentwurf erstellt.
- Schließlich zeige ich anhand von dreien der meistgenutzten freien Themes, wie sich durch das Bearbeiten der CSS-Angaben dieser Seitenentwurf umsetzen lässt. Es handelt sich dabei um die Themes Neve (Version: 2.9.5, ThemeIsle), Ocean WP (Version: 2.0.2, Nick) und Hestia (Version: 3.0.8, ThemeIsle). Als Referenz dient das Standard-Theme Twenty Twenty (Version: 1.6, WordPress-Team).

Welche Voraussetzungen sind nötig?

Gutenberg an sich funktioniert grundsätzlich mit jedem Theme. Nicht grundsätzlich mit jedem Theme funktionieren hingegen die beiden Gutenberg-Breiten alignwide und alignfull. Denn das ist wichtig zu wissen: ***Für die Darstellung der beiden Breiten ist das Theme verantwortlich.*** Das heißt: Solange das Theme nicht die Grundlage geschaffen hat, werden einzelne Elemente niemals die Option für volle oder weite Breite zur Verfügung stellen. CSS allein reicht da nicht. Es braucht zudem den Theme Support. Der findet sich im Allgemeinen in der ***functions.php*** des Themes und lautet:

```
add_theme_support( 'align-wide' );
```

Eine weitere Voraussetzung für das Arbeiten mit alignfull und alignwide ist das Fehlen der Sidebars rechts/links. Immerhin sprechen wir hier von etwas, das die volle Fensterbreite nutzen soll. Da wäre eine Sidebar nur im Weg. Ohnehin arbeiten aber immer mehr Themes ohne Sidebar oder bieten die Möglichkeit, diese zu deaktivieren. Sei es für die ganze Website oder sogar für einzelne Seiten oder Beiträge.

Was bedeuten alignwide und alignfull?

Sagen wir, ein Theme benutzt als Standardbreite für den Content 800 Pixel. Bislang bedeutete dies, dass jedes Element innerhalb dieses Inhaltsbereiches entsprechend auch nur 800 Pixel Breite in Anspruch nehmen konnte. Für die Lesbarkeit von Texten sind 800 Pixel sicherlich hervorragend geeignet. Bilder oder Galerien hingegen wirken so aber recht klein.

Die Gutenberg-Breiten alignwide und alignfull sorgen hier nun für Abhilfe. Mit ihnen lassen sich einzelne Elemente somit breiter ausgeben als die normale Inhaltsbreite. Wie breit das genau ist, hängt allerdings vom Theme ab. Dies ist ohnehin die wichtigste Erkenntnis überhaupt: Das Theme entscheidet, wie voll alignfull und wie breit alignwide dargestellt werden. Dabei kann auch das dahinterstehende Vorgehen sehr unterschiedlich sein, sodass es für die Bearbeitung nicht die eine Lösung gibt, die für alle Themes gilt.

Wie voll ist alignfull?

Eigentlich sollte die Begrifflichkeit selbsterklärend sein: alignfull bedeutet, dass das Element die volle zur Verfügung stehende Breite nutzt. Auf einem Smartphone unterscheidet sich diese zwar kaum von der Inhaltsbreite. Auf dem Desktop hingegen kann das sehr breit sein: so breit, wie das Browserfenster es zulässt.

Allerdings sieht das nicht jedes Theme so. Wenn es um die konkrete Umsetzung meines Entwurfes geht, werden wir sehen, dass Ocean WP unter alignfull nur die volle Breite des Hauptcontainers versteht.
Noch weniger eindeutig verhält es sich mit alignwide.

Wie weit ist alignwide?

Auch hier sollte der Wortsinn schon ein wenig weiterhelfen: alignwide bedeutet weite Ausrichtung. Mit anderen Worten: Wir reden hier von

etwas, das ***breiter als die Inhalts- und schmaler als die volle Fensterbreite*** ist. Wie breit dies genau ist, regelt einmal mehr das Theme.

Wenn das Theme wie Ocean WP hingegen schon alignfull nicht breiter ausgibt als den Hauptcontainer, ergibt sich, dass die weite Breite sogar schmaler ist als die Inhaltsbreite.

Einem Block die Gutenberg-Breiten alignfull oder alignwide zuordnen

Ob ein Theme die Gutenberg-Breiten alignfull und alignwide unterstützt, lässt sich daran erkennen, dass die meisten Blöcke die entsprechende Option in ihrer jeweiligen Werkzeugleiste zeigen.

Bearbeitung eines Bild-Blocks: In der Werkzeugleiste erscheinen die Gutenberg-Breiten alignfull und alignwide.

Diese Option gibt es nicht für alle Blöcke. Da aber die Liste derer, die über diese Option verfügen, sehr lang ist, geht es schneller, nur die zu nennen, die sie nicht haben.

Wesentlich sind hier:

- Absatz-Block
- Sortierter und unsortierter Listen-Block
- Zitat-Block
- Vers-Block
- Buttons-Block
- Vorformatiert-Block
- Code-Block
- Shortcode-Block

Allen diesen genannten Blöcken ist eines gemeinsam: Sie stellen im Panel „Erweitert“ die Option zur Verfügung, eine CSS-Klasse zu hinterlegen. Somit können auch diese Blöcke alignfull oder alignwide sein. Allein für den Shortcode-Block bietet WordPress diese Option nicht. Aber auch hier gibt es eine Möglichkeit: Ein den jeweiligen Shortcode umschließender DIV-Bereich mit der Klasse .alignfull oder .alignwide sorgt für die gewünschte Breite.

Damit das Element im Frontend dann auch in voller oder weiter Breite erscheint, müssen aber passende CSS-Definitionen vorhanden sein. Wie genau die aussehen? Richtig: Dafür ist das aktive Theme verantwortlich.

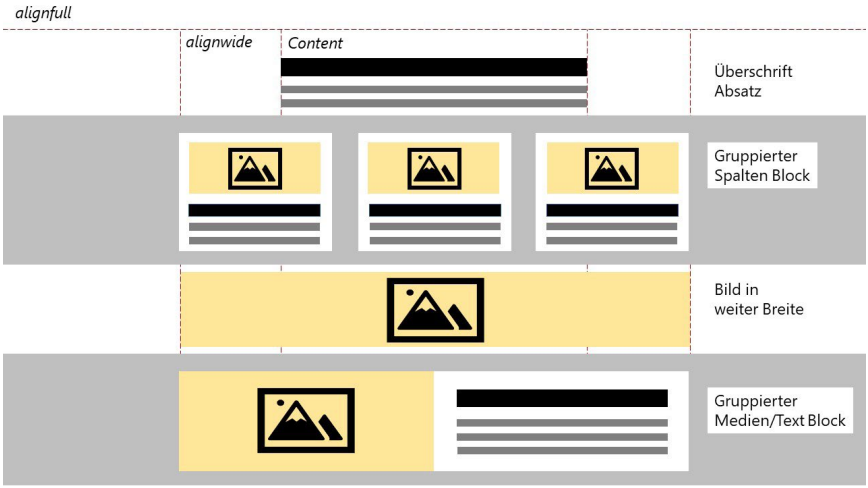
Seiten gestalten mit den Gutenberg-Breiten alignfull und alignwide

Gutenberg ist mehr als nur der neue Editor für die Bearbeitung von Seiten und Beiträgen. Gutenberg hat zudem den Anspruch, auch eine Art Baukastensystem zu sein, mit dem sich ganze Seiten gestalten lassen. Dabei ist die Möglichkeit, einzelne Elemente in unterschiedlichen Breiten auszugeben, nur ein Baustein. Viel wesentlicher ist die Möglichkeit, Elemente miteinander zu verschachteln. Denn nur so lassen sich komplexere Layouts für Seiten erstellen.

Früher war es mit dem TinyMCE, heute besser bekannt als Classic Editor, kaum möglich, auch nur geringfügig HTML einzusetzen. Der Grund dafür lag daran, dass der TinyMCE schon einfache Auszeichnungen wegkorrigiert hat. Von Verschachtelungen mit Bereichen ganz zu schweigen.

Der Entwurf

Gutenberg sei Dank ist aber genau diese Verschachtelung nun möglich. Als Grundlage für die Illustration dient mir nun ein Entwurf, bei dem ich mich auf wesentliche Komponenten beschränke. Vielleicht nicht in genau dieser Reihenfolge, so handelt es sich dabei doch um Bereiche, mit denen sich eine Seite gestalten lässt.



Der Entwurf, der dem Test als Grundlage dient.

Bei meinem Entwurf konzentriere ich mich also auf verschachtelte Container. Diese sind:

- ein gruppierter Spalten-Block,
- ein gruppierter Medien-und-Text-Block.

Der Orientierung dienen eine Überschrift und ein Absatz in normaler Breite und ein Bild in weiter Breite.

Wie funktioniert der Gruppe-Block?

Die Gutenberg-Macher haben sich dazu entschieden, statt eines Wrapper-Blocks den sogenannten Gruppe-Block einzuführen. Dazu gehört die Möglichkeit, Blöcke alleine oder mehrere Blöcke zu einem Gruppe-Block umwandeln zu können.

