Hochschule Hannover

Fakultät III – Medien, Information und Design

Abteilung Information und Kommunikation

Entwicklung eines WordPress-Themes mit Zielgruppe Fotografen

Bachelorarbeit

im Studiengang Informationsmanagement

vorgelegt von

Florian Brinkmann

Erstprüfer: Prof. Dr. Thomas Schult

Zweitprüfer: B. A. Andre Kreutzmann

Hannover, den 22. Januar 2016

Abstract

Die vorliegende Arbeit befasst sich mit der Entwicklung eines WordPress-Themes mit der Zielgruppe Fotografen unter besonderer Berücksichtigung der Richtlinien, die für das offizielle Theme-Verzeichnis erfüllt werden müssen. Dabei werden zuerst die Anforderungen an das Theme ermittelt – sowohl durch die Zielgruppe als auch durch das Theme-Verzeichnis. Anschließend beschäftigt sich die Arbeit mit der Untersuchung bereits bestehender Themes auf diese Funktionen hin, wobei keins der untersuchten alle Funktionen vorweist.

Der größte Teil der Arbeit widmet sich der konkreten Programmierung des Themes, wobei der Code detailliert erläutert wird. Abschließend wird das Theme für die kostenlose Bereitstellung in das *WordPress Theme Directory* hochgeladen und der damit einhergehende Review-Prozess sowie die notwendigen Korrekturen behandelt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listing-Verzeichnis

1 Einleitung

Ausgangssituation

Die vorliegende Arbeit setzt sich mit der Entwicklung eines Themes für das Content-Management-System WordPress auseinander, das die Zielgruppe Fotografen ansprechen soll. Nach Fertigstellung soll das Theme in das offizielle Theme-Verzeichnis aufgenommen werden und kostenlos zur Verfügung stehen. Besondere Aufmerksamkeit wird auf die Aspekte gelegt, die für eine Aufnahme in das Verzeichnis erfüllt werden müssen.

Forschungsstand

Da das Thema im Bereich der Webentwicklung angesiedelt ist, lässt sich kaum aktuelle Printliteratur finden. Diese Situation ist der Tatsache geschuldet, dass der Fortschritt in den Bereichen Webentwicklung und WordPress zu schnell ist, um den aktuellen Stand in einem Buch festzuhalten. Besser geeignet sind Online-Dokumentationen, die das jeweilige Thema umfangreich und aktuell beleuchten können.

Für die Funktionen, Klassen und Hooks von WordPress liegt eine umfangreiche Referenz vor.[1] Die Voraussetzungen, die ein Theme für das Theme-Verzeichnis erfüllen muss, sind im Handbuch des Theme-Review-Teams zu finden.[2] Für CSS, HTML und JavaScript können viele Lösungen und Erläuterungen in den jeweiligen Dokumentationen von Mozilla gefunden werden.[3] Wenn der Einsatz von PHP-Funktionen, -Klassen oder ähnlichem notwendig wird, ist die erste Anlaufstelle das PHP-Handbuch.[4]

Fragestellung und Zielsetzung

Die Zielsetzung der Bachelorarbeit besteht in der Entwicklung eines WordPress-Themes für Fotografen, das in das offizielle Theme-Verzeichnis hochgeladen wird und somit kostenlos zur Verfügung steht. Damit können Fotografen, Foto-Blogger und andere Nutzer sich und ihre Arbeit bestmöglich im Web präsentieren. Darüber hinaus wird das Ergebnis der Arbeit als Referenz für meine spätere Tätigkeit in diesem Bereich dienen.

Als Fragestellung wurde im Studienabschlussseminar formuliert: „Welche Funktionen sollte ein kostenloses WordPress-Theme für Fotografen auf jeden Fall bieten?“ Diese Frage

[1] vgl. WordPress: Code Reference
[2] vgl. WordPress: Theme Review Handbook
[3] vgl. Mozilla: Webtechnologien für Entwickler | MDN
[4] vgl. PHP Group: PHP-Handbuch

muss in jedem Fall beantwortet werden um zu begründen, warum bestimmte Funktionen in das WordPress-Theme integriert werden.

Im Gespräch mit dem Zweitbetreuer der Arbeit, Herrn Kreutzmann, wurde eine weitere Frage aufgeworfen, mit der sich die Arbeit detailliert auseinandersetzen sollte: „Welche Voraussetzungen müssen erfüllt werden, damit ein WordPress-Theme mit Nutzeranforderung in das Theme-Verzeichnis aufgenommen wird?“. Mit dieser Frage im Hinterkopf wird sich die Arbeit genau mit den Anforderungen und Richtlinien beschäftigen, die ein Theme für die Aufnahme in das Verzeichnis erfüllen muss.

Methodische Vorüberlegungen

Die Bachelorarbeit hat einen hohen Praxisbezug, sie beschäftigt sich zum größten Teil detailliert mit der Programmierung des Themes. Davor steht die Beantwortung der Frage nach den Funktionen, die Fotografen benötigen, sowie einer Analyse bestehender WordPress-Themes bezüglich dieser Funktionen.

Um die Frage zu beantworten, welche Funktionen ein WordPress-Theme für Fotografen erfüllen muss, wird eine Analyse von Fotografen-Seiten durchgeführt. Ein Experten-Interview, wie im Studienabschlussseminar als Lösung festgehalten, hat sich als nicht besonders geeignet herausgestellt. Zu diesem Eindruck haben die – natürlich – sehr subjektiven Antworten des befragten Experten geführt. Ein besserer Weg dürfte hier die Analyse von praktischen Beispielen sein.

Damit nicht zufällig irgendwelche Fotografen-Seiten ausgewählt werden, sind die Websites der Preisträger von dem Photo Contest 2015 der World Press Photo Foundation Gegenstand dieser Analyse.[5] Auch wenn nicht alle Preisträger eine Website betreiben, verbleiben 28 für die Analyse.

Nach der Analyse müssen die bereits vorhandenen kostenlosen Themes untersucht werden um auszuschließen, dass es ein solches Theme bereits gibt. Dabei wird das Angebot des offiziellen Theme-Verzeichnissess als Untersuchungsobjekt verwendet.[6] Zudem werden kostenpflichtige Themes begutachtet – auch wenn diese Themes keine direkte Konkurrenz darstellen.

[5] vgl. World Press Photo Foundation: 2015 Photo Contest
[6] vgl. WordPress: Theme-Verzeichnis

Aufbau der Arbeit

Zunächst behandelt die Arbeit die verschiedenen Anforderungen an das Theme, die sowohl aus der Analyse der bestehenden Websites als auch aus den Richtlinien und Regeln des Theme-Verzeichnisses hervorgehen. Darauf folgen die Analyse der bereits vorhandenen kostenlosen und einiger kostenpflichtigen Themes aus dem Bereich sowie die Vorbereitung auf die Entwicklung mit nützlichen Einstellungen, Plugins und Test-Daten. Im Anschluss befasst sich der größte Teil der Arbeit mit der Entwicklung des Themes, gefolgt von der Einarbeitung von Review-Korrekturen.

2 Anforderungen an das WordPress-Theme

An das WordPress-Theme werden von verschiedenen Seiten Anforderungen gestellt. Auf der einen Seite sind das die der Zielgruppe, also der Fotografen, die bestimmte Funktionen erwarten. Auf der anderen Seite werden bestimmte Anforderungen an Themes gestellt, die in das Theme-Verzeichnis von WordPress.org aufgenommen werden sollen.

2.1 Funktionen

Um herauszufinden, welche Funktionen ein Fotograf in einem WordPress-Theme oder allgemein einer Website benötigt, bietet sich die Analyse von bestehenden Websites an. Hierzu wurden die Internetauftritte der Preisträger des Photo Contest 2015 untersucht, der von der World Press Photo Foundation veranstaltet wird.

Von den 41 verschiedenen Preisträgern konnten für 28 Websites ausfindig gemacht werden.[7] Aus der Untersuchung dieser Seiten, die der Arbeit im Anhang A beiliegt, ergeben sich die folgenden Funktionen, die das WordPress-Theme mitbringen sollte, um für eine möglichst große Zahl von Nutzern aus der Zielgruppe interessant zu sein.

2.1.1 Trennung von Portfolio-Elementen und Blogbeiträgen

Auf den Fotografen-Websites, die ein Blog beinhalten, sind die Portfolio-Elemente streng davon getrennt. Für das Erstellen von Portfolio-Elementen bieten sich in WordPress allerdings eher die Beiträge als die Seiten an, da diese auch kategorisiert werden können.

[7] vgl. World Press Photo Foundation: 2015 Photo Contest

Hier muss demnach eine Funktion integriert werden, die es dem Nutzer des Themes erlaubt, bestimmte Beiträge aus dem Blog auszuschließen. Das sollte aber nicht die Standard-Lösung sein, damit die Portfolio-Elemente auch im Blog angezeigt werden können. Hier könnte mit einer Kategorie oder einem Tag gearbeitet werden, die der Nutzer für Beiträge, die nicht im Blog aufgeführt werden sollen, vergeben muss. Ebenfalls denkbar wäre eine Option, dass alle Beiträge vom Post-Format *Galerie* und *Bild* aus dem Blog ausgeschlossen werden können.

2.1.2 Galerie als Slider oder einzelne Bilder anzeigen

Im Grunde gibt es zwei Varianten, wie mehrere Bilder einer Serie auf den untersuchten Websites angezeigt werden: Entweder in einem Slider oder als einzelne Bilder, die meist vergrößert in einer Lightbox angezeigt werden können. Die einzelnen Bilder werden als Thumbnails dargestellt, die nach einem Klick das große Bild in einer Lightbox anzeigen oder einfach in voller Größe untereinander.

Dem Nutzer müssen demnach diese zwei Möglichkeiten geboten werden. Um die größte Flexibilität zu erreichen, müsste diese Einstellung für jeden Beitrag möglich sein. Einfacher und realistischer ist eine generelle Einstellung, sodass alle Galerien entweder als Slider oder als einzelne Bilder dargestellt werden.

2.1.3 Slider oder zufälliges Bild auf der Startseite

Neben einer normalen Inhaltsseite als Startseite haben einige Websites aus der Untersuchung einen Slider eingebaut. Eine Seite hat nur ein Bild angezeigt, dieses aber bei einem erneuten Seitenaufruf zufällig geändert. Dieses Verhalten lässt sich mit zwei verschiedenen Page-Templates erreichen.

2.1.4 Übersicht aller Arbeiten auf einer Seite und Archiv

Auf mehreren Websites werden auf einer Seite alle Serien oder Galerien des Fotografen angezeigt. Das ist meist ein Bild, der Titel und eventuell beschreibender Text, der dann auf die Einzelansicht der Arbeit verlinkt. Auch dafür ist die Lösung der Wahl ein Page-Template.

Als weitere Funktion in diesem Zusammenhang ist mehrfach ein Archiv verwendet worden, wie in Abbildung 1 zu sehen ist. Der Fotograf muss also Galerien markieren können, sodass sie nicht mehr in dieser Übersicht, sondern separat in einem Archiv aufgeführt werden. Hier kann wieder – wie bei den Beiträgen, die nicht im Blog angezeigt werden sollen – mit einer Kategorie beziehungsweise einem Tag gearbeitet werden.

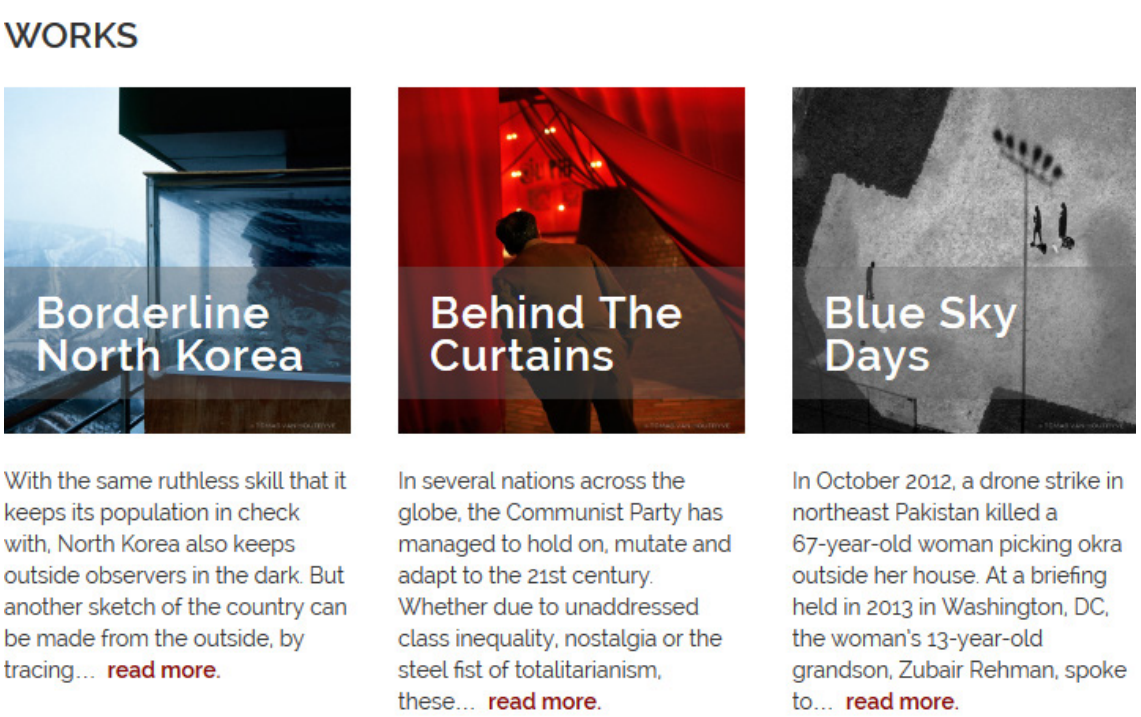


Abbildung 1: Übersicht der Arbeiten mit Archiv-Link auf der Website von Tomas van Houtryve
Quelle: Screenshot von tomasvh.com am 1. November 2015

2.1.5 Kategorie-Seiten für Arbeiten

Auf einer Website hat der Fotograf im Menü seine Arbeiten nach Kategorien verlinkt, damit der Besucher sich nur die aus einer Kategorie anzeigen lassen kann. Die Darstellung der Kategorie-Übersicht findet auf ähnliche Weise statt, wie die Übersicht aller Galerien.