Wie ein Wrapper umschließt die Gruppe ihren Inhalt mit zwei verschachtelten DIV-Bereichen. Mit dem äußeren dieser beiden DIV-Bereiche lässt sich die Gruppe an sich gestalten. Dieser Bereich kann also zum Beispiel die Breite alignfull erhalten. Auch kann man ihm eine Hintergrundfarbe zuordnen. Der innere der beiden DIV-Bereiche hingegen dient einzig dem Zweck, alle Elemente, die er umschließt, zu positionieren und gegebenenfalls auch ihre Ausgabegröße zu begrenzen. Weshalb es schön wäre, ließe sich dieser innere Bereich im Rahmen der Bearbeitungsmöglichkeiten des Gruppe-Blocks modifizieren. Bearbeiten lässt sich dieser innere DIV-Bereich eines Gruppe-Blocks aber leider nur über das CSS des Themes.

Einen gruppierten Spalten-Block erstellen

Im Folgenden verwende ich für alle Test-Themes einen gruppierten Spalten-Block. Das Erstellen einer solchen Verschachtelung kann von außen nach innen erfolgen, also mit dem Gruppe-Block beginnen. Um den

Begriff des Gruppierens zu verdeutlichen, arbeite ich hier aber von innen nach außen. Das Erstellen einer solchen Gruppierung funktioniert also wie folgt:

1. Einen Spalten-Block erstellen und die Anzahl der Spalten festlegen. In meinem Fall sind es drei Spalten.
2. Jede Spalte mit einem Bild-, einem Überschrift- und einem Absatz-Block befüllen.
3. Die drei Elemente der jeweiligen Spalte markieren und per Klick auf das erste Icon der Werkzeugleiste in einen Gruppe-Block umwandeln.
4. Für diese Gruppe die Hintergrundfarbe weiß und die Textfarbe schwarz wählen.
5. Nun dem Spalten-Block die Breite alignwide zuordnen.
6. Den Spalten-Block in einen Gruppe-Block umwandeln. Dies geschieht wieder über Klick auf das erste Icon der Werkzeugleiste.
7. Diesem Gruppe-Block die Breite alignfull und eine Hintergrundfarbe zuordnen.

Einen gruppierten Medien-und-Text-Block erstellen

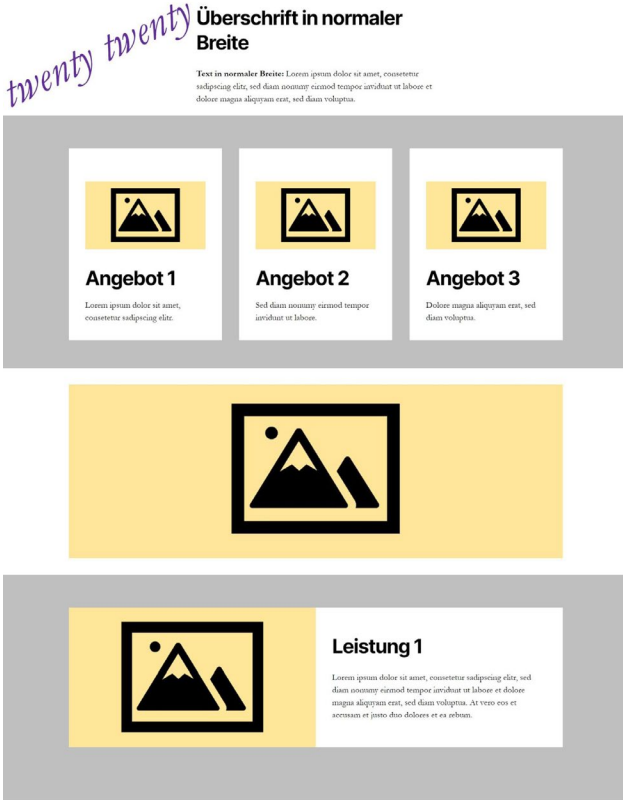
Das Vorgehen für den Medien-und-Text-Block entspricht dem für den gruppierten Spalten-Block weitgehend:

1. Einen Medien-und-Text-Block erstellen. Ein Bild auswählen sowie eine Überschrift erstellen und etwas Text einfügen.
2. Dem Medien-und-Text-Block die Hintergrundfarbe weiß und die Textfarbe schwarz zuordnen. Außerdem „auf Mobilgeräten stapeln“ und „Bild zuschneiden, um die gesamte Spalte zu füllen“ auswählen.
3. Sicherstellen, dass für diesen Block die Breite alignwide aktiviert ist.
4. Den Medien-und-Text-Block zum Gruppe-Block umwandeln.
5. Dem Gruppe-Block die Breite alignfull und eine Hintergrundfarbe zuordnen.

Tipp: Gruppe-Blöcken am besten immer eine Hintergrundfarbe zuordnen, selbst wenn gar keine gewünscht ist. In dem Fall für den Hintergrund weiß beziehungsweise die Hintergrundfarbe des Inhaltsbereiches wählen. Auf diese Weise lassen sich Abstände leichter festlegen. Viele Themes liefern bereits die passende CSS-Definition für Gruppen mit Hintergrund.

Theme Twenty Twenty

Als Referenz dient Theme Twenty Twenty, denn das liefert für alle Eventualitäten die passenden CSS-Angaben.



Theme Twenty Twenty versteht die Gutenberg-Breiten alignfull und alignwide.

Der Screenshot zeigt: Theme Twenty Twenty setzt alles genauso um wie gedacht. Die weite Breite ist entsprechend schmaler als die volle Breite. Dabei beanspruchen alignwide-Elemente verschachtelt und alleine die gleiche Breite. Auch mobil sieht alles gut aus. Es ist also keinerlei Veränderung der hinterlegten CSS-Definitionen nötig.

Twenty Twenty bietet im Übrigen keine Einstellungsoption für die Inhaltsbreite und nutzt hierfür sowie für alle Elemente, die nicht alignfull oder alignwide sind, die zentralen 58rem. Für alignfull setzt es die die volle Fensterbreite, für alignwide die zentralen 120rem an.

Hinweis: Alle im Weiteren genannten CSS-Änderungen gehören in das zusätzliche CSS des Themes, zu erreichen über den Customizer. Die Erstellung eines Child-Themes ist nicht nötig.

Theme Neve

Neve bietet die Layout-Option, mit der sich die Inhaltsbreite bestimmen lässt. Ich habe hier eine Breite von 800 Pixeln gewählt. Überdies lässt sich einstellen, welches Layout für Seiten und welches für Beiträge als Standard gelten soll: Contained oder Full Width. Für beide Layouts lässt sich zudem die Sidebar abwählen. Diese Einstellungen lassen sich aber auch pro Seite oder Beitrag individuell wählen.

Das Theme sorgt übrigens dafür, dass bei Nutzung des Contained Layouts mit Sidebar Elemente auf der Seite niemals breiter als der Inhaltsbereich sein können. Also habe ich für meinen Entwurf das Contained Layout ohne Sidebar gewählt. Somit erscheint der normale Inhaltsbereich gut lesbar schmal, während für die breiten Container der nötige Platz existiert.

Mit diesen Einstellungen realisiert Neve meinen Entwurf grundsätzlich selbst. Das Ganze funktioniert auch für Seiten mit der Einstellung Full Width ohne Sidebar.

Theme Neve versteht Gutenberg, kennt aber keine Paddings für Gruppen mit Hintergrund.

Auch auf Mobilgeräten funktioniert alles wie gewünscht. Einzig fällt auf, dass die Gruppen ein wenig Innenabstand vertragen könnten.

Hier kommt der Tipp wieder ins Spiel, gruppierten Elementen eine Hintergrundfarbe zu geben. Mittels der Klasse *.wp-block-group.has-background* lässt sich allen Gruppen-Blöcken mit Hintergrundfarbe der gewünschte Innenabstand zuordnen. Das gilt dann natürlich auch bei weißer Hintergrundfarbe. Auf diese Weise gibt es immer genug Weißraum.

Neve Beitragsseiten

Schwieriger wird es hingegen mit Beiträgen. Auch hier gelten die Optionen:

- Contained mit oder ohne Sidebar rechts/links
- Full Width mit oder ohne Sidebar rechts/links

Ohne Korrektur wird der innere Container der Gruppe-Blöcke auf Desktops linksbündig und sein Inhalt schmaler (nur 530px) ausgegeben. Für diese Definition nutzt Neve entsprechend diesen inneren der beiden DIV-Bereiche einer Gruppe.

CSS für Theme Neve

```
/* === Theme Neve === */
/* Innenabstand für Gruppen mit Hintergrund */
.wp-block-group.has-background {
padding: 2em;
}
/* Vor- und Nachabstand für alignwide */
.alignwide {
margin-top:32px; margin-bottom:32px;
}
/* Beitragsseiten: Innere Container zentriert */
@media (min-width: 960px) {
.single-post-container .entry-content .alignfull > [class*="__inner-container"],
.single-post-container .entry-content .alignwide > [class*="__inner-container"] {
margin: 0 auto!important; }
}
```

Zum Code →

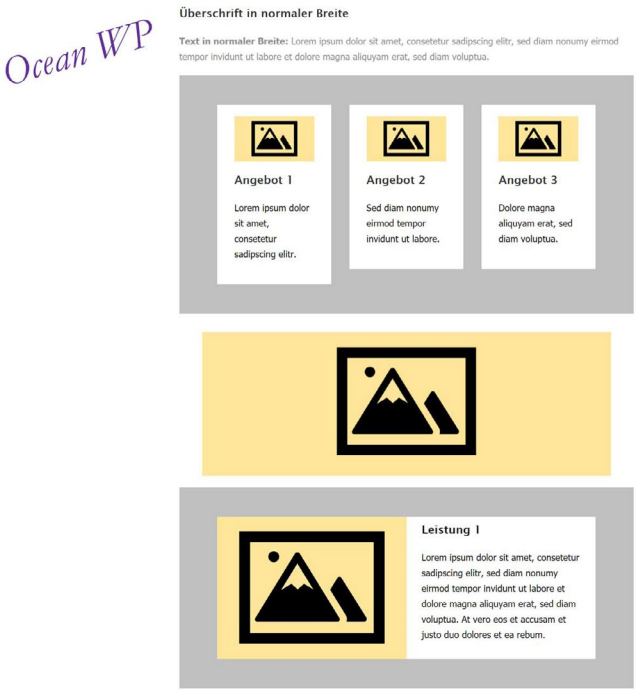
Theme Ocean WP

Auch Ocean WP bietet unterschiedliche Layouts für Seiten und Beiträge. Sie können ***weit***, ***boxed*** oder ***getrennt*** sein, jeweils mit oder ohne Sidebars. Außerdem kann die Breite bei Layouts ohne Sidebar „die ganze Breite“ versus „100% ganze Breite“ bedeuten. Diese Unterschiede muss man erst einmal verstehen.