Auch hier wird wieder ein Seiten-Template zum Einsatz kommen. Für jede Seite mit diesem Seiten-Template wird im Customizer, der in WordPress Anpassungen mit einer Live-

Vorschau ermöglicht, dann eine Option angezeigt, mit der die Kategorie festgelegt werden kann, aus der die Galerien angezeigt werden sollen.

2.1.6 Verschiedene Darstellungsmöglichkeiten für Portfolio- und Kategorie-Übersicht

Die Verweise auf die einzelnen Arbeiten werden auf den untersuchten Websites in verschiedenen Arten dargestellt. Ein Beispiel ist da die Website von Tomas van Houtryve aus Abbildung 1, die neben Titel und Bild noch einen Text anzeigt. In Abbildung 2 ist die Website von Sergei Ilnitsky zu sehen, auf der standardmäßig nur Thumbnails zu den Serien gezeigt werden. Der Titel wird beim Hovern mit der Maus eingeblendet, was auf Touchscreens allerdings nicht funktioniert und deshalb nicht optimal ist.

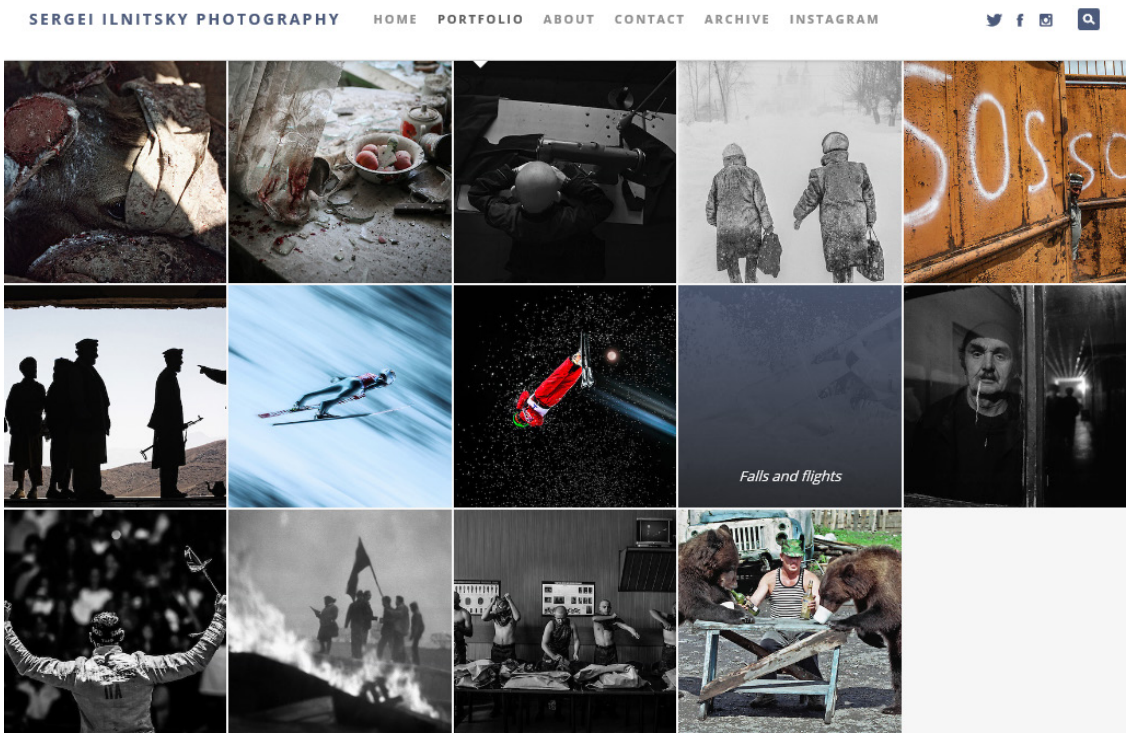


Abbildung 2: Übersicht des Portfolios von Sergei Ilnitsky.
Quelle: Screenshot von ilnitsky-photography.com am 1. November 2015

Diese zwei Varianten sollte der Nutzer des Themes mindestens auswählen können: Entweder werden die Verweise auf die Einzelansichten mit Titel, Bild und Beschreibung angezeigt, oder nur mit Thumbnail.

2.2 Regeln und Richtlinien des Theme-Verzeichnisses

Um in das Theme-Verzeichnis von WordPress.org aufgenommen zu werden, muss ein WordPress-Theme bestimmte Regeln erfüllen. Hier gibt es zwei Abstufungen: Einerseits gibt es verbindliche Regeln, deren Verletzung ein Ausschlusskriterium darstellt.[8] Andererseits gibt es noch sogenannte *Best-Practices*, deren Einhaltung empfohlen, bei deren Verletzung ein Theme aber nicht ausgeschlossen wird.[9]

2.2.1 Erforderliche Regeln

Code

Der Code eines WordPress-Themes darf keine PHP- oder JavaScript-Fehler auswerfen. Darüber hinaus muss ein Theme einen validen Doctype sowie das `lang`-Attribut enthalten. Zudem müssen Benutzereingaben bereinigt werden und es dürfen keine Hooks verändert oder entfernt werden, die nichts mit der Präsentation und damit den Aufgaben eines Themes zu tun haben.

Außerdem muss das Theme den Test mit dem Theme-Check-Plugin bestehen und seine Funktionen, globale Variablen, Konstanten, et cetera, die im globalen Namensraum definiert werden, mit einem einzigartigen Präfix kennzeichnen, damit es keine versehentlichen Namensgleichheiten mit anderen Funktionen gibt.[10]

Core-Funktionalität und Funktionen

Wenn vorhanden, müssen Themes zuerst Funktionalitäten aus dem WordPress-Core verwenden. Themes dürfen keine Hinweise im Backend einbinden, die auf eine Funktion des Themes verweisen und keine WordPress-eigenen Funktionen hinter einer Paywall verstecken oder die Admin-Toolbar deaktivieren. Darüber hinaus ist es Theme-Entwicklern nicht erlaubt, den Pfad zum Theme mit `TEMPLATEPATH` und den Pfad zum Stylesheet mit `STYLESHEETPATH` auszugeben. Hierfür gibt es die beiden Funktionen `get_template_directory()` sowie `get_stylesheet_directory()`.

Daneben sollten Änderungen am Inhalt nicht fest im Code verankert sein, sondern über Funktionsparameter, Filter- oder Action-Hooks umgesetzt werden, und es muss möglich

[8] vgl. WordPress: Theme Review Handbook › Required
[9] vgl. WordPress: Theme Review Handbook › Recommended
[10] vgl. WordPress: Theme Review Handbook › Required › Code

sein, für das Theme ein Child-Theme anzulegen. Das Kommentar-Template eines Themes muss über die `comments_template()`-Funktion eingebunden sein und Theme-Tags sowie die Beschreibung müssen zutreffend für Funktionen und Design des Themes gewählt beziehungsweise formuliert werden. Template-Tags sowie Action- und Filter-Hooks müssen im Theme korrekt eingesetzt werden.[11]

Präsentation versus Funktion

Dieser Abschnitt ist im Handbuch etwas knapp gehalten und fordert lediglich, dass Themes keine Nutzerinhalte generieren sowie keine Optionen oder Funktionen integrieren dürfen, die nicht im Bereich der Themes angesiedelt sind.[12]

Justin Tadlock, einer der Admins aus dem Theme-Review-Team, verdeutlicht diesen Punkt in einem Kommentar. Mit diesem Absatz sei gemeint, dass Custom-Post-Types, Custom-Taxonomies und Shortcodes nicht erlaubt sind.[13]

Dokumentation und Übersetzungsfähigkeit

Theme-Entwickler sollen Besonderheiten ihres Themes ausreichend dokumentieren. Dazu gehören beispielsweise Limitierungen bei Menü-Items oder eigene Funktionen, die das Theme mitbringt.[14] Überdies müssen alle Strings des Themes übersetzbar sein. Zudem muss eine Text-Domain in der `style.css` festgelegt werden, die auch als Theme-Slug für die Übersetzungen fungiert. In welcher Sprache das Theme in das Theme-Verzeichnis hochgeladen wird, ist egal – diese Sprache muss aber für alle Strings verwendet werden.[15] Da der WordPress-Core amerikanisches Englisch als Standard-Sprache nutzt, ist es am sinnvollsten, für das Theme ebenfalls diese Sprache zu nutzen.[16]

Für die Übersetzung gibt es in WordPress verschiedene Funktionen.[17] Diese teilen sich grob in die Gruppen mit und ohne Kontext auf. Die Funktionen erwarten immer als ersten Parameter den String, der übersetzt werden soll, und als letzten die Text-Domain des Themes. Dazwischen kommt bei den Kontext-Funktionen der Kontext, damit gleiche Wörter mit unterschiedlichen Bedeutungen, wie der englische Begriff *Post*, mehrfach im Übersetzungs-Tool aufgeführt werden. Andernfalls könnte *Post* nur einmal übersetzt werden –

[11] vgl. WordPress: Theme Review Handbook › Required › Core Functionality and Features
[12] vgl. WordPress: Theme Review Handbook › Required › Presentation vs Functionality
[13] vgl. Tadlock (2015): Meeting Minutes 06/11/15 › Kommentar
[14] vgl. WordPress: Theme Review Handbook › Required › Documentation
[15] vgl. WordPress: Theme Review Handbook › Required › Language
[16] vgl. Schilling (2014): Language chooser in 4.0
[17] vgl. WordPress: Codex › L10n

entweder als *Beitrag* oder *Veröffentlichen*.[18] Um eine Beschreibung für Übersetzer zu hinterlegen, wofür ein String genau steht, kann ein PHP-Kommentar mit `translators:` am Anfang genutzt werden. Dieser muss der letzte Kommentar vor Aufruf der Übersetzungsfunktion sein.[19]

Lizenzierung und Theme-Name

Das Theme und die verwendeten Skripte, Bilder und andere Ressourcen müssen zu 100 Prozent unter der GPL oder einer kompatiblen Lizenz stehen. Darüber hinaus muss die Lizenz in der `style.css` angegeben werden, ebenso wie die URL zu der Lizenz. Auch die Lizenzen von verwendeten Schriften, Bildern und Skripten müssen ersichtlich sein. Der komplette Code und das Design des Themes müssen vom Entwickler stammen oder ihm legal gehören.[20]

Bei der Namensgebung des Themes muss beachtet werden, dass die Begriffe *WordPress* oder *Theme* nicht verwendet werden. In allen öffentlich sichtbaren Texten muss WordPress korrekt geschrieben werden – mit großem *W* und *P*.[21]

Optionen und Einstellungen

Optionen des Themes müssen in einem einzelnen Array gespeichert werden. Hinzu kommt, dass Standardwerte nicht in der Datenbank, sondern nur als sogenannte *sane defaults* gespeichert werden dürfen.[22]

Um herauszufinden, ob ein Nutzer die Rechte für die Bearbeitung von Optionen hat, sollte auf die `edit_theme_options`-Fähigkeit geprüft werden, nicht auf eine bestimmte Rolle wie *Administrator*. Theme-Optionen müssen in den Customizer integriert werden.[23]

Plugins und Screenshot

Themes dürfen Plugins empfehlen, aber keine Plugins in ihren eigenen Code integrieren. Darüber hinaus dürfen Themes keine Aufgaben übernehmen, die besser von Plugins erledigt werden können.[24]

---

[18] vgl. WordPress: Plugin Handbook › Internationalization › How to Internationalize Your Plugin › Disambiguation by context
[19] vgl. WordPress: Plugin Handbook › Internationalization › How to Internationalize Your Plugin › Descriptions
[20] vgl. WordPress: Theme Review Handbook › Required › Licensing
[21] vgl. WordPress: Theme Review Handbook › Required › Naming
[22] vgl. Bennett: Using Sane Defaults in Themes
[23] vgl. WordPress: Theme Review Handbook › Required › Options and Settings
[24] vgl. WordPress: Theme Review Handbook › Required › Plugins

Der Screenshot für das Theme sollte das Theme so zeigen, wie es mit den Standard-Optionen aussieht. Als Screenshot ein Logo oder einen Entwurf des Themes zu verwenden, ist nicht erlaubt. Er sollte nicht größer als 1.200 x 900 Pixel sein.[25]

Sicherheit und Privatsphäre

Ein Theme sollte ohne die Zustimmung des Nutzers keine Daten an einen anderen Server senden. Darüber hinaus darf jede Art der Nutzerdaten-Erhebung nur nach Aktivierung durch den Nutzer erfolgen.