„Weit“ bedeutet im Gegensatz zu „boxed“ eigentlich nur, dass der Hauptcontainer keinen Rahmen hat. Und im Gegensatz zu „getrennt“ heben sich Inhaltsbereich und Widgets auch nicht farblich vom Hintergrund ab.

Ganze Breite bedeutet, dass die Elemente auf der Seite die gewählte Breite des Hauptcontainers einnehmen. Demgegenüber bedeutet „100% ganze Breite“, dass der Hauptcontainer die komplette zur Verfügung stehende Fensterbreite nutzt.

Auch für Ocean WP habe ich eine Breite von 800 Pixel – sowohl für Seiten als auch für Beiträge – und für alle das Layout weit mit ganzer Breite gewählt.



Bei Ocean WP erscheint die Gutenberg-Breite alignwide schmaler als der Content.

Der Screenshot zeigt: Alles ist auf die 800 Pixel des Hauptcontainers begrenzt. alignfull nutzt dabei die vollen 100 Prozent, alignwide beschränkt sich auf 90 Prozent. Nur bei 100% ganze Breite würden die alignfull-Elemente an die Grenzen der Fensterbreite stoßen. Aber das gilt dann gleich für jeglichen Inhalt, so auch für jeden Absatz, der damit auf Desktops kaum mehr lesbar ist.

Damit alignfull nun die komplette Fensterbreite und alignwide einen Bereich zwischen Inhalts- und Fensterbreite einnehmen, braucht es grundlegende Definitionen. Außerdem braucht es für alignwide Angaben, die dafür sorgen, dass solche Elemente mit und ohne Verschachtelung gleich breit erscheinen. Auch für Beitragsseiten und die mobile Darstellung braucht alignwide Anpassungen.

CSS für Theme Ocean WP

```css
/* === Ocean WP === */
/* Definition für volle Breite */
.alignfull {
margin-left: calc( -100vw / 2 + 100% / 2 );
margin-right: calc( -100vw / 2 + 100% / 2 );
max-width: 100vw;
width: 100vw;
}
/* Aufheben der vorhandenen Definition für weite Breite */
.alignwide {
margin: unset;
width: unset;
max-width: unset;
}
/* Elemente mit weiter Breite ohne Verschachtelung */
.alignwide {
margin: 32px -112px;
max-width:1024px;
}
/* auf Beitragsseiten */
.single-post .alignwide {
margin: 32px auto;
}
/* Anpassung für Mobilgeräte */
@media (max-width:1023px) {
.alignwide {
max-width:800px !important;
margin: 32px auto;}
}
/* Definition für verschachtelte Elemente mit weiter Breite */
.alignfull [class*="__inner-container"] > .alignwide {
margin-left: auto!important;
margin-right: auto!important;
max-width: 1024px!important;
}
```

Zum Code →

Theme Hestia

Hestia ist der einzige Testkandidat, der die Gutenberg-Breite alignwide und ihre Außenabstände berechnet und das Ergebnis in das style-Attribut des Elementes schreibt. Mit meinem Entwurf kann Hestia jedenfalls fast gar nichts anfangen.

Theme Hestia berechnet die Gutenberg-Breite alignwide.

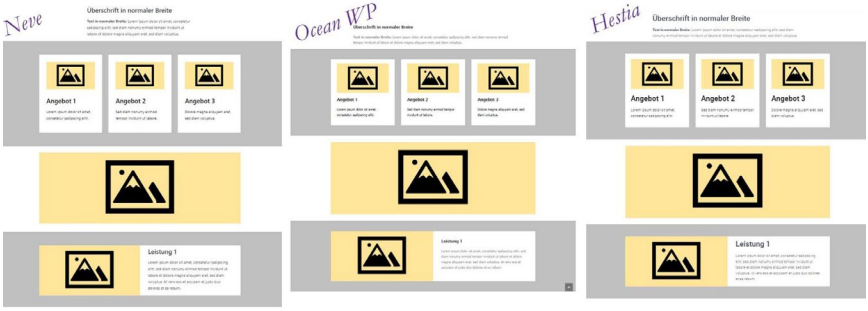
Theme Hestia gibt ohne eigene Bearbeitung bei Verschachtelungen alignwide und alignfull gleich breit aus. Einzelne alignwide-Elemente hingegen stellt das Theme schmaler dar. Dabei berechnet es die Abstände rechts und links. Schließlich fehlen auch die nötigen Innenabstände.

Eine Option, mit der sich die gewünschte Containerbreite festlegen ließe, bieten die Theme-Einstellungen nicht. Insgesamt stellt sich Hestia für die Bearbeitung recht hartnäckig dar.

CSS für Theme Hestia

```
/* === Theme Hestia === */
/* Verschachtelte Container und alignwide-Elemente für kleine Mobilgeräte und Tablets */
@media (min-width: 320px) {
.alignfull [class*="__inner-container"] > .alignwide {
margin-left: auto!important;
margin-right: auto!important;
max-width: 800px!important;
}
.alignwide {
margin: 32px auto!important;
max-width: 800px!important;}
}
/* Verschachtelte Container und alignwide-Elemente für mindestens 1024 Pixel Breite */
@media (min-width: 1024px) {
.alignwide {
margin: 32px -90px!important;
max-width: 800px!important;
width: 800px!important;
}
.alignfull [class*="__inner-container"] > .alignwide {
margin-left: auto!important;
margin-right: auto!important;
max-width: 800px!important;
width: 800px!important;}
}
/* Verschachtelte Container und alignwide-Elemente für Desktops größer 1200 Pixel Breite */
@media (min-width: 1200px) {
.alignwide {
margin: 32px -136px!important;
max-width: 1024px!important;
width: 1024px!important;
}
.alignfull [class*="__inner-container"] > .alignwide {
margin-left: auto!important;
margin-right: auto!important;
max-width: 1024px!important;
width: 1024px!important;}
}
/* Innenabstand für Elemente mit Hintergrund */
.wp-block-group.has-background {padding: 2em;}
```

Zum Code →



Am Ende des Tages: Alle drei Themes nach der Bearbeitung.

Fazit

Alles könnte so einfach sein – ist es aber nicht. Das hat viel mit der Abhängigkeit vom Theme zu tun. Ohne sie kann das Ganze nicht funktionieren – durch sie kommt es aber immer wieder zu Problemen. Das kann auch ambitionierte Benutzer*innen schnell an die eigenen Grenzen führen.

Wer Gutenberg als Baukastensystem versteht, braucht also ein Theme, das dieser Idee Rechnung trägt. Schön sind natürlich immer Optionen für grundsätzliche Einstellungen, so vor allem für die Breite der Container und das Abschalten der Sidebar. Wichtiger wären aber die passenden CSS-Definitionen, mit denen sich alle denkbaren Verschachtelungen darstellen lassen.

In meinem Entwurf ist zum Beispiel gar nicht die Möglichkeit berücksichtigt, dass der Inhalt eines alignfull-Bereiches ja auch nur inhaltsbreit sein könnte. Twenty Twenty kann das problem-

los umsetzen. Neve und Ocean WP können es auf Beiträgen, nicht aber auf Seiten. Und Hestia kann es gar nicht.

Alles in allem sollte ein Gutenberg-Theme also am besten wie ein weißes Blatt Papier daherkommen, auf dem sich mit Gutenberg malen lässt. So ist es sehr willkommen, dass das neueste Standardtheme, Twenty Twenty-One, genau in diese Richtung geht. Entsprechend setzt es in seiner aktuellen Version 1.1 – genauso wie Twenty Nineteen (Version 1.9) – meinen Entwurf perfekt um.

Weitere Themes, die das auch oder mindestens so gut wie Neve können, sind:

- Atomic Blocks (Version: 1.2.6, Atomic Blocks),
- Blocksy (Version 1.7.60, CreativeThemes – bei Seiten ohne Sidebar mit schmaler Seitenbreite und weitem Inhaltsbereich),
- Chaplin (Version 2.5.17, Anders Norén – braucht nur eine Anpassung für alignwide),
- GeneratePress (Version 3.0.2, Tom Usborne – braucht Anpassung für mobile Darstellung),
- Hamilton (Version 2.0.7, Anders Norén) und
- Seedlet (Verson 1.1.2, Automattic).

Um es letztlich mit einem großen deutschen Dichter zu sagen:

Drum prüfe, wer sich ewig bindet,
Ob sich nicht das passende Gutenberg-Theme findet.

Frei nach Friedrich Schiller (Das Lied von der Glocke)

Der Autor

Bernhard Kau

Bernhard Kau ist WordPress-Entwickler und beschäftigt sich seit 2009 mit der Programmierung von Plugins. Als aktives Mitglied der internationalen WordPress-Community ist er einer der Co-Organisatoren des WordPress-Meetups Berlin und hat 2019 das WordCamp Europe – die größte WordPress-Konferenz weltweit – als Local-Lead in Berlin mitorganisiert. Neben seinem Blog, auf dem er wöchentlich aus seinem Arbeitsalltag berichtet, betreibt er als Co-Host einen Podcast zum Thema WordPress.

Custom-Post-Type-Entwicklung für den Block-Editor

Auch zwei Jahre nach der Einführung von Gutenberg sind noch nicht alle WordPress-Projekte und die darin eingesetzten Plugins an den neuen Block-Editor angepasst worden. Daher soll dieser Beitrag zeigen, was notwendig ist, um ein Plugin für den Block-Editor vorzubereiten. Hierzu beschreibt der Artikel den gesamten Entwicklungsprozess eines Custom-Post-Types für den Block-Editor.

Erstellung eines Plugins

Ein Plugin ist in WordPress im einfachsten Fall eine einzelne PHP-Datei, die einen Kommentar mit den Meta-Daten zum Plugin enthält. Es kann aber auch aus vielen Dateien bestehen, die in Komponenten aufgeteilt einem objektorientierten Ansatz folgt und sich auf viele Dateien verteilt. Um diesen Artikel aber einfach zu halten, folgen die Codebeispiele einem prozeduralen Ansatz.

Als erstes wird eine PHP-Datei in einem neuen Unterordner erstellt. Hierbei wird meist der gleiche Name für die PHP-Datei wie auch für den Ordner gewählt. Ein Beispiel wäre also eine Datei mit dem Pfad **wp-content/plugins/testimonial-cpt/testimonial-cpt.php** und folgendem Inhalt:

```php
<?php
/**
 * Plugin Name:       Testimonial Post Type
 * Plugin URI:        https://github.com/2ndkauboy/testimonial-cpt/
 * Description:       Register a post type to manage testimonials.
 * Version:           1.0.0
 * Requires at least: 5.5
 * Requires PHP:      7.2
 * Author:            Bernhard Kau
 * Author URI:        https://kau-boys.de
 * License:           GPL v2 or later
 * License URI:       https://www.gnu.org/licenses/gpl-2.0.html
 * Text Domain:       testimonial-cpt
 * Domain Path:       /languages
 */
```

Zum Code →

Der Plugin-Header benötigt lediglich einen Wert für „Plugin Name“, damit das Plugin aktiviert werden kann. Nach Möglichkeit sollten aber alle Werte ausgefüllt werden. Eine ausführlichere Version wird im Plugin-Handbook beschrieben.

Erstellung eines Custom-Post-Types

Das Plugin wird – wie der Name schon erahnen lässt – einen eigenen Inhaltstype, also einen Custom-Post-Type (CPT), für Testimonials registrieren. Dafür wird die Funktion **register_post_type** verwendet. Diese erhält als erstes Argument den Namen des Custom-Post-Types und als zweites Argument ein Array mit weiteren Parametern. Diese werden ebenfalls im Developer-Handbook näher beschrieben.

Die Erstellung eines Custom-Post-Types kann durch Generatoren sehr erleichtert werden. Hierzu kann etwa der Generator auf generatewp.com verwendet werden. Noch einfacher geht es mit der WP-CLI, dem Kommandozeilen-Tool für WordPress. Darin enthalten ist ein Befehl, der die wichtigsten Funktionen und Argumente für einen Post-Type generiert. Ein Aufruf sieht im einfachsten Fall wie folgt aus:

```
wp scaffold post-type testimonial --textdomain=testimonial-cpt
--plugin=testimonial-cpt
```

Dies erstellt eine Datei **post-type/testimonial.php** im Plugin, die in die Hauptdatei des Plugins kopiert oder über **require** in diese eingebunden werden kann. Wenn kein Generator verwendet wird, sollten mindestens folgende Zeilen enthalten sein:

```php
<?php

/**
 * Registers the `testimonial` post type.
 */
function testimonial_init() {
	register_post_type(
		'testimonial',
		array(
			'labels'       => array(
				'name'          => __( 'Testimonials', 'testimonial-cpt' ),
				'singular_name' => __( 'Testimonial', 'testimonial-cpt' ),
			),
			'public'       => true,
			'supports'     => array( 'title', 'editor' ),
			'has_archive'  => true,
			'show_in_rest' => true,
		)
	);
}
add_action( 'init', 'testimonial_init' );
```

Zum Code →

Damit der Post-Type mit dem Block-Editor funktioniert, muss unbedingt der Parameter **show_in_rest** auf **true** gesetzt werden. Dies ist notwendig, da der Block-Editor Inhalte über die WordPress REST-API in den Editor lädt und wieder speichert. Ist der Parameter **public** auf **true** gesetzt, dann können die Einträge des Custom-Post-Types auch von außen abgerufen werden. Es ist weiterhin möglich, einen eigenen REST-API Controller zu definieren, aber für die allermeisten Post-Types wird das nicht notwendig sein. Hier wird dann stattdessen der **WP_REST_Posts_Controller** Controller verwendet, womit auch für den Custom-Post-Type alle Operationen möglich sind wie für die internen Post-Types **post** und **page**.

Ein weiterer wichtiger Parameter ist **supports**, über den verschiedene Felder und Funktionen freigeschaltet werden. Aus dem Core werden folgende Werte zur Verfügung gestellt: ‚title', ‚editor', ‚comments', ‚revisions', ‚trackbacks', ‚author', ‚excerpt', ‚page-attributes', ‚thumbnail', ‚custom-fields' und ‚post-formats'. Soll der eigene Inhaltstyp also beispielsweise auch ein „Beitragsbild" haben können, dann muss noch **thumbnail** zum Array hinzugefügt werden.

Erstellung einer Custom-Taxonomy für den Post-Type

Wenn eigene Inhaltstypen erstellt werden, dann wird häufig auch eine eigene Taxonomie, also eine Custom-Taxonomy (CT), für eine Kategorisierung bzw. Verschlagwortung von Inhalten eingesetzt. Es ist grundsätzlich möglich, die Kategorien und Schlagwörter von Beiträgen zu verwenden. Um hier aber eine bessere Trennung zu erreichen, sollte eine Custom-Taxonomy gewählt werden. Auch diese kann mit den Generatoren erstellt werden. Ein Aufruf mit der WP-CLI könnte hier wie folgt aussehen:

```
wp scaffold taxonomy testimonial_type --post_types=taxonomy --
label="Type" --textdomain=testimonial-cpt --plugin=testimonial-cpt
```

Dies erstellt eine Datei **taxonomies/testimonial_type.php**, die auch wieder in die Hauptdatei kopiert oder eingebunden werden kann. Wird eine Taxonomy registriert, sollten mindestens folgende Zeilen angegeben werden:

```php
<?php

/**
 * Registers the `testimonial_type` taxonomy, for use with 'taxonomy'.
 */
function testimonial_type_init() {
	register_taxonomy(
		'testimonial_type',
		array( 'testimonial' ),
		array(
			'labels'       => array(
				'name'          => __( 'Types', 'testimonial-cpt' ),
				'singular_name' => __( 'Type', 'testimonial-cpt' ),
			),
			'hierarchical' => false,
			'public'       => true,
			'show_in_rest' => true,
		)
	);
}
add_action( 'init', 'testimonial_type_init' );
```

Zum Code →

Als erstes Argument der Funktion wird der Name der Taxonomy genannt und im zweiten ein Array von Post-Type, die mit dieser Taxonomy kategorisiert werden können. Hier muss also der zuvor registrierte Custom-Post-Type angegeben werden. Im dritten Argument folgt wieder ein Array mit weiteren Parametern, die im Handbook beschrieben werden.

Ein wichtiger Parameter ist **hierarchical**, der angibt, ob es zu einem Term der Taxonomy einen übergeordneten geben kann, wie es bei den Beitrags-Kategorien der Fall ist. Ist der Wert auf **false** gesetzt, was der

Standardwert ist, dann verhält sich die Taxonomy wie die Schlagwörter von Beiträgen, die keine Hierarchie haben.

Damit die Taxonomy mit dem Block-Editor funktioniert, ist es auch hier wieder wichtig, dass der Parameter **show_in_rest** auf **true** gesetzt ist.

Fertigstellung der ersten Plugin-Version

Das Plugin könnte nun bereits genutzt werden, um Einträge im Post-Type Testimonials zu erstellen. Hier würden dann allerdings alle Texte in englischer Sprache erscheinen, da noch keine Übersetzungen vorhanden sind. Gerade wenn das Plugin in einer deutschen WordPress-Installation zum Einsatz kommt, sollte eine solche Datei erstellt werden. Alle Strings, die potentiell übersetzbar sein sollen, werden bereits von den Generatoren mit den entsprechenden Übersetzungsfunktionen ausgestattet. In den WP-CLI Befehlen wurde hierbei auch schon jeweils die passende Textdomain für das Plugin angegeben.

Nun muss eine Vorlage für die Übersetzungen erstellt und dann ins Deutsche übersetzt werden. Auch hierzu gibt es einen WP-CLI Befehl zur Generierung einer solchen Datei:

```
wp i18n make-pot . languages/testimonial-cpt.pot
```

Dieser Befehl erstellt eine sogenannte POT (Portable Object Template) Textdatei, welche als Vorlage für die Lokalisierung in andere Sprachen dient.