Um keine unsicheren Einträge in der Datenbank zu speichern, müssen alle vertrauensunwürdigen Dateneingaben vor Speicherung validiert und bereinigt (im Englischen: *sanitize*) werden. Vor der Ausgabe sollten alle Eingaben, die nicht vertrauenswürdig sind, mit Escaping-Funktionen behandelt werden – für normale Eingabefelder gibt es die `esc_attr()`-Funktion und für Textareas die `esc_textarea()`-Funktion. Zudem darf der Theme-Entwickler keine URL-Shortener für Links im Theme verwenden.[26]

Theme-Shop und Links

Wenn in dem Theme ein Link zu dem Entwickler eingebaut wird, sollte es davon nur einen geben. Für den Fall, dass ein Theme-Entwickler auf seiner Website Themes verkauft, sollte er das nur unter der GPL oder einer vergleichbaren Lizenz tun, um mit seinen kostenfreien Themes in das Theme-Verzeichnis aufgenommen zu werden.[27] Justin Tadlock verdeutlicht diesen letzten Aspekt wie folgt:

> „In a nutshell, if you want to have themes in the directory, all the WordPress themes/plugins on your site need to be 100% GPL or compatible. No split licensing with part of the package under a more restrictive license.“[28]

Stylesheets und Skripte

Skripte, Stylesheets und Favicons dürfen nicht hart in ein Theme geschrieben werden. Eine Ausnahme sind Skripte, die einen Workaround für bestimmte Browser einbinden. Alle anderen Skripte und Stylesheets müssen mit den vorgesehenen Funktionen eingebunden werden. Der Theme-Entwickler darf außerdem keine Analyse- oder Tracking-Codes einbauen und Skripte und Stylesheets nicht minifizieren, außer die Original-Version wird ebenfalls bereitgestellt.

---

[25] vgl. WordPress: Theme Review Handbook › Required › Screenshot
[26] vgl. WordPress: Theme Review Handbook › Required › Security and Privacy
[27] vgl. WordPress: Theme Review Handbook › Required › Selling, credits and links
[28] Tadlock: Themes should be 100% GPL (2015)

Die Entwickler von Themes müssen die bereits mit dem Core ausgelieferten Skripte verwenden, statt eigene Versionen derselben einzubinden. Wenn Themes eigene Skripte einbinden, müssen diese mit dem Theme ausgeliefert werden und dürfen – mit Ausnahme von Google-Bibliotheken – nicht von fremden Servern eingebunden werden. Wenn ein Theme ein bestimmtes Tag im Stylesheet angibt, muss es die erwartete Funktion auch unterstützen beziehungsweise liefern, wofür das Tag steht.

Template

Wenn ein Theme Templates verwendet, müssen die entsprechenden Dateien mit der `get_template_path()-` oder `locate_template()`-Funktion eingebunden werden. Anstatt der `bloginfo()`-Funktion mit entsprechenden Parametern sollen die äquivalenten `*_url()`-Template-Tags genutzt werden. Zudem muss der Entwickler der Template-Hierarchie folgen.[29]

2.2.2 Empfohlene Regeln

Core-Funktionalität und Funktionen

Themes sollten für das Login- und Suchformular das Markup vom Core verwenden. Des Weiteren müssen einige Funktionen, wenn sie eingebunden sind, die Core-Implementierung von WordPress unterstützen. So müssen automatische Feed-Links über die `add_theme_support()`-Funktion mit dem entsprechenden Parameter aktiviert werden, ebenso wie Beitragsbilder, der Custom-Header und Custom-Backgrounds. Beitragsbilder müssen über die `the_post_thumbnail()`-Funktion ausgegeben werden.

Des Weiteren müssen Sidebars mit der `register_sidebar()`-Funktion registriert und der `dynamic_sidebar()`-Funktion ausgegeben werden. Wenn ein Theme eigene Widgets bereitstellt, müssen die mit der `register_widget()`-Funktion an den Action-Hook `widgets_init` gehängt werden. Für die Ausgabe von Menüs ist die `wp_nav_menu()`-Funktion vorgesehen – um die Menüs in der `functions.php` zu registrieren, gibt es die `register_nav_menu()-` oder `register_nav_menus()`-Funktion. Um das Design des Editors anzupassen, muss die `add_editor_style()`-Funktion im Zusammenspiel mit der `editor-style.css`-Datei verwendet werden.[30]

[29] vgl. WordPress: Theme Review Handbook › Required › Templates
[30] vgl. WordPress: Theme Review Handbook › Core Functionality and Features

Design

Die verwendete Schrift eines Themes sollte gut lesbar sein. Das bedeutet, mindestens 14 Pixel groß, es sei denn, die Schrift hat eine große x-Höhe. Daneben sollte bei Fließtext auf eine ausreichende Zeilenhöhe geachtet und die Überschriften so gestaltet werden, dass sie sich deutlich vom Fließtext unterscheiden. Neben der Zeilenhöhe muss auch auf die Zeilenlänge geachtet werden, die sich im Bereich von 45 bis 75 Zeichen bewegen sollte.

Bei der Farbwahl sollte in jedem Fall auf einen ausreichenden Kontrast zwischen Schriftfarbe und Hintergrund geachtet werden. Zudem sollten Nutzer mit Farbfehlsichtigkeit berücksichtigt werden: Wichtige Informationen oder ähnliches sollten nicht lediglich durch eine andere Farbe hervorgehoben werden.

Im Detail sollte ein Theme responsive sein, sich also an den Viewport anpassen und auch auf Smartphones gut dargestellt werden. Design-Details sollten nicht vom Inhalt ablenken, Vektor-Icons mit einem Bild-Fallback für ältere Browser und Animationen nicht nur um ihretwillen eingesetzt werden.[31]

Dokumentation und Links

Für die Dokumentation gilt, dass eine `readme.txt`-Datei beigefügt werden kann. Zudem wäre es wünschenswert, eine Liste der Änderungen in einer `changelog.txt` bereitzustellen – ein Changelog sollte es in jedem Fall geben. Überdies sollten alle Funktionen, die das Theme bereitstellt, ausreichend dokumentiert werden, damit sich auch andere Nutzer gut in dem Code zurechtfinden.[32]

Die Angabe der Theme-URI im Stylesheet ist optional. Wenn sie genutzt wird, muss sie auf eine Seite mit weiteren Informationen zu dem Theme verweisen, nicht auf eine Demo. Wenn eine Demo verlinkt ist, muss deren Inhalt das Theme näher beschreiben. Auch die URI des Theme-Autors ist optional und muss bei Nutzung auf die private Website oder eine Projekt- beziehungsweise Entwickler-Website verweisen.[33]

Template

Wenn ein WordPress-Theme bestimmte Dateien verwendet, muss es für deren Einbindung die entsprechenden Funktionen nutzen. Das ist beispielsweise für die `header.php` die

[31] vgl. WordPress: Theme Review Handbook › Design
[32] vgl. WordPress: Theme Review Handbook › Documentation
[33] vgl. WordPress: Theme Review Handbook › Selling, Themes and Links

`get_header()`-Funktion und `get_footer()` für die `footer.php`. Wenn eigene Templates erstellt werden, sollten sie entsprechend dokumentiert werden.

Themes können die Ausgabe von Funktionen über die Parameter oder Filter steuern. Wenn sie die entsprechenden Theme-Dateien verwenden, müssen relevante Funktionen eingefügt werden, wie etwa die `wp_head()`-Funktion zum Einbinden von Stylesheets, Skripten und mehr.[34]

3 Analyse bereits vorhandener Fotografen-Themes

Um kein Theme zu erstellen, dessen Anspruch an Funktion und Design bereits ein anderes kostenloses Theme erfüllt, müssen die bestehenden WordPress-Themes in diesem Bereich untersucht werden. Die direkte Konkurrenz für das Theme befindet sich im Theme-Verzeichnis, weshalb sich die Untersuchung kostenloser Themes auf dieses Verzeichnis beschränkt.

Darüber hinaus wird auch bei den kostenpflichtigen Themes nach gleichwertigen Themes gesucht, um eventuelle Funktionsunterschiede festzustellen, die nur diese Themes bieten können, weil sie nicht im Theme-Verzeichnis angeboten werden und damit nicht den Anforderungen aus Kapitel 2.2 unterliegen. Für diese Analyse werden zehn WordPress-Themes von ThemeForest, einem der führenden Marktplätze auf dem Gebiet, untersucht.[35]

3.1 Kostenlose Themes im Theme-Verzeichnis

Das Theme-Verzeichnis bietet zur Einschränkung bei der Auswahl verschiedene Tags an, die der Besucher auswählen kann. Eins davon ist das *photoblogging*-Tag, das einen wahrscheinlichen Sucheinstieg für potenzielle Interessenten des Themes bietet. Zum Untersuchungszeitpunkt am 2. November 2015 gibt es unter diesem Tag 154 Themes.[36] Einige davon scheinen doppelt ausgegeben zu werden, weshalb am Ende 126 Themes zur Analyse übrig bleiben.

[34] vgl. WordPress: Theme Review Handbook › Templates
[35] vgl. Koster (2015): Explore WordPress' free theme options
[36] vgl. WordPress: Theme Directory › photoblogging-Tag

Wie aus der Analyse hervorgeht, die der Arbeit in Anhang B beigefügt ist, bietet kein Theme alle Funktionen, die ein Fotografen-Theme nach den Untersuchungen in Kapitel 2.1 bieten sollte. Einige der untersuchten Themes bieten die Möglichkeit, bestimmte Beiträge auf einer Portfolio-Übersicht anzuzeigen, wie etwa das Theme *Pinboard* oder *Virtue*. Bei Virtue muss für diese Funktion allerdings ein Plugin installiert werden – von Haus aus liefert das Theme diese Funktion nicht.

Keins der untersuchten Themes bietet die Möglichkeit, auf einer Seite nur bestimmte Bilder aus dem Portfolio anzuzeigen. Auch die Möglichkeit einer Archiv-Funktion wird von keinem unterstützt.

3.2 Kostenpflichtige Themes von ThemeForest

ThemeForest bietet am 9. November 2015 insgesamt 5.773 WordPress-Themes zum kostenpflichtigen Download an.[37] Für die Untersuchung werden die zehn meistverkauften Themes mit den Tags *portfolio* und *photography* gewählt, die den Begriff *Photography* oder Abwandlungen im Titel tragen.[38] Das Theme *Anan* wird dabei ausgelassen, da bei der Untersuchung am 9. November die Demo nicht funktioniert hat.[39] Die Untersuchungsergebnisse sind in Anhang C zu finden.

Auch bei den hier untersuchten Themes bietet keins alle Funktionen, die aus der Fotografen-Website-Analyse als nützlich hervorgegangen sind. Die Möglichkeit, einen Slider einzubauen, bieten alle untersuchten Themes. Eine Portfolio-Ansicht ist bei acht der zehn Themes möglich, die sich oft auch direkt filtern lässt – einige Themes bieten auch verschiedene Varianten des Portfolio-Layouts an. Eine bestimmte Seite festzulegen, die nur eine Kategorie davon anzeigt, scheint häufig nicht möglich zu sein.

Bei einigen Themes lassen sich Galerien in verschiedenen Arten darstellen – sowohl als Slider als auch als Thumbnail-Übersicht. Die Möglichkeit eines Portfolio-Archivs bietet kein Theme. In allen Fällen scheinen die Portfolio-Elemente strikt von den Blogbeiträgen getrennt und mit einer anderen Inhaltsart umgesetzt zu sein, als WordPress ursprünglich mitbringt – diese Tatsache würde die Themes vom Theme-Verzeichnis bereits ausschließen.

[37] vgl. ThemeForest: Premium WordPress Themes & WordPress Templates
[38] vgl. ThemeForest: Meistverkaufte WordPress-Themes mit Tags „photography“ und „portfolio“
[39] vgl. ThemeForest: ANAN – For Photography Creative Portfolio

Nachteile von Premium-Themes

Auch wenn ein Theme mit fast unbegrenzten Möglichkeiten in der Anpassung auf den ersten Blick für einen Nutzer sehr attraktiv scheint, gibt es große Nachteile, die mit der Nutzung sogenannter *Premium-Themes* einhergehen. Ein großes Problem sind Shortcodes, die von Themes mitgebracht werden. Wenn beispielsweise eine Meldung mit dem Shortcode des Fluxus-Themes umgesetzt wird, dann sieht das in dem Beitrag oder der Seite im Backend folgendermaßen aus:[40]

```
[alert]
This theme is available at ThemeForest for a price of $45. Buy It.
[/alert]
```

Listing 1: Shortcode für eine Meldung im Fluxus-Theme

Im Frontend wird dann eine Meldung ausgegeben, wie in Abbildung 3 zu sehen ist. Wenn der Nutzer aber zu einem anderen Theme wechselt und in seinen Beiträgen und Seiten Shortcodes eingesetzt hat, werden einfach die Shortcodes ausgegeben. Das Ergebnis im Frontend würde dann so aussehen, wie in Listing 1.