Die Basissprache sollte immer amerikanisches Englisch sein. Für eine Übersetzung ins Deutsche kann dann mit Hilfe der POT-Datei eine sprachspezifische PO (Portable Object) Textdatei erstellt werden, aus der eine binäre MO (Machine Object) Datei generiert wird, welche WordPress verwendet, um alle Strings zu übersetzen.

Grundsätzlich kann man diese Schritte ohne zusätzliche Programme durchführen. Ich empfehle an dieser Stelle aber das kostenfreie Programm Poedit, welches für Windows, Linux und Mac verfügbar ist. Die Pro-Version bietet zwar einige Zusatzfunktionen für die Übersetzung von WordPress-Plugins, aber auch die freie Version reicht aus.

Poedit generiert beim Speichern der PO-Datei automatisch eine MO-Datei mit gleichem Namen im selben Ordner. Damit die Übersetzungen allerdings auch verwendet werden, muss WordPress mitgeteilt werden, wo sich diese Datei befindet. Bei Plugins, die nicht im WordPress Plugin-Directory gelistet werden, hat sich der Unterordner **languages** als Standard in vielen Plugins und Themes etabliert. Eine Datei mit der deutschen Übersetzung sollte hierbei unter de Namen **testimonial-cpt-de_DE.mo** gespeichert werden. Der Name setzt sich aus der Text-Domain gefolgt von der Locale zusammen.

Damit diese Datei im richtigen Ort gefunden wird, ist folgender PHP-Code notwendig:

```php
<?php

/**
 * Load the translation file for the plugin.
 */
function testimonial_cpt_load_textdomain() {
        load_plugin_textdomain(
                'testimonial-cpt',
                false,
                dirname( plugin_basename( __FILE__ ) ) . '/languages'
        );
}
add_action( 'plugins_loaded', 'testimonial_cpt_load_textdomain' );
```

Zum Code →

Damit ist die erste Version des Plugins fertig, und es können Inhalte im neuen Post-Type erstellt und mit der neuen Taxonomie verschlagwortet werden.

Festlegung von Standardinhalten

Wenn Custom-Post-Types eingesetzt werden, dann wird damit in der Regel ein bestimmter Zweck verbunden. Inhalte dieses eigenen Post-Types haben oft eine sehr ähnliche Struktur. Um die Gleichmäßigkeit der Struktur zu erleichtern, kann beim Registrieren des Post-Types ein Template definiert werden. Hierzu wird der Code wie folgt erweitert:

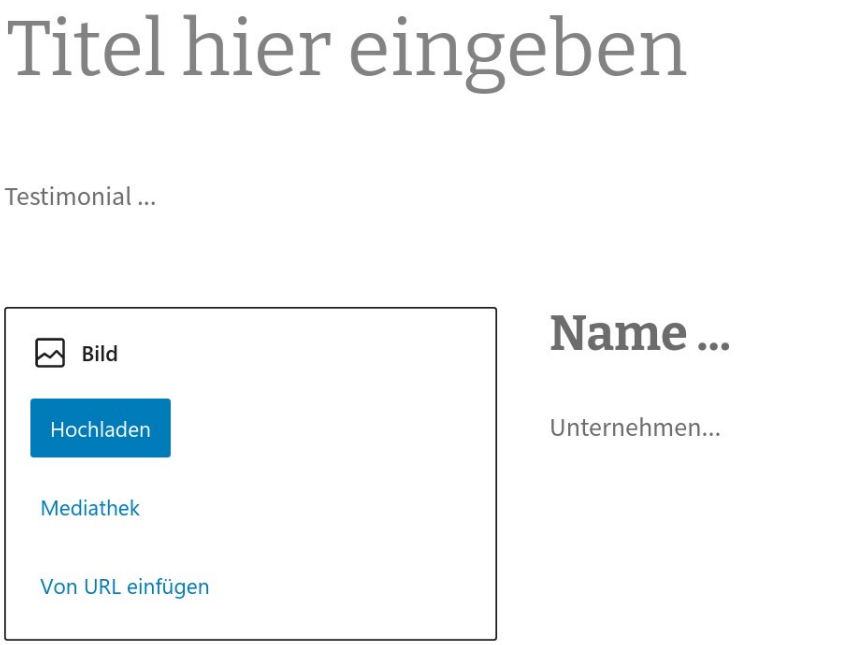
```php
<?php

/**
 * Registers the `testimonial` post type.
 */
function testimonial_init() {
	register_post_type(
		'testimonial',
		array(
			'labels'       => array(
				'name'          => __( 'Testimonials', 'testimonial-cpt' ),
				'singular_name' => __( 'Testimonial', 'testimonial-cpt' ),
			),
			'public'       => true,
			'supports'     => array( 'title', 'editor' ),
			'has_archive'  => true,
			'show_in_rest' => true,
			'template'     => array(
				array(
					'core/paragraph',
					array(
						'placeholder' => __( 'Testimonial ...', 'testimonial-cpt' ),
					),
				),
				array(
					'core/columns',
					array(),
					array(
						array(
							'core/column',
							array(),
							array(
								array(
									'core/image'
								),
							)
						),
						array(
							'core/column',
							array(),
							array(
								array(
									'core/heading',
									array(
										'level' => 2,
										'placeholder' =>  __( 'Name ...', 'testimonial-cpt' ),
									)
								),
								array(
									'core/paragraph',
									array(
										'placeholder' =>  __( 'Company ...', 'testimonial-cpt' ),
									)
								),
							)
						),
					)
				),
			),
		)
	);
}
add_action( 'init', 'testimonial_init' );
```

Zum Code →

Das Codebeispiel zeigt ein Template, das durch einen Absatz eingeleitet wird. Darunter befindet sich ein Spaltenblock mit zwei Spalten. In der ersten Spalte ist ein Bild-Block eingefügt und in der zweiten eine Überschrift zweiter Ordnung gefolgt von einem Absatz.

Die Blöcke können hierbei Attribute übergeben bekommen. Im Beispiel erhalten die Absätze und die Überschrift einen Platzhaltertext, der allerdings nicht angezeigt werden würde, wenn er nicht ersetzt wird.

Wird nun ein neuer Eintrag in dem Post-Type erstellt, dann ergibt sich folgendes Bild im Editor:



Da im Template kein Bildpfad in den Attributen angegeben wurde, erscheint der Bild-Block im „Bearbeitungsmodus“. Es kann also direkt ein Bild hochgeladen oder aus der Mediathek ausgewählt werden.

Durch den Einsatz eines solchen Templates kann also die Erstellung neuer Einträge vereinfacht und standardisiert werden. Dennoch wäre es aber möglich, dass das Layout nachträglich verändert wird. Dafür bestehen zwei Möglichkeiten:

Template sperren

Die erste Möglichkeit ist die Sperrung des Templates. Hierbei kann eingeschränkt werden, ob weitere Blöcke hinzugefügt werden können, oder aber, ob alle Blöcke gesperrt sind. Bei der Registrierung kann das gesamte Template gesperrt werden, indem der Parameter **template_lock** auf **all** gesetzt wird. Soll nur das Hinzufügen von zusätzlichen Blöcken verhindert werden, dann wird stattdessen der Wert **insert** verwendet. Es gibt aktuell keine Möglichkeit nur das Entfernen von Blöcken einzuschränken.

Verfügbare Blöcke einschränken

Die zweite Möglichkeit zur Begrenzung der Inhaltsstruktur ist die Einschränkung von verfügbaren Blöcken. Gerade bei speziellen Inhaltstypen, wie einem Testimonial aus diesem Artikel, sollte die Auswahl reduziert werden. Es macht vermutlich wenig Sinn, einen Widget-Block oder einen Embed zu verwenden.

Ausnahmsweise ist es nicht möglich, diese Einschränkung bei der Registrierung des Post-Types festzulegen. Stattdessen kann die Auswahl an Blöcken global über den Filter **allowed_block_types** gesteuert werden:

```php
<?php

function testimonial_cpt_allowed_blocks( $allowed_block_types, $post ) {
	if ( 'testimonial' === $post->post_type ) {
		return array(
			'core/heading',
			'core/paragraph',
			'core/list',
			'core/quote',
			'core/image',
			'core/video',
		);
	}

	return $allowed_block_types;
}
add_action( 'allowed_block_types', 'testimonial_cpt_allowed_blocks', 10, 2 );
```

Zum Code →

Da es sich um einen globalen Filter handelt und nur für den eigenen Post-Type die Auswahl eingeschränkt werden soll, muss geprüft werden, ob gerade dieser Post-Type bearbeitet wird. Ist dies der Fall, wird ein Array mit nur einer kleinen Auswahl an Blöcken zurückgeliefert. Ansonsten wird die ursprüngliche Auswahl zurückgegeben.

Es wäre hier auch denkbar, nur einzelne Blöcke aus dem Array zu entfernen. Da allerdings viele Plugins und Themes mittlerweile eigenen Blöcke mitbringen und nur einige davon im JavaScript-Code registriert werden, gibt es leider keine einfache Möglichkeit, diese im PHP rauszufiltern. Der Code müsste also ständig um neue Blöcke, die herausgefiltert werden sollen, erweitert werden. Daher ist es ratsam, eine Positivliste zu definieren.

Definition von Styles für den Post-Type

Wird im Post-Type eine spezielle Struktur verwendet, dann kann es sinnvoll sein, hierfür Styles zu definieren, die ein grundlegendes Design festlegen. Auch wenn es die Aufgabe von Themes ist, die Styles zu definieren, so sollte doch zumindest die Ausgabe ohne Styles aus dem Theme ansprechend aussehen.

Für das Einbinden von CSS und JavaScript Dateien sind mit dem Block-Editor ein paar neue Actions hinzugekommen. Prinzipiell ist es aber auch möglich, mit der gewöhnlichen Action Styles zu laden:

```php
<?php

function testimonial_cpt_frontend_scripts() {
        wp_enqueue_style(
                'testimonial-cpt-style',
                plugins_url( 'style.css', __FILE__ ),
                array(),
                filemtime( plugin_dir_path( __FILE__ ) . 'style.css' )
        );
}
add_action( 'wp_enqueue_scripts', 'testimonial_cpt_frontend_scripts' );
```

Zum Code →

Mit dieser Funktion werden die Styles lediglich im Frontend geladen. Bereits vor der Einführung des Block-Editors gab es die Action **admin_enqueue_scripts**, die Skripte und Styles im Dashboard geladen hat. Für Themes gab es außerdem die Funktion **add_editor_style()**, mit der eine Datei in den alten TinyMCE-Editor geladen werden konnte. Für den Block-Editor gibt es nun aber zwei neue Actions. Sollen Styles nur im

Block-Editor im Backend geladen werden, kann diese Action verwendet werden:

```php
<?php

function testimonial_cpt_editor_scripts() {
        wp_enqueue_style(
                'testimonial-cpt-editor',
                plugins_url( 'editor.css', __FILE__ ),
                array(),
                filemtime( plugin_dir_path( __FILE__ ) . 'editor.css' )
        );
}
add_action( 'enqueue_block_editor_assets', 'testimonial_cpt_editor_scripts' );
```

Zum Code →

Werden im Plugin Styles definiert, die sowohl im Frontend als auch im Backend benötigt werden, etwa für die Gestaltung von eigenen Blöcken, dann können diese in nur einer Action in beiden Bereichen geladen werden. In diesem Fall wird folgende Action verwendet:

```php
<?php

function testimonial_cpt_blocks_scripts() {
        wp_enqueue_style(
                'testimonial-cpt-blocks',
                plugins_url( 'blocks.css', __FILE__ ),
                array(),
                filemtime( plugin_dir_path( __FILE__ ) . 'blocks.css' )
        );
}
add_action( 'enqueue_block_assets', 'testimonial_cpt_blocks_scripts' );
```

Zum Code →

Alternativ ist es natürlich auch möglich, nur die letzte Action zu verwenden und innerhalb der Funktion mit der Conditional-Function **is_admin()** zu prüfen, welcher Bereich aufgerufen wird und abhängig davon dann die notwendigen Styles zu laden. In den meisten Plugins findet man aber mindestens die letzten beiden Actions.

Fazit

In diesem Artikel wurde demonstriert, wie ein Plugin entwickelt wird, das einen Custom-Post-Type registriert und diesen für den neuen Block-Editor vorbereitet. Dabei wurden ein paar neue Parameter und Funktionen vorgestellt, die mit dem Block-Editor hinzugekommen sind. Gerade die Verwendung der REST-API für den Block-Editor ermöglicht eine Vielzahl von interessanten neuen Anwendungsfällen. Der Einsatz von Block-Templates oder sogar eigenen Blöcken kann zudem die Arbeit mit den Inhalten im Vergleich zum alten Editor und den darin oft verwendeten Shortcodes stark verbessern.

Die Ergebnisse dieses Artikels können bei GitHub als fertiges Plugin heruntergeladen werden. Darin sind alle Code-Beispiele enthalten sowie – im Falle der Registrierung des Custom-Post-Types und der Custom-Taxonomy – die vollständigen durch die WP-CLI generierten Dateien.

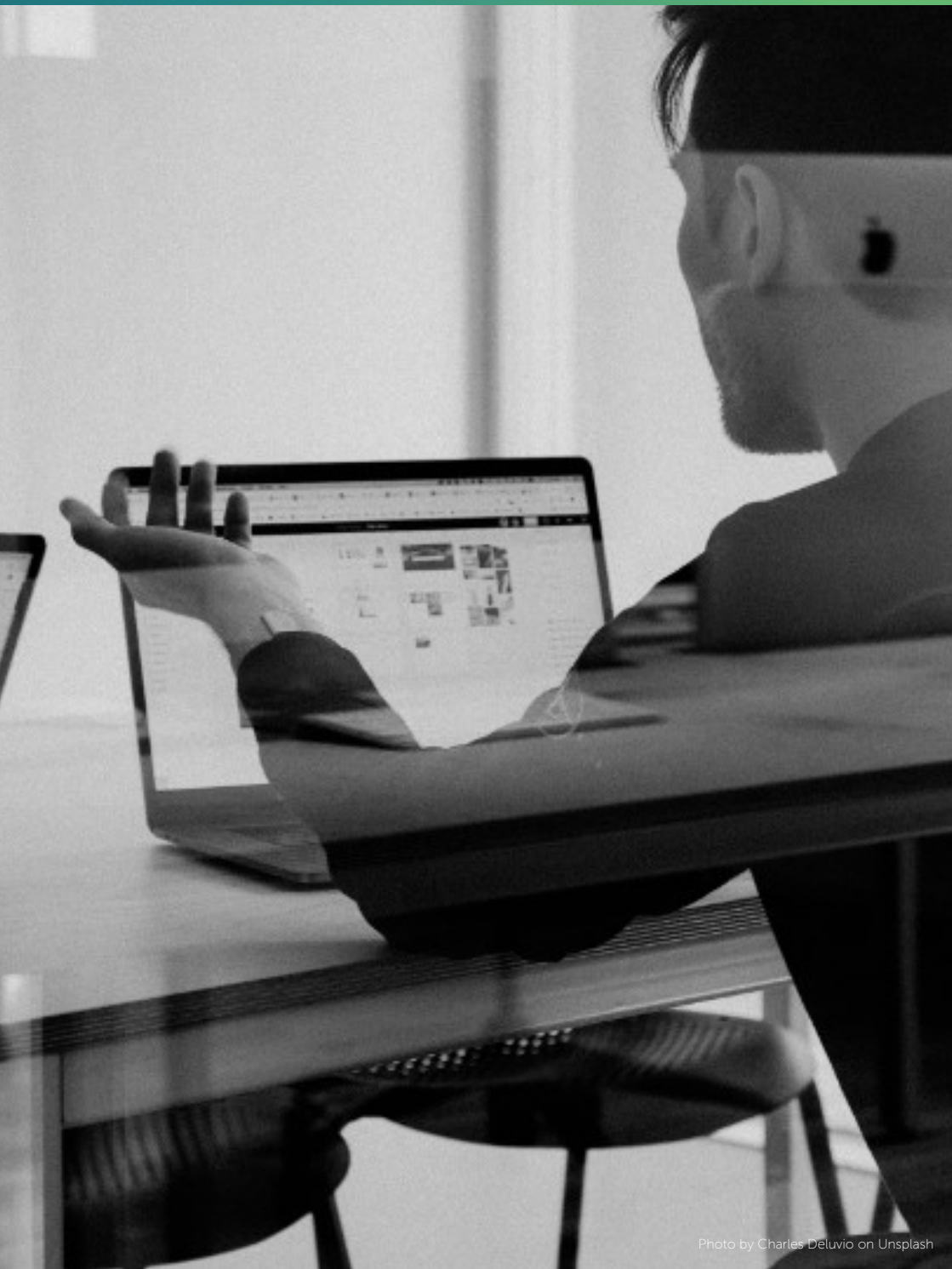


Photo by Charles Deluvio on Unsplash

Der Autor

Simon Kraft

Simon Kraft ist seit 2008 WordPress-Entwickler. Er ist ein aktives Mitglied der WordPress-Community und im Pluginkollektiv. Als Speaker und Berater beschäftigt er sich mit Performance-Optimierung und den Auswirkungen des Internets auf den Klimawandel.

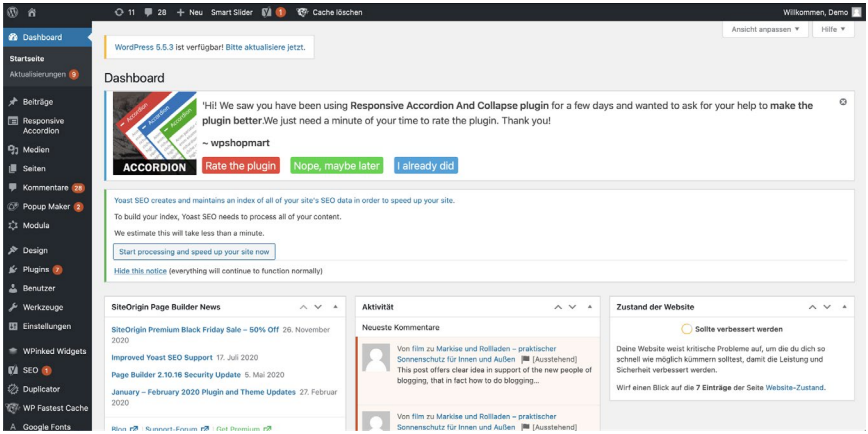
WordPress-Support als Must-Have

Die Idee von wiederkehrenden Dienstleistungen für Kund*innen, die eine neue Webseite ihr Eigen nennen, ist für Agentur-Inhaber*innen nicht neu. Dennoch wird hier viel zu häufig eine Chance verpasst, nicht nur eine dauerhaft stabile und nachhaltige Beziehung zu Kunden zu etablieren, sondern auch verlässliche Einnahmen zu generieren.