Abbildung 3: Mit Shortcode erzeugte Meldung im Fluxus-Theme.
Quelle: Screenshot von inthe.me/demo/fluxus/features/alerts/ am 10. November 2015

Wenn Shortcodes also nicht von einem Plugin bereitgestellt werden, gehen diese Funktionen bei einem Theme-Wechsel verloren. Der Nutzer muss dann für funktionierende Inhaltselemente entweder das alte Theme weiternutzen oder alle Shortcodes durch eine andere Lösung ersetzen.

Ein ähnliches Problem bekommen Nutzer durch die inhaltserstellenden Funktionen solcher Themes. Bei dem Invictus-Theme werden die Galerien vermutlich durch einen Custom-Post-Type realisiert, wie in dem Video auf der Hilfe-Seite zu sehen ist.[41] Das bedeutet, dass nach einem Theme-Wechsel sämtliche Galerien, die über diese Funktion angelegt wurden, nicht mehr vorhanden sind. Der Nutzer muss sich die Galerien mit einer anderen Lösung wieder neu anlegen.

[40] vgl. inTheme: Alerts – Fluxus
[41] vgl. do.media: Help Document – Invictus › III. Adding Photos to your gallery

Das ist der Grund, warum im Theme-Verzeichnis keine Themes erlaubt sind, die Shortcodes, Custom-Post-Types oder andere Funktionen beinhalten, über die der Nutzer Inhalte erstellen kann, die nach einem Theme-Wechsel verloren gehen. Justin Tadlock hat nicht erlaubte Funktionen und einige Grauzonen-Situationen in einem Blog-Post erläutert.[42]

4 Vorbereitung auf die Theme-Entwicklung

Außer einer funktionierenden WordPress-Installation gibt es einige Dinge, mit denen die Theme-Entwicklung leichter von der Hand geht. Dazu gehören hilfreiche Plugins, WordPress-Einstellungen sowie Demo-Inhalte, die auch verschiedene Grenzfälle behandeln.

4.1 Debug-Modus

Allgemein für die Entwicklung mit WordPress nützlich und empfehlenswert ist die Aktivierung des Debug-Modus. Wenn sich eine WordPress-Installation in diesem Modus befindet, werden PHP-Fehlermeldungen sowohl im Frontend als auch im Backend angezeigt.

Um den Debug-Modus zu aktivieren, muss eine Zeile in der `wp-config.php` angepasst werden, die sich im obersten Ordner der WordPress-Installation oder eine Ebene höher befindet:

```
define('WP_DEBUG', false);
```

Listing 2: Debug-Modus deaktiviert

Hier muss der Wert `false` in `true` geändert werden, um den Debug-Modus zu aktivieren.[43]

4.2 Entwickler-Plugins

Das Theme-Review-Team empfiehlt ihren Testern verschiedene Plugins, die sie bei dem Theme-Review nutzen sollen.[44] Das bedeutet im Umkehrschluss, dass diese Plugins auch nützlich bei der Entwicklung von Themes sind. In dem Beitrag vom 24. Juli 2015 sind folgende Plugins empfohlen:

[42] vgl. Tadlock (2015): The theme review team's content creation discussion
[43] vgl. WordPress: Debugging in WordPress
[44] vgl. Castaneda (2015): Theme Review Tools

Developer

Mit dem Developer-Plugin können verschiedene Plugins installiert werden, die bei der Entwicklung hilfreich sind, wie beispielsweise ein Plugin zum schnellen Wechseln des Backend-Nutzers.[45] Die Plugins, die *Developer* zur Installation anbietet, überschneiden sich zum großen Teil mit den weiteren empfohlenen Plugins – lediglich das Beta-Tester-Plugin kann nicht über dieses Plugin installiert werden.

Theme Check

Das Theme-Check-Plugin ist das wichtigste Plugin für die Theme-Entwicklung, da alle Reviewer ein Theme mit diesem Plugin testen und bereits ein automatischer Test beim Upload durchgeführt wird. Dabei prüft das Plugin beispielsweise, ob keine Shortcodes und Custom-Post-Types erstellt werden.[46]

Debug Bar

Das Plugin *Debug Bar* fügt ein Debug-Menü in die Admin-Toolbar ein, das die Datenbankabfragen, Cache und Weiteres anzeigt. Wenn der Debug-Modus aktiviert ist, zeigt es auch PHP-Warnungen und -Notizen an.[47]

Log Deprecated Notices

Das Plugin *Log Deprecated Notices* macht genau das, was der Name vermuten lässt: Es meldet veraltete Funktionen, Funktionsargumente und Dateien. Darüber hinaus meldet es auch falsch genutzte Funktionen, die WordPress seit Version 3.1 meldet.[48]

Monster Widget

Mit dem Plugin *Monster Widget* wird ein Widget bereitgestellt, das alle Core-Widgets enthält. Um alle Widgets zu testen, muss also lediglich dieses eine Widget in eine Sidebar gezogen werden.[49]

WordPress Beta Tester

Um immer die neueste Entwickler-Version von WordPress zu nutzen, kann das Plugin *WordPress Beta Tester* genutzt werden. Dieses Plugin ermöglicht es, zwei verschiedene Entwickler-Zweige zu testen: Standardmäßig aktualisiert das Plugin eine WordPress-

[45] vgl. automattic u.a.: „Developer"-Plugin
[46] vgl. Wood/Prosser: „Theme Check"-Plugin
[47] vgl. WordPress u.a.: „Debug Bar"-Plugin
[48] vgl. Nacin: „Log Deprecated Notices"-Plugin
[49] vgl. automattic/Fields/Willett: „Monster Widget"-Plugin

Installation auf die neuesten Beta- oder Release-Candidate-Versionen. Es gibt aber auch die Möglichkeit, den allerneuesten Entwicklungsstand als Ziel einzustellen, in dem allerdings mehr Bugs auftreten können.[50]

Regenerate Thumbnails

WordPress generiert beim Hochladen von Bildern verschiedene Thumbnail-Größen. Diese Größen können vom Theme-Entwickler verändert werden oder es können weitere hinzugefügt werden. Um die Thumbnails bereits hochgeladener Bilder neu zu erstellen, gibt es das Plugin *Regenerate Thumbnails*.[51]

4.3 Test-Daten

Um möglichst viele Anwendungsfälle wie etwa Artikel ohne Titel, einen Titel aus einem langen Wort oder einen Beitrag ohne Inhalt abzudecken, gibt es Test-Daten, die sich an Entwickler richten. Neben den offiziellen Test-Daten von dem Theme-Review-Team gibt es noch ein weiteres Test-Set, das auf Grundlage der offiziellen Daten erstellt aber auch weiterentwickelt wurde.[52, 53] Diese Test-Daten können im WordPress-Backend unter *Werkzeuge › Daten importieren* in die WordPress-Installation integriert werden.

5 Entwicklung des WordPress-Themes

Das Theme soll einen klassischen Aufbau haben. Oben findet ein Header-Bereich Platz, in dem auf der linken Seite der Titel der Website steht und auf der rechten Seite eine horizontale Navigation. Darunter befindet sich der eigentliche Inhaltsbereich, der von einem Footer abgeschlossen wird.

5.1 Template-Hierarchie in WordPress

Ein WordPress-Theme besteht aus verschiedenen Dateien. Darunter gibt es einige, die normalerweise immer verwendet werden, wie die `header.php` mit den Header-Informationen und die `footer.php`. Es gibt in WordPress sieben verschiedene Arten von

[50] vgl. Westwood: „WordPress Beta Tester“-Plugin
[51] vgl. Mills: „Regenerate Thumbnails“-Plugin
[52] vgl. WordPress: Codex › Theme Unit Test
[53] vgl. Novotny: WP Test

angezeigten Seiten, für die jeweils unterschiedliche Dateien in einer bestimmten Reihenfolge von WordPress gesucht werden, um den Inhalt anzuzeigen. Einen guten Überblick über die Template-Hierarchie bietet eine Grafik, die in Anhang D der Arbeit beigefügt ist.

5.1.1 Archiv-Seite mit Beispiel

Eine Archiv-Seite kann die Beiträge eines Autors anzeigen, die Beiträge aus einer Kategorie, die mit einem bestimmten Tag, zeitliche Archive sowie Archive zu Custom-Post-Types. Wenn beispielsweise eine Kategorie-Seite angezeigt werden soll, dann sucht WordPress in dem Theme zuerst nach einer Datei, die dem Muster `category-$slug.php` entspricht. Der Slug ist dabei die URL-Form der Kategorie – bei der Kategorie *WordPress* wäre der Slug standardmäßig *wordpress*.

Es würde also zuerst nach der Datei `category-wordpress.php` gesucht. Wenn diese Datei nicht gefunden wird, wird eine Datei nach dem Muster `category-$id.php` gesucht, wobei `$id` der ID der Kategorie entspricht. Wenn auch diese Datei nicht vorhanden ist, wird auf die `category.php` zurückgegriffen. Danach würde WordPress nach der `archive.php` suchen, die ein Fallback für alle Archiv-Arten darstellt. Wenn es sich um ein Archiv mit einer Pagination handelt und keine `archive.php` gefunden wird, sucht WordPress die `paged.php`. Als letzten Fallback gibt es die `index.php`, die für alle Inhaltsarten die letzte Möglichkeit und deshalb eine notwendige Datei in jedem WordPress-Theme ist.

5.1.2 Einzelseite

Neben den Archiv-Seiten gibt es die Einzelseiten, die entweder die Einzelansicht eines Beitrags oder einer Seite sein können. Die gängigen Dateien, die für diesen Typ verwendet werden, sind die `single.php` für die Einzelansicht eines Beitrags und die `page.php` für eine Seite.

Außerdem gibt es hier unter anderem wie bei den Archiv-Seiten noch die Möglichkeit, ein Template für eine bestimmte Seite anzulegen, die anhand des Slugs oder der ID erkannt wird. Auch Templates für bestimmte Anhang-Typen wie etwa Bilder oder MP3-Dateien sind möglich. Seit WordPress 4.3 gibt es vor dem Fallback `index.php` noch eine `singular.php`, die das Äquivalent zur `archive.php` darstellt.

5.1.3 Startseite der Website, Blog-Übersicht, Kommentar-Pop-Up, 404-Seite und Suchergebnisse

Um die Startseite der WordPress-Installation darzustellen, wird zuerst nach der `front-page.php` gesucht. Wenn die nicht vorhanden ist und es sich um eine statische Seite als Startseite handelt, wird auf dieselben Dateien wie bei einer normalen Seite zurückgegriffen. Wenn die letzten Beiträge auf der Startseite angezeigt werden, wird vor der `index.php` noch die `home.php` gesucht.

Die `home.php` ist die erste Datei, die für die Blog-Übersicht genutzt wird. Ist sie nicht vorhanden, wird auf die `index.php` zurückgegriffen. Für die Anzeige des Kommentar-Pop-Ups kann die `comments-popup.php` vom Theme-Entwickler bereitgestellt werden oder es wird wieder die `index.php` verwendet. Für die 404-Seite gibt es in der Template-Hierarchie die `404.php` und für die Suchergebnisse die `search.php`.

5.2 Hooks in WordPress

Wichtig in der WordPress-Entwicklung sind die sogenannten Hooks, die standardisierte Möglichkeiten bieten, sich in das System *einzuhängen*. Dabei werden zwei Arten von Hooks unterschieden: Action- und Filter-Hooks.[54]

5.2.1 Action-Hooks

Action-Hooks werden bei einer bestimmten Aktion ausgeführt, zum Beispiel beim Veröffentlichen eines Kommentars. Durch diese Action-Hooks kann ein Theme- oder Plugin-Entwickler auf bestimmte Aktionen reagieren. Um beispielsweise eine E-Mail an den Administrator zu schicken, wenn ein neuer Kommentar geschrieben wurde, kann der Hook `comment_post` verwendet werden, wie in Listing 3 zu sehen ist – das Beispiel ist abgewandelt dem Werk von Williams, Damstra und Stern entnommen.[55, 56]

```
function email_new_comment() {
   wp_mail( 'admin@example.com', 'Neuer Kommentar',
         'Es gibt einen neuen Kommentar auf deiner Website!' );
}
```

[54] Williams/Damstra/Stern (2015), S. 166-171.
[55] vgl. WordPress: Code Reference › Hooks › comment_post
[56] Williams/Damstra/Stern (2015), S. 167.

```
add_action( 'comment_post', 'email_new_comment' );
```

Listing 3: Beispiel für einen Action-Hook

An die `add_action()`-Funktion wird zuerst der Hook übergeben und als zweiter Parameter der Name der Funktion. Darüber hinaus gibt es noch Parameter um die Priorität sowie die Anzahl der Funktionsparameter anzugeben.[57]

5.2.2 Filter-Hooks

Im Gegensatz zu Action-Hooks kann mittels Filter-Hooks der Inhalt vor dem Speichern in die Datenbank oder vor der Anzeige im Frontend verändert werden. So kann beispielsweise vor Ausgabe des Beitragsinhaltes dieser mit einem Filter-Hook verändert werden, wie das Beispiel in Listing 4 zeigt, das leicht umgestellt aus dem Werk von Brad Williams, David Damstra und Hal Stern entnommen ist.[58]

```
function prowp_profanity_filter( $content ) {
   $profanities = array( 'sissy', 'dummy' );
   $content     = str_ireplace(
      $profanities, '[censored]', $content
   );

   return $content;
}

add_filter( 'the_content', 'prowp_profanity_filter' );
```

Listing 4: Beispiel für einen Filter-Hook

Statt der `add_action()`-Funktion wird hier `add_filter()` verwendet, das dieselben Parameter wie `add_action()` erwartet.[59] Als Hook wird in diesem Beispiel `the_content` verwendet, das die Veränderung des Seiten- oder Beitragsinhalts ermöglicht.[60] Hier werden beispielsweise in `prowp_profanity_filter()` die Begriffe *sissy* und *dummy* durch *[censored]* ersetzt, bevor der Inhalt ausgegeben wird.

[57] vgl. WordPress: Code Reference › Functions › add_action
[58] Williams/Damstra/Stern (2015), S. 166-167.
[59] vgl. WordPress: Code Reference › Functions › add_filter
[60] vgl. WordPress: Code Reference › Hooks › the_content

5.3 Customizer

Der Customizer ermöglicht dem Anwender eine Live-Vorschau von Änderungen und ist über den Menüpunkt *Design* › *Anpassen* zu erreichen. Seit April 2015 müssen Themes für das Theme-Verzeichnis den Customizer nutzen, um Theme-Optionen einzubinden – eigene Optionsseiten sind nicht mehr erlaubt.[61]

Um eine Customizer-Option zu erstellen, müssen mindestens die Methoden `add_setting()` sowie `add_control()` von `WP_Customize_Manager` verwendet werden.[62, 63, 64] Die Funktion, die diese Methoden nutzt, muss an den Action-Hook `customize_register` übergeben werden.[65] Mit `add_setting()` wird eine Customizer-Einstellung erstellt – sie kümmert sich um die Speicherung und die Bereinigung der Daten. `add_control()` hingegen erstellt zu einer bestimmten Einstellung das Formularelement, über das der Nutzer die Werte anpassen kann.

Daneben gibt es noch weitere Methoden, beispielsweise kann mit `add_section()` und `add_panel()` ein eigener Bereich erstellt werden, um die Theme-Optionen zusammenzufassen.[66, 67]

5.4 Metadaten des Themes in der style.css

Ein Theme muss in jedem Fall eine `style.css` beinhalten.[68] Hier werden die Metadaten eingetragen, die bei der Theme-Auswahl im WordPress-Backend angezeigt werden. Hierbei können eine Reihe von Tags wie etwa `Theme Name` und `Licence URI` genutzt werden.[69] Um in das Theme-Verzeichnis aufgenommen zu werden, sind mindestens die Tags

[61] vgl. Tadlock (2015): Details on the new theme settings (customizer) guideline
[62] vgl. WordPress: Code Reference › Classes › WP_Customize_Manager › WP_Customize_Manager::add_setting
[63] vgl. WordPress: Code Reference › Classes › WP_Customize_Manager › WP_Customize_Manager::add_control
[64] vgl. WordPress: Code Reference › Classes › WP_Customize_Manager
[65] vgl. WordPress: Code Reference › Hooks › customize_register
[66] vgl. WordPress: Code Reference › Classes › WP_Customize_Manager › WP_Customize_Manager::add_section
[67] vgl. WordPress: Code Reference › Classes › WP_Customize_Manager › WP_Customize_Manager::add_panel
[68] vgl. WordPress: Theme Handbook › Theme Basics › Template Files › Common WordPress template files
[69] vgl. WordPress: Codex › Theme Development › Theme Stylesheet

`Theme Name`, `Description`, `Author`, `Version` sowie `License` und `License URI` zu verwenden – auf diese Tags prüft das Theme-Check-Plugin.[70]

Die `style.css` des WordPress-Themes sieht folgendermaßen aus:

```
/*
Theme Name: Hannover
Theme URI: https://florianbrinkmann.de/wordpress-themes/hannover/
Author: Florian Brinkmann
Author URI: https://florianbrinkmann.de
Description: Hannover is great for photographers. You can show all gal-
lery and image posts on one portfolio page.
Version: 1.0
License: GNU General Public License v2
License URI: http://www.gnu.org/licenses/gpl-2.0.html
Tags: photoblogging, white, responsive-layout
Text Domain: hannover
*/
```

Listing 5: Metadaten in der style.css

Zuerst wird mit *Hannover* der Name des Themes festgelegt und die URI angegeben, die nähere Informationen zum Theme bereithält. Im Anschluss wird der Entwickler sowie dessen Website angegeben und eine Beschreibung des Themes eingefügt. Nach Angabe der aktuellen Theme-Version wird die Lizenz sowie ein Link dazu eingetragen. Durch Angabe von Tags kann das Theme besser im Theme-Verzeichnis gefunden werden – hier sind nur Werte erlaubt, die sich im englischsprachigen Directory filtern lassen. Die Angabe `Text Domain` ist wichtig für die Übersetzbarkeit des Themes.[71]

5.5 Kopfbereich des Themes in der header.php

In die `header.php` eines WordPress-Themes gehören sowohl der sichtbare als auch unsichtbare Header.[72] Der unsichtbare Bereich ist alles, was vor dem öffnenden `body`-Tag steht, also der `Doctype`, das öffnende `html`-Tag sowie das `head`-Element mit den entsprechenden Inhalten. Der sichtbare Teil ist der Header, den der Website-Besucher sehen kann.

[70] vgl. Wood/Obenland (2015): theme-check/checks/style_needed.php
[71] vgl. WordPress: Theme Handbook › Internationalization › Text Domains
[72] vgl. WordPress: Theme Handbook › Theme Basics › Template Files › Template Partials

5.5.1 Unsichtbarer Teil des Headers

```
<!DOCTYPE html>
<html <?php language_attributes(); ?> class="no-js">
<head>
   <meta charset="<?php bloginfo( 'charset' ); ?>">
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <?php if ( is_singular() && pings_open() ) { ?>
      <link rel="pingback" href="<?php bloginfo( 'pingback_url' ); ?>">
   <?php }
   wp_head(); ?>
</head>
<body <?php body_class(); ?>>
```

Listing 6: Unsichtbare Seiteninformationen in der header.php

Zuerst wird in Listing 6 nach dem HTML5-Doctype innerhalb des `html`-Elements mit der `language_attributes()`-Funktion das `lang`-Attribut ausgegeben, dessen Inhalt abhängig von der gewählten Sprache im WordPress-Backend ist.[73] Um den Zeichensatz der Website auszugeben, wird die Funktion `bloginfo()` mit dem Parameter `charset` genutzt.[74]

Damit das responsive Layout auf mobilen Geräten richtig angezeigt wird, muss das `viewport`-Meta-Tag eingefügt werden.[75] Im Anschluss wird mit den Funktionen `is_singluar()` und `pings_open()` geprüft, ob der Nutzer sich auf einer Einzelansicht befindet und ob Pingbacks aktiviert sind.[76, 77] Wenn das der Fall ist, wird die Pingback-URL mittels der `bloginfo()`-Funktion und dem Parameter `pingback_url` ausgegeben.[78] Darüber ist es möglich, automatisch andere Websites im Kommentarbereich zu erwähnen, die auf den Beitrag verlinkt haben.[79]

Wichtig ist der Einsatz von `wp_head()`.[80] Diese Funktion führt den Action-Hook `wp_head` aus, mit dem WordPress die Skripte, Stylesheets, Meta-Tags und beispielsweise das `title`-Element einfügt.[81] Darüber hinaus nutzen auch Plugins diesen Hook, um Skripte, Stylesheets und Ähnliches einzubinden.

---

[73] vgl. WordPress: Code Reference › Functions › language_attributes
[74] vgl. vlastuin: Code Reference › Functions › bloginfo › Show Character Set
[75] vgl. xrds u.a. (2015): Using the viewport meta tag to control layout on mobile browsers
[76] vgl. WordPress: Code Reference › Functions › is_singular
[77] vgl. WordPress: Code Reference › Functions › pings_open
[78] vgl. WordPress: Codex › Function Reference › bloginfo › Parameters
[79] vgl. WordPress: Codex › Introducion to Blogging › Pingbacks
[80] vgl. WordPress: Code Reference › Functions › wp_head
[81] vgl. WordPress: Code Reference › Hooks › wp_head

Abschließend werden im `body`-Element mit `body_class()` abhängig von der angezeigten Seite unterschiedliche Klassen eingefügt.[82] Welche das sind, kann in der `get_body_class()`-Funktion eingesehen werden, die von `body_class()` aufgerufen wird. Bei einem Blick in den Quellcode wird beispielsweise ersichtlich, dass auf der Startseite die Klasse `home` ausgegeben wird.[83]

5.5.2 Skip-To-Content-Link

Bevor der dauerhaft sichtbare Teil des Headers eingefügt wird, muss eine Hilfe für Screen-Reader-Nutzer sowie Nutzer, die nur die Tastatur nutzen, eingefügt werden. Sie müssen die Möglichkeit haben, den Header zu überspringen, und gleich beim Inhalt der Seite weiterzumachen, damit sie nicht gezwungen sind, bei jedem neuen Seitenaufruf die komplette Navigation durchzugehen.

```
<a class="screen-reader-text skip-link" href="#content">
   <?php _e( '[Skip to Content]', 'hannover' ); ?>
</a>
```

Listing 7: Skip-To-Content-Link in der header.php

Listing 7 zeigt den Code, der diese Möglichkeit umsetzt. Ein Link-Element wird mit den Klassen screen-reader-text und skip-link ausgetauscht, damit er mit CSS versteckt und beim Fokus-Zustand mit der Tastatur angezeigt werden kann. Der Link verweist auf die ID `content`, in der sich der Hauptinhalt des Themes befinden wird. Der Link-Text wird mit der Übersetzungsfunktion `_e()` angezeigt, damit er direkt ausgegeben wird.[84]

5.5.3 Sichtbarer Header

Im sichtbaren Header-Bereich werden die Elemente eingebunden, die auf jeder Seite gleich sind. Das sind im Fall des Hannover-Themes das Logo beziehungsweise der Seitentitel und die optionale Beschreibung auf der linken und das Menü auf der rechten Seite. Wie der Teil nach Fertigstellung aussieht, ist in Abbildung 4 zu sehen.

[82] vgl. WordPress: Code Reference › Functions › body_class
[83] vgl. WordPress: Core-Trac › tags/4.3.1/src/wp-includes/post-template.php › Zeile 544
[84] vgl. WordPress: Code Reference › Functions › _e

Abbildung 4: Screenshot des Theme-Headers.

Um ein Logo festlegen zu können, soll die Header-Bild-Funktion von WordPress eingesetzt werden. Wenn der Theme-Nutzer kein Bild auswählt, wird stattdessen der Titel der Website angezeigt.

Ausgabe des Header-Bildes

```
<header id="header">
   <div class="site-branding">
      <?php if ( get_header_image() ) {
         if ( is_front_page() && is_home() ) { ?>
            <h1 class="logo"><img src="<?php header_image(); ?>"
               alt="<?php bloginfo( 'name' ); ?>"></h1>
         <?php } else {
            if ( ! is_front_page() ) { ?>
               <a href="<?php echo esc_url( home_url( '/' ) ); ?>"
                  rel="home">
            <?php } ?>
            <img class="logo" src="<?php header_image(); ?>"
               alt="<?php bloginfo( 'name' ); ?>">
            <?php if ( ! is_front_page() ) { ?>
               </a >
            <?php }
         }
      } else {
```

Listing 8: Ausgabe vom Header-Bild in der header.php

In Listing 8 ist der Quellcode abgebildet, der für die Ausgabe des Header-Bildes sorgt, sofern es im Backend eingestellt ist. Innerhalb des `header`-Elements werden diese Informationen in ein `div` verpackt, um später einfach auf der linken Seite platziert werden zu können.

Anschließend wird mit `if ( get_header_image() )` überprüft, ob ein Header-Bild gesetzt ist.[85] Danach muss vor der Ausgabe des Bildes erneut eine Entscheidung getroffen werden, um die bestmögliche Barrierefreiheit des Themes zu gewährleisten. Rian Rietveld, WordPress-Entwicklerin mit Fokus auf Barrierefreiheit, hat dazu einen interessanten Artikel geschrieben, nach dem nur auf der Startseite der Website-Titel oder das Logo in ein

[85] vgl. WordPress: Code Reference › Functions › get_header_image

`h1`-Element eingefasst werden soll. Auf anderen Seiten ist die `h1` der Titel der Seite, der Beitragstitel et cetera.[86]

Aus diesem Grund wird mit `if ( is_front_page() && is_home() )` geprüft, ob der Header gerade auf der Startseite angezeigt wird, die gleichzeitig die Blog-Seite ist.[87, 88] Ist diese Bedingung erfüllt, wird innerhalb einer `h1`-Überschrift das Logo ausgegeben. Die URL zu dem Bild liefert die Funktion `header_image()` und als alternativen Text, wenn das Bild nicht geladen werden kann, wird mit der `bloginfo()`-Funktion und dem Parameter `name` der Name der Website ausgegeben.[89, 90] Falls der Nutzer sich auf einer anderen Seite befindet, wird das Logo mittels der `home_url()`-Funktion auf die Startseite verlinkt und ohne Überschriften-Element ausgegeben.[91] Um sicherzugehen, dass die zurückgegebene URL valide ist, wird die Ausgabe mit `esc_url()` bereinigt.[92]

Ausgabe des Website-Titels

```
    if ( is_front_page() && is_home() ) { ?>
        <h1 class="site-title"><?php bloginfo( 'name' ); ?></h1>
    <?php } else {
        if ( ! is_front_page() ) { ?>
            <a href="<?php echo esc_url( home_url( '/' ) ); ?>" rel="home">
        <?php } ?>
        <p class="site-title"><?php bloginfo( 'name' ); ?></p>
        <?php if ( ! is_front_page() ) { ?>
            </a >
        <?php }
    }
}
```

Listing 9: Ausgabe des Website-Titels in der header.php

Wenn kein Header-Bild festgelegt ist, soll stattdessen der Website-Titel ausgegeben werden. Hierbei müssen dieselben Gesichtspunkte der Barrierefreiheit beachtet werden wie bei dem Logo. Deshalb wird zuerst geprüft, ob die Startseite angezeigt wird und es sich dabei um die Blog-Ansicht handelt. In diesem Fall wird der Seitentitel ohne Verlinkung als `h1`-Überschrift ausgegeben. Andernfalls wird der Name als einfacher Absatz ausgezeichnet und – sofern nicht die Startseite angezeigt wird – verlinkt.