Woraus Sie Ihre neuen Service-Tarife aufbauen können, welche Werkzeuge und Dienste Sie nutzen können, um WordPress-Service effektiv anzubieten und wie Sie Ihre Kunden vom Mehrwert des Ganzen überzeugen, wollen wir uns im Folgenden genauer ansehen.

Warum WordPress-Service und -Wartung für Ihre Kunden wichtig ist

Nehmen wir für einen Moment an, Sie haben gerade eine neue Webseite fertiggestellt. Nach Wochen der Vorbereitung, Konzeption, Gestaltung und Umsetzung sind nicht nur Sie mit Ihrer Arbeit zufrieden, auch Ihr Kunde ist glücklich. Doch ganz egal, wie elegant der Code des Themes oder wie gewissenhaft die Auswahl der eingesetzten Plugins auch sein mag, ab dem Zeitpunkt der Fertigstellung nagt der Zahn der Zeit an einer Webseite. Und ohne regelmäßige Aufmerksamkeit werden sich kleine Plugin-Updates mit der Zeit ansammeln und von größeren WordPress-Core-Updates Gesellschaft bekommen. Was zunächst einfach ignoriert werden kann, wird spätestens dann kritisch, wenn sich unter den auflaufenden Updates erste Security-Fixes tummeln. Kleine Nachlässigkeiten summieren sich auf und werden nach einer Weile zu echten Problemen.



Ohne gelegentliche Aufmerksamkeit sammeln sich in einem WordPress-Backend schnell ausstehende Aktualisierungen, ungelesene Nachrichten und Spam.

Im schlimmsten Fall führt das dazu, dass die Webseite gar nicht mehr erreichbar ist und die Investition Ihres Kunden verloren ist. Hier verhält es sich ganz ähnlich wie mit einem Neuwagen. Ab dem Moment, in dem er vom Band rollt, bedarf er regelmäßiger Pflege. Ölstand und Wischerflüssigkeit wollen kontrolliert werden, kleine Mängel sollten beseitigt werden, bevor sie zu größeren Problemen führen und die Fahrtüchtigkeit des Wagens beeinflussen. Genau wie bei einer Webseite führt ein einzelnes Problem in der Regel noch nicht zu einem Totalausfall. Aber spätestens, wenn Sie irgendwo liegenbleiben werden Sie sich wünschen, wegen dieses komischen Geräusches in die Werkstatt gegangen zu sein. Doch während es vielen von uns einleuchtet, dass es wirtschaftlich sinnvoller ist, unsere Autos regelmäßig warten zu lassen, statt sie alle zwei Jahre zu verschrotten, bedarf die Übersetzung des Ganzen in den digitalen Raum noch immer einer gewissen Hilfestellung.

Ein regelmäßiges Einkommen mit Wartung und Support

Doch neben den ganz unbestreitbaren Vorteilen, den regelmäßige Updates und Wartung für Ihre Kunden bringen, liegen auch die Vorteile für Sie als Anbieter auf der Hand: regelmäßiges und planbares Einkommen. Ich habe lange genug für und in Agenturen gearbeitet, um zu wissen, dass fast jede Auftragslage, egal wie gut sie sein mag, ein gewisses Maß an Unsicherheit birgt. Hier zusätzlich zu Einzelaufträgen verschiedener Größe auf einen Grundstock an regelmäßigen Einnahmen zurückgreifen zu können, sorgt für etwas mehr Planungssicherheit.

Das Schöne an WordPress ist außerdem, dass es mit den richtigen Werkzeugen und Methoden der perfekte Unterbau für langlebige Webseiten sein kann. Das Verwalten von Usern und Inhalten ist nach kurzer Einweisung einfach und dank eines riesigen Ökosystems an hochwertigen Erweiterungen und Services können Sie ohne große Investition schnell in Ihr Wartungsbusiness einsteigen.

Was gehört in einen attraktiven Wartungsvertrag?

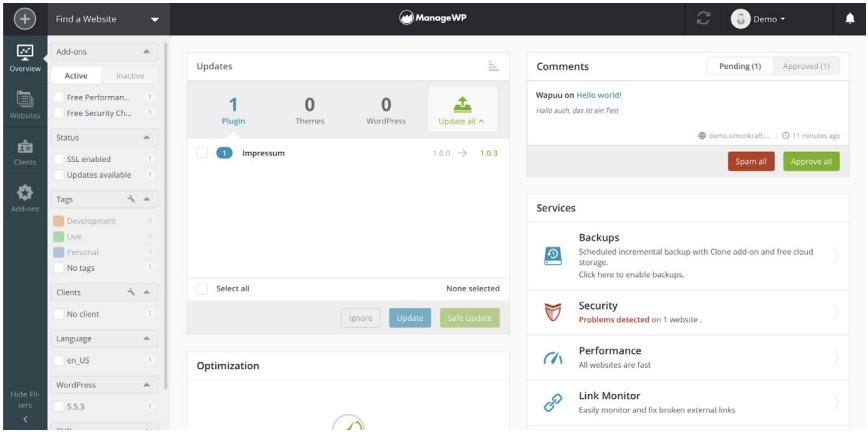
Wenn Sie sich mit dem Schnüren von Service-Paketen beschäftigen, werden Sie schnell feststellen, dass ein „One-size-fits-all"-Ansatz hier nur schwer funktioniert. Kleine Webseiten kommen mit regelmäßigen Updates und monatlichen Backups aus. Andere Kunden brauchen eine engmaschigere Betreuung, häufigere Updates oder sogar redaktionelle Hilfe. Welche Dienstleistungen Sie für Ihre Kunden bündeln möchten, liegt natürlich ganz in Ihrem eigenen Ermessen, einige Grundlagen sollten aber abgedeckt sein.

Updates und Backups

Regelmäßige Updates von WordPress-Core, Plugins und Themes sollten in jedem Fall zu Ihrem Service-Umfang gehören. Auch wenn WordPress selbst seit Sommer 2020 die Möglichkeit für automatische Plugin- und

Theme-Updates bietet, gibt es hier eine Menge zu tun. Im Optimalfall koppeln Sie Updates mit einem vorgeschalteten Backup, sodass sie bei etwaigen Problemen auf die vorherige Version zurückrollen können. Um Updates wirklich sicher testen zu können, lohnt sich außerdem eine eigene Testinstallation mit dem Setup des Kunden.

Backups von Datenbank und Dateisystem der Webseite sollten möglichst automatisch in einem angemessenen Zeitraum angefertigt werden. Dieser Zeitraum richtet sich ganz nach dem jeweiligen Anwendungsfall. Eine News-Seite mit dutzenden von neuen Beiträgen pro Tag sollte sicher häufiger gesichert werden, als eine relativ statische Webseite, die nur sporadische Änderungen erfährt.



Im ManageWP-Dashboard finden Sie alle Informationen zu beliebig vielen WordPress-Webseiten auf einen Blick.

Weder Updates noch Backups müssen Sie manuell anfertigen. Lösungen wie ManageWP oder InfiniteWP geben Ihnen die Möglichkeit, diese Aufgaben nicht nur für mehrere Webseiten an einer zentralen Stelle

durchzuführen, sondern auch automatisch regelmäßig ablaufen lassen. Alternativ können Sie auch mit dem WordPress-Kommandozeilen-Tool WP CLI eigene Scripte bauen, die sich um die entsprechenden Aufgaben kümmern.

Support

Ob Sie Support-Anfragen zu einem Pauschalpreis anbieten, nach Stunden abrechnen oder ein Inklusiv-Kontingent pro Monat anbieten, ist ganz Ihnen überlassen. Wichtig ist nur, dass Sie Ihren Kunden eine einfache Möglichkeit geben, sich mit Fragen und Problemen an Sie zu wenden. Für den Anfang sind Sie hier mit einem einfachen E-Mail-Postfach gut bedient. Kümmern Sie sich um mehrere Kunden oder arbeiten in einem größeren Team, kommt einfache E-Mail schnell an ihre Grenzen. Moderne Ticketsysteme wie HelpScout oder Zammad können Ihnen hier viel Arbeit abnehmen und die Kommunikation mit Ihren Kunden in geregelte Bahnen lenken. Für Kunden, die eine persönlichere Betreuung erfordern, können Sie natürlich auch telefonisch Support leisten.

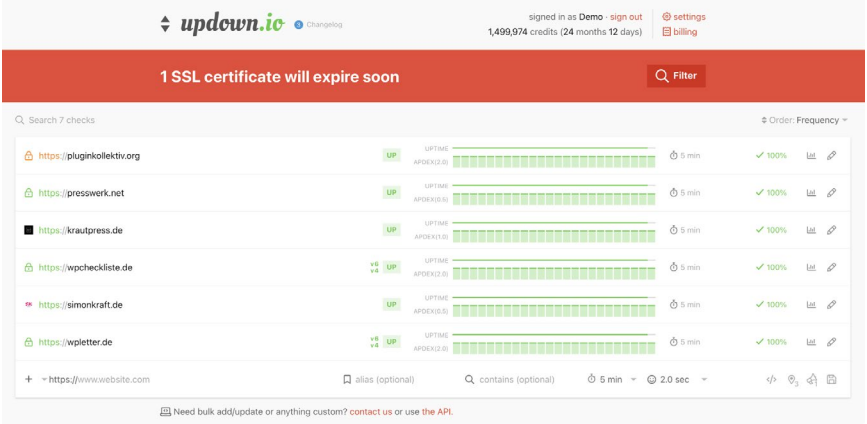
In allen Fällen sollten Sie hier transparent sein. Teilen Sie klar mit, was Ihre Reaktionszeiten für Kommunikation und Fehlerbehebungen sind und wie Sie mit Anfragen und Problemen außerhalb der üblichen Bürozeiten umgehen. Während vor allem größere Kunden gern ein Service-Level-Agreement in Händen halten, können Sie für kleinere Kunden durchaus auch informellere Kommunikationsregeln aufstellen.

Monitoring

Webseiten können aus den verschiedensten Gründen nicht mehr erreichbar sein. Die meisten Probleme lassen sich schnell beheben, vorausgesetzt Sie wissen davon. Wir wollen vermeiden, dass Sie von Ihrem Kunden auf Probleme aufmerksam gemacht werden müssen, oder noch schlimmer, dass Besucher der Webseite längere Zeit nicht auf diese zugreifen

können. Website-Monitoring löst dieses Problem recht elegant. Je nach eingesetztem Service werden Sie per E-Mail, Slack-Nachricht oder SMS über Probleme informiert und können umgehend aktiv werden.

Auch hier sind die beiden zuvor genannten Tools, ManageWP und Infinite-WP, hilfreich. Aber auch gesonderte Dienste wie updown.io oder selbst betriebenen Software wie Nagios kann hier hilfreich sein.



updown.io hat Ihre Webseiten im Blick und informiert Sie bei Bedarf über Probleme.

Sie möchten mit dem Monitoring noch etwas tiefer gehen und nicht nur offensichtliche Probleme in der Website-Erreichbarkeit adressieren? Mit Sentry, können Sie nicht nur PHP-Fehler mitschneiden, sondern sogar auftretende JavaScript-Probleme protokollieren. Auch Sentry kann als SaaS genutzt oder selbst betrieben werden.

Redaktionelle Unterstützung und SEO

Hier kommt es ganz auf Ihre persönlichen Fähigkeiten oder die Ihres Teams an. Von fertigen Redaktionsplänen mit größeren Mengen von

Inhaltserstellung bis hin zu gelegentlichen Änderungen auf Zuruf – Ihrer Kreativität sind kaum Grenzen gesetzt. Laufende Optimierung und Überwachung von Suchmaschinen-Rankings der Webseite können ebenfalls ein wertvoller Bestandteil des Wartungsplans sein.

Nützliche Zusatzdienste und Lizenzen

Viele Anbieter inkludieren in Wartungsplänen Lizenzen für verschiedene Bezahl-Plugins oder -Themes. Solange Kund*innen einen laufenden Wartungsvertrag haben, können sie zum Beispiel auf die Volumenlizenz der Agentur für Caching-Plugins oder einen beliebten Page-Builder zurückgreifen.

Auch wenn Sie Ihr Angebot so sicherlich um den einen oder anderen sinnvollen Zusatznutzen ergänzen können, sollten Sie darauf achten, dass sich Ihre Kunden über die Details dieses Arrangements im Klaren sind. Machen Sie deutlich, welche Funktionen sie nicht mehr werden nutzen können, wenn sie den Wartungsvertrag kündigen, wie viel die Einzellizenz kosten würde und woher sie die entsprechenden Lizenzen bekommen können.

Regelmäßige Reports

Um Ihre Arbeit ein Stück weit zu dokumentieren und Ihren Kunden etwas klarer zu machen, dass sie für Ihr Geld tatsächlich arbeiten, bieten sich regelmäßige Reports an.

Hier können Sie auflisten, wann zuletzt Updates und Backups gemacht wurden, Sie können Statistiken aus dem Monitoring einbinden, Veränderungen in SEO-Rankings und Performance einflechten und auf bearbeitete Support-Anfragen eingehen.

Der Umfang dieses Dokuments variiert natürlich je nach Umfang des Wartungsplans. Vor allem bei umfangreicheren Services haben Sie aber die Gelegenheit, Ihre Reports zu wirklich informativen Dokumenten zu

machen. Somit erleichtern Sie Ihren Kunden, die Webseite produktiv zu nutzen und weiterzuentwickeln, und lassen Sie Ihre Kunden ganz nebenbei wissen, dass sich ihre Investition in Ihre Dienste lohnt.

Alles Weitere nach Augenmaß

Für die weitere Ausgestaltung Ihres Angebots können Sie auf Erfahrungen zurückgreifen, die Sie mit Ihren Kunden in der Vergangenheit gesammelt haben. Wo sind die meisten Rückfragen aufgetreten? Welche Probleme waren für Ihre Kunden am frustrierendsten? Hier können Sie ansetzen, um Pakete zu schnüren, die für Ihren Kundenstamm optimal sind.

Nehmen Sie bei Angeboten für solche Pläne Rücksicht auf die Umstände des Kunden und fügen Sie Ihrem jeweiligen Angebot Dienstleistungen nur dann hinzu, wenn sie für den aktuellen Kunden einen echten Mehrwert bieten.

Sie sind nicht sicher, wie Sie einen Kunden von der Sinnhaftigkeit eines Wartungsvertrags überzeugen können? Bieten Sie das erste Jahr als Posten im Rahmen eines Webseiten-Launches an und lassen Sie die Qualität Ihrer Arbeit für Sie sprechen. Kostenlos anbieten sollen Sie die Wartung natürlich dennoch nicht.

Automatisierung und Rationalisierung

Die Liste möglicher Dienstleistungen können Sie fast endlos erweitern. Ausgehend davon, dass Sie nicht planen, Ihr gesamtes Geschäft auf Pflege und Wartung zu verlegen und auch kein eigenes Team für diesen Aufgabenbereich abstellen wollen, sind jedoch Abstriche nötig. Für den Anfang sollten Sie sich auf Dienstleistungen konzentrieren, die sich einfach (teil-) automatisieren lassen. Je mehr regelmäßige Aufgaben Sie nicht manuell erledigen müssen, desto mehr Zeit können Sie Neukunden, Support-Anfragen oder weiteren Verbesserungen widmen.

Für Aufgaben, die Sie nicht automatisieren können oder wollen – etwa, weil sie zu selten vorkommen – lohnt es sich meistens, klare Arbeitsabläufe zu definieren. Dokumentationen solcher Prozesse helfen Ihnen selbst beim nächsten Durchlauf und können als eine Art Checkliste fungieren, sind aber auch dann hilfreich, wenn Sie sich die Aufgaben mit einer (oder mehreren) Person(en) teilen und auf diesem Wege eine gleichbleibende Qualität sicherstellen können.

Das sind Ihre nächsten Schritte
So gelingt Ihnen der Einstieg in die WordPress-Wartung:

1. Verschaffen Sie sich einen Überblick über die verschiedenen Tools und Dienste, die Sie einsetzen können. Testen Sie sie mit eigenen Webseiten, und entscheiden Sie, welche Tools am besten in Ihren Workflow passen. Ein einheitliches Set-up, das für möglichst viele Kunden zur Anwendung kommen kann, spart Ihnen später viel Arbeit.

2. Sprechen Sie mit Ihren Kunden über Ihre Ideen, fragen Sie Bedürfnisse ab und bleiben Sie in regelmäßigem Kontakt über die weiteren Entwicklungen.

3. Schnüren Sie ein Basis-Paket für Wartung und Service. Erweitern Sie es individuell nach Kundenbedürfnissen und passen Sie zum Beispiel Update- und Backup-Intervalle an.

4. Versuchen Sie einen ersten Wartungsvertrag unters Volk zu bringen. Haben Sie einen besonders pflegeleichten Lieblingskunden? Hier können Sie ansetzen und einen ersten Test starten. Ob ein Neukunde oder ein Bestandskunde den Anfang macht, liegt dabei natürlich in Ihrem eigenen Ermessen. Meine Empfehlung wäre allerdings, das geplante Setup mit einem einzelnen Kunden zu testen und erst nach und nach für mehrere Kunden anzubieten.

5. Bedenken Sie, dass Sie nicht zwingend alle Teile der Dienstleistung selbst erbringen müssen. Greifen Sie auf Ressourcen in Ihrem Netzwerk zurück, um Aufgaben wie SEO, Content oder Entwicklung an entsprechende Expert*innen auslagern zu können.

6. Stecken Sie klare Parameter ab, innerhalb derer Sie arbeiten möchten. Die Wartung und Pflege einer Webseite, die Sie nicht selbst erstellt haben, kann unerwartete Fallstricke mit sich bringen. WordPress-Installationen, die auf ungewöhnliche Hosting-Setups setzen, können gewohnte Arbeitsabläufe stören. Das alles sind natürlich keine zwingenden Ausschlusskriterien. Bestehende Webseiten können vor Vertragsabschluss überprüft und Hoster können gewechselt werden. Aber seien Sie sich von Anfang an klar, ob Sie bereit sind, solche Aufgaben zu übernehmen, oder ob es nicht lohnender ist, Pflege und Wartung nur für Webseiten anzubieten, die Sie selbst erstellt haben.

Auch wenn Sie Ihr Angebot in Form gebracht und erste glückliche Kunden davon überzeugt haben, lohnt es sich, weiter zu experimentieren und zu optimieren. Bleiben Sie auf dem neuesten technischen Stand, informieren Sie sich gelegentlich über neue Tools und Services, die Sie nutzen können, und vergessen Sie nie, im Dialog mit Ihren Kunden zu bleiben, um auch auf deren veränderte Bedürfnisse weiter eingehen zu können.

Host Europe

Sie haben weitere Fragen?

Unser Sales Team ist gerne für Sie da.

0800 626 4624

www.hosteurope.de