[86] vgl. Rietveld (2014): HTML5 Headings in WordPress: A11y versus SEO?
[87] vgl. WordPress: Code Reference › Functions › is_front_page
[88] vgl. WordPress: Code Reference › Functions › is_home
[89] vgl. WordPress: Code Reference › Functions › header_image
[90] vgl. WordPress: Codex › Function Reference › bloginfo › Parameters
[91] vgl. WordPress: Code Reference › Functions › home_url
[92] vgl. WordPress: Code Reference › Functions › esc_url

Ausgabe der Beschreibung

Unter dem Header-Bild oder dem Website-Titel soll die Beschreibung angezeigt werden, wenn sie vom Theme-Nutzer im Backend eingetragen wurde.

```
    $description = get_bloginfo( 'description', 'display' );
    if ( $description ) { ?>
        <p class="site-description"><?php echo $description; ?></p>
    <?php } ?>
</div>
```

Listing 10: Ausgabe der Beschreibung in der header.php

An die `get_bloginfo()`-Funktion können dieselben Parameter übergeben werden, wie an `bloginfo()`, da `bloginfo()` lediglich das Ergebnis des `get_bloginfo()`-Aufrufs ausgibt.[93, 94] Bei direkter Nutzung der `get_bloginfo()`-Funktion wird das Ergebnis nicht gleich im Frontend ausgegeben sondern kann für weitere Operationen genutzt werden.[95]

Mit `get_bloginfo( 'description', 'display' )` wird die Beschreibung der Website ermittelt und durch Angabe des zweiten Parameters für die Anzeige im Frontend bereinigt. Wenn die Variable `$description` nicht leer ist, der Nutzer also eine Beschreibung im Backend eingetragen hat, wird diese als Absatz angezeigt.

Ausgabe des Menüs

Auf der rechten Seite des Headers soll das Hauptmenü der Website dargestellt werden. Die entsprechende Funktion, die für die Ausgabe von Menüs sorgt, ist `wp_nav_menu()`:

```
<?php if ( has_nav_menu( 'primary' ) ) { ?>
    <button id="menu-toggle" class="menu-toggle">
        <?php _e( 'Menu', 'hannover' ); ?>
    </button>
    <nav>
        <h2 class="screen-reader-text">
            <?php /* translators: hidden screen reader headline for the
                main navigation */
            _e( 'Main navigation', 'hannover' ); ?>
        </h2>
        <?php wp_nav_menu(
            array(
                'theme_location' => 'primary',
                'menu_class'     => 'primary-menu',
                'container'      => ''
            )
```

[93] vgl. WordPress: Codex › Function Reference › get_bloginfo › Parameters
[94] vgl. WordPress: Code Reference › Functions › bloginfo › Source
[95] vgl. WordPress: Code Reference › Functions › get_bloginfo

```
            ); ?>
        </nav>
    <?php } ?>
</header>
<div id="content">
```

Listing 11: Ausgabe des Menüs in der header.php

Wie in Listing 11 dargestellt, wird zunächst geprüft, ob ein Menü zur Ausgabe vorhanden ist. Dafür wird `has_nav_menu()` verwendet, der als Parameter eine Menüposition übergeben werden muss.[96] Anschließend wird ein `button`-Element angelegt, das nachher für die Anzeige des Menüs in kleinen Viewports genutzt wird, dessen Beschriftung erneut über `_e()` ausgegeben wird. Die Navigation selbst wird von dem HTML5-Element `nav` umschlossen, innerhalb dessen für Screen-Reader-Nutzer eine `h2`-Überschrift platziert wird.[97] Um den Übersetzern die Arbeit zu erleichtern, wird vor `_e()` ein `translators:`-Kommentar eingefügt, der in Übersetzungs-Tools als Hilfe angezeigt wird.[98]

Im Anschluss wird das Menü angezeigt. Die `wp_nav_menu()`-Funktion erwartet als Parameter ein Options-Array, dessen wichtigster Schlüssel `theme_location` ist – der entsprechende Wert gibt an, welche Menüposition angezeigt werden soll. Dieser Position muss der Nutzer im Backend ein Menü zuweisen, um es im Frontend darstellen zu lassen. Als Klasse soll dem Menü `primary-menu` zugewiesen und ein zusätzlicher Container zu dem `nav`-Element soll nicht ausgegeben werden. Der Standard-Wert wäre hier ein `div`.[99]

Um diesen Teil in späteren Dateien nicht wiederholen zu müssen, wird auch das umschließende `div` für den Inhalt inklusive eventuell vorhandener Sidebar in der `header.php` geöffnet.

Menüposition registrieren

Damit die Menüposition im Backend ausgewählt werden kann, muss sie in der `functions.php` registriert werden. Hierfür gibt es in WordPress zwei verschiedene Möglichkeiten: Mit der `register_nav_menu()`-Funktion kann ein einzelnes Menü registriert werden – `register_nav_menus()` erlaubt die Registrierung mehrerer Menüs.[100, 101]

---

[96] vgl. WordPress: Code Reference › Functions › has_nav_menu
[97] vgl. Perrier u.a. (2015): MDN › HTML › <nav>
[98] vgl. WordPress: Plugin Handbook › Internationalization › How to Internationalize Your Plugin › Descriptions
[99] vgl. WordPress: Code Reference › Functions › wp_nav_menu
[100] vgl. WordPress: Code Reference › Functions › register_nav_menu
[101] vgl. WordPress: Code Reference › Functions › register-nav_menus

Um später leichter weitere Menüpositionen einrichten zu können, wird in Listing 12 auf die `register_nav_menus()`-Funktion zurückgegriffen. Als Parameter muss ein Array übergeben werden, das als Schlüssel einen eindeutigen Bezeichner der Position festlegt und als Wert die Beschriftung für das Backend enthält. An dieser Stelle kommt die Übersetzungs-Funktion `__()` zum Einsatz, die wie `_e()` funktioniert, aber den String nicht sofort ausgibt.[102] Davor wird ein Kommentar für Übersetzer eingefügt, damit klar ist, wofür dieser String ist. Damit `hannover_register_menus()` früh genug ausgeführt wird, wird sie an den `init`-Hook gehängt. [103]

```php
<?php
function hannover_register_menus() {
   register_nav_menus(
      array(
         /* translators: Name of menu position in the header */
         'primary' => __( 'Primary Menu', 'hannover' )
      )
   );
}

add_action( 'init', 'hannover_register_menus' );
```

Listing 12: Registrierung der Hauptmenüs in der functions.php

5.6 Theme-Support für Post-Formate, Feed-Links, Title-Tag, Beitragsbilder, Header-Bild und HTML5

In WordPress gibt es einige Funktionen, die mit der `add_theme_support()`-Funktion aktiviert werden müssen. Dazu gehören beispielsweise Beitragsbilder, Post-Formate wie *Galerie* und *Bild* sowie automatische Feed-Links. Wie in Listing 13 zu sehen, wird die `hannover_add_theme_support()`-Funktion an den Hook `after_setup_theme` angehängt, da `init` in diesem Fall zu spät sein könnte.[104]

```php
function hannover_add_theme_support() {
   add_theme_support( 'custom-header' );
   add_theme_support( 'automatic-feed-links' );
   add_theme_support( 'title-tag' );
   add_theme_support( 'post-formats', array(
      'aside',
      'link',
      'gallery',
      'status',
      'quote',
```

[102] vgl. WordPress: Code Reference › Functions › __
[103] vgl. WordPress: Code Reference › Hooks › init
[104] vgl. WordPress: Code Reference › Functions › add_theme_support

```
        'image',
        'video',
        'audio',
        'chat'
    ) );
    add_theme_support( 'html5', array(
        'comment-list',
        'comment-form',
        'search-form',
        'gallery',
        'caption',
    ) );
    add_theme_support( 'post-thumbnails' );
}

add_action( 'after_setup_theme', 'hannover_add_theme_support' );
```

Listing 13: add_theme_support() in der functions.php

Um eine Funktion zu aktivieren, muss sie als Parameter an die Funktion `add_theme_support()` übergeben werden. Mit der Übergabe des Parameters `custom-header` wird dem Theme-Nutzer ermöglicht, im Customizer ein Header-Bild festzulegen, das im Fall des Hannover-Themes als Logo verwendet wird.[105]

`automatic-feed-links` fügt Links zu RSS-Feeds in das `head`-Element der Website ein und mit Angabe von `title-tag` wird seit WordPress 4.1 der Dokumenttitel eingefügt. [106, 107] Um die Unterstützung für Post-Formate zu aktivieren, müssen zwei Parameter an die `add_theme_support()`-Funktion übergeben werden: Zuerst `post-formats`, gefolgt von einem Array mit den Formaten, die aktiviert werden sollen. Das Hannover-Theme wird alle möglichen Formate unterstützen.[108]

Zudem ist es möglich, das Markup von WordPress-Funktionen, wie beispielsweise das Galerie-Markup, auf HTML5 umzustellen. Dazu wird als erster Parameter `html5` und als zweiter ein Array übergeben, das die Bereiche enthält, die mit dem modernen Markup ausgeliefert werden sollen. Auch hier wird das Theme alle möglichen Bereiche unterstützen.[109] Um Beitragsbilder zu ermöglichen, muss der Parameter `post-thumbnails` übergeben werden.[110]

[105] vgl. WordPress: Codex › add_theme_support › Custom Header
[106] vgl. WordPress: Codex › add_theme_support › Feed Links
[107] vgl. Obenland (2014): Title Tags in 4.1
[108] vgl. WordPress: Codex › Post Formats › Supported Formats
[109] vgl. WordPress: Codex › add_theme_support › HTML5
[110] vgl. WordPress: Codex › add_theme_support › Post Thumbnails

5.7 Blog-Übersicht

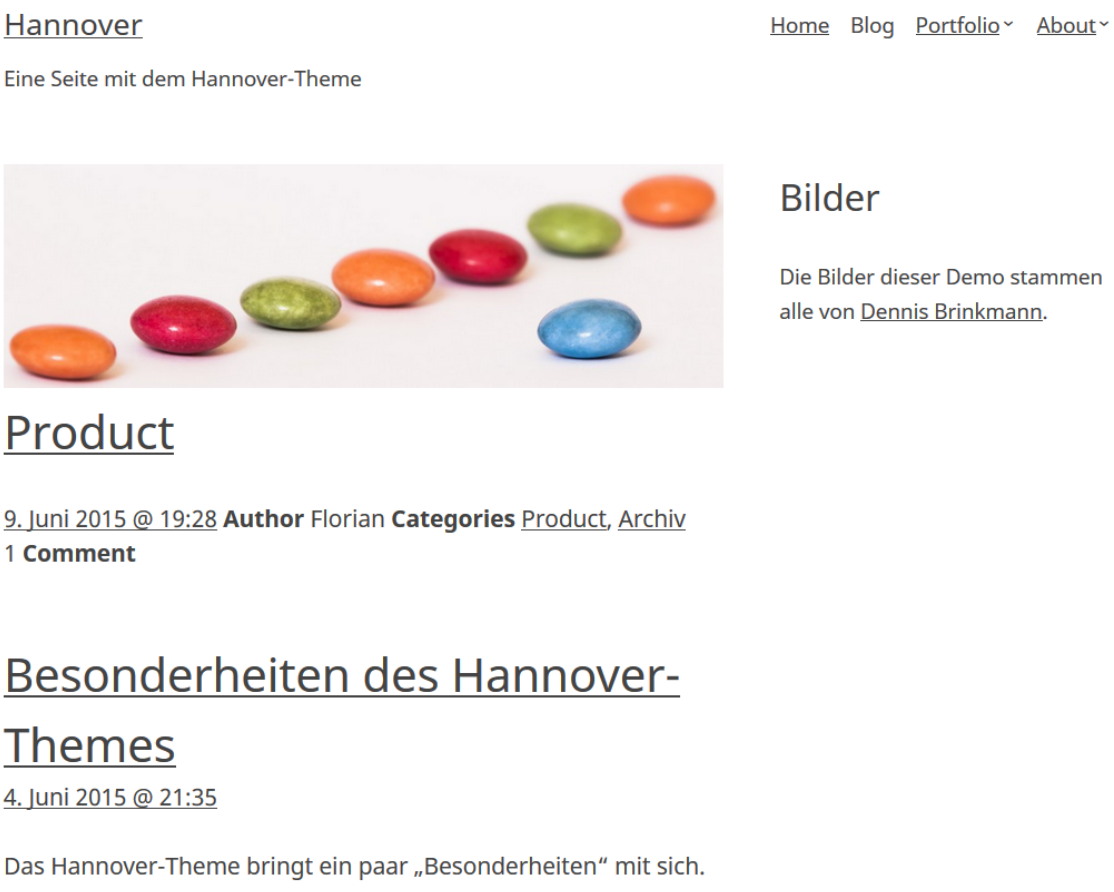


Abbildung 5: Screenshot von der Blog-Übersicht.

Für die Blog-Übersicht ist die `index.php` zuständig, die wie die `style.css` in jedem Fall notwendig ist.[111] In Abbildung 5 ist zu sehen, wie die Blog-Übersicht des Hannover-Themes aussehen wird: Links werden die Beiträge dargestellt, rechts eine Sidebar.

```
<?php get_header(); ?>
   <main role="main">
      <?php if ( have_posts() ) {
         if ( is_home() && ! is_front_page() ) { ?>
            <header>
               <h1 class="page-title screen-reader-text">
                  <?php single_post_title(); ?>
               </h1>
            </header>
         <?php }
         while ( have_posts() ) {
            the_post();
            get_template_part(
               'template-parts/content', get_post_format()
            );
         }
```

[111] vgl. WordPress: Theme Handbook › Template Files Section › Post Template Files › index.php

```
        }
        the_posts_pagination( array( 'type' => 'list' ) ); ?>
    </main>
<?php get_sidebar();
get_footer();
```

Listing 14: Blog-Übersicht in der index.php

In Listing 14 ist die `index.php` abgebildet, in der zuerst mit der `get_header()`-Funktion die `header.php` eingebunden wird.[112] Innerhalb des `main`-Elements, das den Hauptinhalt enthält, wird zunächst mit `is_home()` und `! is_front_page()` überprüft, ob die Blog-Übersicht auf einer anderen Seite als der Startseite angezeigt wird.[113, 114] In diesem Fall wird innerhalb eines `header`- und `h1`-Elements mit `single_post_title()` der Titel der Seite ausgegeben.[115]

Anschließend werden die Beiträge ausgegeben. Dafür wird zuerst mit `have_posts()` geprüft, ob Beiträge aus der aktuellen Anfrage zurückgegeben wurden.[116] Ist das der Fall, wird mit einer `while`-Schleife jeder Beitrag durchlaufen. Innerhalb der Schleife werden mit `the_post()` die Daten des aktuellen Beitrags zugänglich gemacht und mit `get_template_part()` die Datei eingebunden, die ihn in der Übersicht anzeigt.[117, 118] Als Parameter wird zuerst ein Slug übergeben, der zweite Parameter ist ein optionaler Name der spezialisierten Template-Datei.

In diesem Fall mit `template-parts/content` und `get_post_format()` als Parameter sucht WordPress Theme-Dateien abhängig von dem Post-Format – mit `get_post_format()` wird der Slug des verwendeten Post-Formats ausgegeben.[119] Wenn beispielsweise ein Beitrag mit dem Post-Format *Galerie* angezeigt wird, sucht WordPress nach einer Datei namens `content-gallery.php` in dem Ordner `template-parts`. Gibt es diese Datei nicht, wird die `content.php` geladen, ebenso wie für Beiträge ohne spezielles Post-Format. An dieser Stelle können somit einfach verschiedene Ausgaben für die unterschiedlichen Post-Formate erreicht werden.

[112] vgl. WordPress: Code Reference › Functions › get_header
[113] vgl. WordPress: Code Reference › Functions › is_home
[114] vgl. WordPress: Code Reference › Functions › is_front_page
[115] vgl. WordPress: Code Reference › Functions › single_post_title
[116] vgl. WordPress: Code Reference › Functions › have_posts
[117] vgl. WordPress: Code Reference › Functions › the_post
[118] vgl. WordPress: Code Reference › Functions › get_template_part
[119] vgl. WordPress: Code Reference › Functions › get_post_format

Mit `the_posts_pagination()` wird nach Anzeige der Beiträge gegebenenfalls eine Pagination ausgegeben, über die ältere Beiträge erreicht werden können.[120] `get_sidebar()` und `get_footer()` binden abschließend die Dateien `sidebar.php` und `footer.php` ein.[121, 122]

5.8 Vorschau eines normalen Beitrags

Wie in Kapitel 5.7 beschrieben, wird zur Darstellung normaler Beiträge ohne Post-Format die `content.php` genutzt. Dabei gibt es zwei Unterscheidungen: Mit Beitragsbild oder ohne.

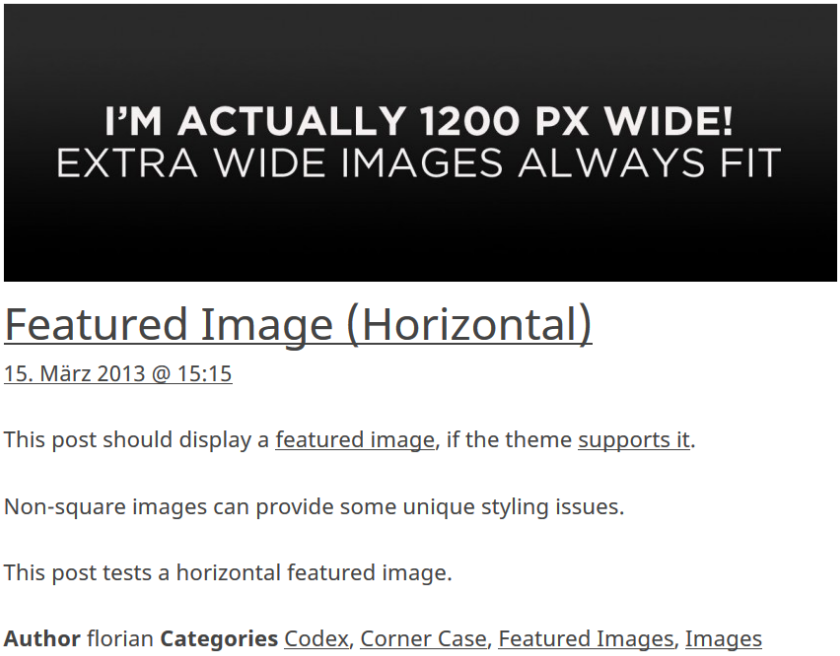


Abbildung 6: Ansicht eines normalen Beitrags mit Beitragsbild in der Übersicht.

Abbildung 6 zeigt den Entwurf eines Beitrags mit Beitragsbild. Unter dem Beitragsbild wird der Titel angezeigt, gefolgt vom Veröffentlichungsdatum. Danach steht der Inhalt des Beitrags und ein Link zu der Einzelansicht. Abschließend werden die Kategorien, Tags, der Autor sowie Trackback- und Kommentar-Anzahl dargestellt. Wenn kein Beitragsbild gesetzt ist, fällt es einfach weg.

[120] vgl. WordPress: Code Reference › Functions › the_posts_pagination
[121] vgl. WordPress: Code Reference › Functions › get_footer
[122] vgl. WordPress: Code Reference › Functions › get_sidebar

Da einige dieser Elemente voraussichtlich später in anderen Dateien wiederverwendet werden, werden sie in die `functions.php` ausgelagert. In Listing 15 ist die komplette `content.php` des Hannover-Themes abgebildet.

```
<article <?php post_class(); ?> id="post-<?php the_ID(); ?>">
   <header class="entry-header">
      <?php the_post_thumbnail( 'large' );
      hannover_the_title( 'h2', true ); ?>
      <a href="<?php the_permalink(); ?>" class="entry-date">
         <?php hannover_the_date(); ?>
      </a>
   </header>
   <div class="entry-content">
      <?php hannover_the_content(); ?>
   </div>
   <footer>
      <p><?php hannover_entry_meta() ?></p>
   </footer>
</article>
```

Listing 15: content.php

Um jeden Beitrag mit hilfreichen Klassen anzureichern, wird die Funktion `post_class()` verwendet.[123] Mit Hilfe von `the_ID()`, womit die ID des aktuellen Beitrags ausgegeben wird, wird darüber hinaus noch eine eindeutige ID für jeden Beitrag erstellt. In das `header`-Element sollen das Beitragsbild, der Titel sowie das Veröffentlichungsdatum eingefügt werden. Um das Beitragsbild auszugeben, wird die `the_post_thumbnail()`-Funktion verwendet und als Parameter die Größe angegeben, die angezeigt werden soll.[124]

5.8.1 Titel ausgeben

Mit der Funktion `hannover_the_title()`, die in Listing 16 abgebildet ist, wird der Titel des Beitrags ausgegeben. Als Parameter wird die Überschriften-Hierarchie angegeben sowie ein boolescher Wert, ob die Überschrift verlinkt sein soll oder nicht.

```
function hannover_the_title( $heading, $link ) {
   if ( $link ) {
      the_title( sprintf(
         '<%1$s class="entry-title"><a href="%2$s" rel="bookmark">',
         $heading, esc_url( get_permalink() )
      ), sprintf( '</a></%s>', $heading ) );
   } else {
      the_title( sprintf(
         '<%1$s class="entry-title">',
```

[123] vgl. WordPress: Code Reference › Functions › post_class
[124] vgl. WordPress: Code Reference › Functions › the_post_thumbnail

```
            $heading, esc_url( get_permalink() )
        ), sprintf( '</%s>', $heading ) );
    }
}
```

Listing 16: hannover_the_title()-Funktion in der functions.php

In der Funktion wird lediglich die `the_title()`-Funktion ausgeführt und angepasst. `the_title()` erwartet als ersten Parameter den Code, der vor dem Titel eingefügt werden soll und als zweiten den für danach.[125] Um möglichst einfach die Überschriftenebene und den Link in den HTML-Code einzusetzen, wird in beiden Fällen die Funktion `sprintf()` genutzt.

Als ersten Parameter erwartet die Funktion den String, der formatiert werden soll. Danach kommen die Werte, die die Platzhalter ersetzen sollen.[126] Der erste Platzhalter stellt dabei die Überschriftenebene dar – dieser Platzhalter wird durch die Variable `$heading` ersetzt. Der zweite Platzhalter wird durch die URL des Beitrags ersetzt und erzeugt so einen funktionsfähigen Link. Hinter dem Titel wird das schließende Link-Tag sowie die korrekte Überschriftenebene ausgegeben. Wenn als boolescher Wert `false` angegeben wurde, wird die Verlinkung einfach weggelassen.

5.8.2 Datum und Uhrzeit ausgeben

Wie in Listing 15 zu sehen, wird nach dem Titel ein Link auf den Beitrag ausgegeben, der die `hannover_the_date()`-Funktion umschließt. Auch diese Funktion gehört in die `functions.php` und ist in Listing 17 abgebildet.

```
function hannover_the_date() {
    /* translators: 1=date, 2=time */
    printf( __( '%1$s @ %2$s', 'hannover' ),
        get_the_date(),
        get_the_time()
    );
}
```

Listing 17: hannover_the_date()-Funktion in der functions.php

Ähnlich wie bei `hannover_the_title()` wird in dieser Funktion mit dem Ersetzen von Platzhaltern gearbeitet. Diesmal wird jedoch die `printf()`-Funktion verwendet, da das

[125] vgl. WordPress: Code Reference › Functions › the_title
[126] vgl. PHP Group: sprintf

Ergebnis nicht von einer anderen PHP-Funktion genutzt, sondern gleich ausgegeben werden soll.[127]

Das Standardformat soll dem Muster `Datum @ Uhrzeit` entsprechen – Übersetzer haben die Möglichkeit, dieses Muster anzupassen, weshalb in einem Übersetzer-Kommentar angegeben wird, welcher Platzhalter wodurch ersetzt wird. Mit `get_the_date()` und `get_the_time()` werden die beiden Platzhalter durch das Datum beziehungsweise die Uhrzeit ersetzt. Um dasselbe Datum- und Uhrzeit-Format anzuzeigen, wie der Nutzer im Backend eingestellt hat, wird kein Parameter an die beiden Funktionen übergeben.[128, 129]

5.8.3 Inhalt mit angepasstem Weiterlesen-Link

```php
function hannover_the_content() {
   /* translators: text for the more tag. s= title */
   the_content(
      sprintf(
         __( 'Continue reading “%s”', 'hannover' ),
         esc_html( get_the_title() )
      )
   );
}
```

Listing 18: hannover_the_content()-Funktion in der functions.php

Um den Inhalt des Beitrags inklusive Weiterlesen-Link auszugeben, würde theoretisch der `the_content()`-Aufruf ohne Parameter ausreichen.[130] Der Standard-Weiterlesen-Link sieht allerdings so aus: `(mehr …)`. Für Screen-Reader-Nutzer ist das nicht besonders hilfreich, da der Kontext fehlt. Im Theme-Review-Handbuch wird im Bereich *Accessibility* zu diesem Thema geschrieben, dass der Titel des Beitrags mit in den Weiterlesen-Link integriert werden sollte.[131]

Aus diesem Grund wird eine eigene Funktion erstellt, um `the_content()` mit verändertem Weiterlesen-Link auszugeben, die in Listing 18 abgebildet ist. An die `the_content()`-Funktion wird mittels `sprintf()` ein Muster mit Platzhalter übergeben, dessen Platzhalter `%s` durch den Titel des Beitrags ersetzt wird, der über

[127] vgl. PHP Group: printf
[128] vgl. WordPress: Code Reference › Functions › get_the_date
[129] vgl. WordPress: Code Reference › Functions › get_the_time
[130] vgl. WordPress: Code Reference › Functions › the_content
[131] vgl. WordPress: Theme Review Handbook › Accessibility › Required › Link Text

`get_the_title()` ermittelt wird.[132] Daraus entsteht ein Weiterlesen-Link des Formats *Continue reading "Beitragstitel"*.

5.8.4 Meta-Daten im Footer des Beitrags

Im Footer eines normalen Beitrags sollen der Autor, die Kategorien, Tags sowie Anzahl von Kommentaren und Trackbacks angezeigt werden. Die Funktion, die den Footer anzeigt, heißt `hannover_entry_meta()` und wird hier der Länge wegen stückweise abgebildet.

Den Autoren ausgeben

```
function hannover_entry_meta() { ?>
   <span class="author"><?php
      /* translators: name of the author in entry footer. s=author name*/
      printf( __( 'Author %s', 'hannover' ),
   '<span>' . get_the_author() . '</span>' ) ?></span>
```

Listing 19: Ausgabe des Autors in hannover_entry_meta() in der functions.php

Listing 19 zeigt den Anfang der Funktion, mit dem der Autor des Beitrags ausgegeben wird. Da es in jedem Fall einen Autor gibt, muss hier nicht geprüft werden, ob ein Autor vorliegt. Innerhalb des `span`-Elements wird in dem Formatierungs-String die Formatierungs-Anweisung durch den Autor ersetzt, der mit der `get_the_author()`-Funktion ermittelt wird.[133]

Die Kategorien ausgeben

```
<?php if ( get_the_category() ) { ?>
   <span class="categories"><?php /* translators: Label for category list
         in entry footer. s=categories */
      printf( _n(
         'Category %s',
         'Categories %s',
         count( get_the_category() ),
         'hannover'
      ), /* translators: term delimiter */
         '<span>' . get_the_category_list( __( ', ', 'hannover' ) )
         . '</span>' ) ?></span>
<?php }
```

Listing 20: Ausgabe der Kategorien in hannover_entry_meta() in der functions.php

[132] vgl. WordPress: Code Reference › Functions › get_the_title
[133] vgl. WordPress: Code Reference › Functions › get_the_author

In Listing 20 ist dargestellt, wie die Kategorien des Beitrags ausgegeben werden. Standardmäßig ist es nicht möglich, einen Beitrag ohne Kategorie zu veröffentlichen. Um sicherzugehen, wird mit `get_the_category()` dennoch geprüft, ob eine Kategorie übergeben wurde.[134]

Für den Fall, dass ein Beitrag nur einer einzelnen Kategorie zugewiesen wurde, soll die Beschriftung *Category* anstatt *Categories* ausgegeben werden. Für diese Singular-Plural-Unterscheidung gibt es eigene Übersetzungsfunktionen – eine davon ist `_n()`. Als ersten Parameter erwartet die Funktion den Singular-String, gefolgt vom Plural. Der dritte Parameter ist eine Zahl, anhand derer die Funktion den Singular oder Plural ausgibt: Wenn die Zahl größer als eins ist, wird der Plural angezeigt, ansonsten der Singular. Der vierte Parameter ist die Text-Domain.[135]

Um die Anzahl von Kategorien zu ermitteln, wird das Ergebnis-Array aus dem `get_the_category()`-Aufruf mit der `count()`-Funktion gezählt.[136] Um die Kategorie-Liste auszugeben, gibt es die `get_the_category_list()`-Funktion, der als erster Parameter das Trennzeichen übergeben wird.[137]

Die Tags ausgeben

Die Tags werden mit einem ähnlichen System angezeigt, wie die Kategorien.

```
if ( get_the_tags() ) { ?>
   <span class="tags"><?php
      /* translators: Label for tag list in entry footer. s=tags */
      printf( _n(
         'Tag %s',
         'Tags %s',
         count( get_the_tags() ),
         'hannover'
      ), /* translators: term delimiter */
         '<span>' . get_the_tag_list( '', __( ', ', 'hannover' ) )
         . '</span>' ) ?></span>
<?php }
```

Listing 21: Anzeige der Tags in der hannover_entry_meta() in der functions.php

Zunächst wird in Listing 21 mit der `get_the_tags()`-Funktion überprüft, ob dem Beitrag Tags zugewiesen wurden.[138] Ist das der Fall, wird nach demselben Schema wie bei den Kategorien der Singular oder Plural des Begriffs *Tag* ausgegeben. Um die Tag-Liste zu

[134] vgl. WordPress: Code Reference › Functions › get_the_category
[135] vgl. WordPress: Code Reference › Functions › _n
[136] vgl. PHP Group: count
[137] vgl. WordPress: Code Reference › Functions › get_the_category_list
[138] vgl. WordPress: Code Reference › Functions › get_the_tags

ermitteln, stellt WordPress `get_the_tag_list()` bereit. Vor dem Trennzeichen erwartet diese Funktion als ersten Parameter einen String, der vor der Liste angezeigt werden soll.[139] Da die Beschriftung bereits innerhalb der Übersetzungsfunktion `_n()` geregelt ist, wird ein leerer String übergeben.

Kommentare und Trackbacks trennen

Der besseren Übersichtlichkeit halber sollen Kommentare und Trackbacks nicht vermischt dargestellt werden, wie WordPress es normalerweise tun würde. Um die Kommentare und Trackbacks zu trennen, gibt es die `separate_comments()`-Funktion, die in Listing 22 zum Einsatz kommt.[140] Der Code ist zum großen Teil aus der `comments_template()`-Funktion entliehen, die die Kommentar-Arten ebenfalls getrennt zurückgeben kann.[141] Zunächst werden das `$wp_query`-Objekt für Abfragen und das `$post`-Objekt für die Daten des aktuellen Beitrags zugänglich gemacht. Anschließend wird ein Array an Argumenten für `get_comments()` vorbereitet. Die Reaktionen sollen absteigend nach dem Datum sortiert werden und nur abgefragt werden, wenn sie den Status `approve` besitzen, also freigeschaltet sind. Außerdem müssen es Reaktionen zu dem Beitrag sein, der gerade angezeigt wird.

Diese Argumente werden an die `get_comments()`-Funktion übergeben, die die Reaktionen als Array zurückgibt – hier sind die Reaktionen aber noch gemischt und nicht nach Typ getrennt.[142] Dafür wird im nächsten Schritt das Reaktions-Array an die `separate_comments()`-Funktion übergeben und das Ergebnis in der Eigenschaft `comments_by_type` des `$wp_query`-Objekts gespeichert. Auf diesen Wert wird in der nächsten Zeile referenziert und das Ergebnis in `$comments_by_type` gespeichert, die abschließend zurückgegeben wird.

```
function hannover_get_comments_by_type() {
   global $wp_query, $post;
   $comment_args               = array(
      'order'   => 'ASC',
      'orderby' => 'comment_date_gmt',
      'status'  => 'approve',
      'post_id' => $post->ID,
   );
   $comments                   = get_comments( $comment_args );
   $wp_query->comments_by_type = separate_comments( $comments );
   $comments_by_type           = &$wp_query->comments_by_type;
```

[139] vgl. WordPress: Code Reference › Functions › get_the_tag_list
[140] vgl. WordPress: Code Reference › Functions › separate_comments
[141] vgl. WordPress: Code Reference › Functions › comments_template
[142] vgl. WordPress: Code Reference › Functions › get_comments

```
    return $comments_by_type;
}
```

Listing 22: Funktion zur Zählung von Kommentaren und Trackbacks in der functions.php

Anzahl der Kommentare ausgeben

Um die Anzahl der Kommentare in dem Fußbereich des Beitrags auszugeben, wird der Code in Listing 23 benötigt. Zunächst wird mit dem Aufruf der Funktion `hannover_get_comments_by_type()` das Array mit den Reaktionen bereitgestellt, die nach Typ gruppiert sind. Im Anschluss wird überprüft, ob das Unter-Array mit dem Schlüssel `comment` existiert, in dem die Kommentare vorliegen. Ist das der Fall, werden mit der `count()`-Funktion die Kommentare gezählt.

```
$comments_by_type = hannover_get_comments_by_type();
if ( $comments_by_type['comment'] ) {
    $comment_number = count( $comments_by_type['comment'] ); ?>
    <span class="comments"><?php /* translators: Label for comment number
        in entry footer. s=comment number */
        printf( _n(
            '%s Comment',
            '%s Comments',
            $comment_number,
            'hannover'
            ), '<span>' . number_format_i18n( $comment_number )
                . '</span>' ) ?></span>
<?php }
```

Listing 23: Anzeige der Kommantarzahl in hannover_entry_meta() in der functions.php

Danach wird wieder ähnlich verfahren wie bei der Anzeige von Kategorien und Tags. Es wird nach Singular und Plural unterschieden und als Zahl für die `_n()`-Funktion die `$comment_number`-Variable genutzt. Um die Anzahl der Kommentare im richtigen Format für die eingestellte Sprache anzuzeigen, wird das Ergebnis an `number_format_i18n()` übergeben.[143]

Anzahl der Trackbacks ausgeben

```
    if ( $comments_by_type['pings'] ) {
    $trackback_number = count( $comments_by_type['pings'] ); ?>
        <span class="trackbacks"><?php /* translators: Label for trackback
        number in entry footer. s=trackback number */
        printf( _n(
            '%s Trackback',
            '%s Trackbacks',
            $trackback_number,
            'hannover'
        ), '<span>' . number_format_i18n( $trackback_number ) . '</span>' )
```

[143] vgl. WordPress: Code Reference › Functions › number_format_i18n

```
      ?></span>
   <?php };
}
```

Listing 24: Ausgabe der Trackback-Zahl in hannover_entry_meta() in der functions.php

Wie in Listing 24 ersichtlich, ähnelt die Ausgabe der Trackback-Anzahl sehr dem Code aus Listing 23. Statt auf den Array-Abschnitt mit dem Schlüssel `comment` wird hier auf `pings` geprüft. Entsprechend wird auch dieser Array-Zweig im Anschluss gezählt und das Ergebnis in `$trackback_number` gespeichert.

5.9 Die Sidebar

Auf der Übersichtsseite des Blogs und der Einzelansicht soll rechts eine Sidebar angezeigt werden. In der `index.php` wurde die `sidebar.php` bereits mit der Funktion `get_sidebar()` eingebunden – jetzt muss sie erstellt werden.

In Listing 25 ist die gesamte `sidebar.php` abgebildet. Um das `aside`-Element und die Überschrift für Screen-Reader-Nutzer nur dann auszugeben, wenn der Sidebar auch wirklich Widgets zugewiesen sind, wird mit `is_active_sidebar()` dieser Umstand überprüft. Als Parameter wird der Funktion die eindeutige Bezeichnung der Sidebar übergeben, die angezeigt werden soll.[144] Nach der Überschrift für Screen-Reader-Nutzer wird mit `dynamic_sidebar()` die Sidebar ausgegeben – als Parameter muss auch hier wieder der Bezeichner übergeben werden.

```
<?php if ( is_active_sidebar( 'sidebar-1' ) ) { ?>
   <aside id="secondary" role="complementary">
      <h2 class="screen-reader-text">
         <?php /* translators: screen reader text for the sidebar */
         _e( 'Sidebar', 'hannover' ) ?>
      </h2>
      <?php dynamic_sidebar( 'sidebar-1' ); ?>
   </aside>
<?php }
```

Listing 25: sidebar.php

Damit der Theme-Nutzer die Sidebar im Backend mit Widgets befüllen kann, muss sie registriert werden. In Listing 26 ist die entsprechende Funktion abgebildet.

```
function hannover_register_sidebars() {
   register_sidebar( array(
      'name'          => __( 'Main Sidebar', 'hannover' ),
```

[144] vgl. WordPress: Function Reference › Functions › is_active_sidebar

```
        'id'            => 'sidebar-1',
        'description'   => __( 'Widgets in this area will be shown
           on all normal posts and pages.', 'hannover' ),
        'before_widget' => '<div id="%1$s" class="widget %2$s">',
        'after_widget'  => '</div>',
        'before_title'  => '<h3 class="widget-title">',
        'after_title'   => '</h3>',
    ) );
}

add_action( 'widgets_init', 'hannover_register_sidebars' );
```

Listing 26: Registrierung der Sidebar in der functions.php

Um eine Sidebar in WordPress zu registrieren, gibt es `register_sidebar()`.[145] Der Funktion kann ein Options-Array übergeben werden, um die Standard-Werte zu überschreiben. Der Wert zum Schlüssel `name` wird im Backend als Name der Sidebar angezeigt – `__()` ist an dieser Stelle eine weitere Übersetzungsfunktion, der kein Kontext übergeben wird und die den String nicht sofort ausgibt.[146] `id` ist der Wert, der als Parameter an `is_active_sidebar()` übergeben wird und muss eindeutig sein. Die `description` wird wie der Name der Sidebar auch im Backend angezeigt. Um Code vor und nach dem Widget auszugeben, werden `before_widget` und `after_widget` bereitgestellt – der erste Platzhalter in `before_widget` wird durch einen eindeutigen Bezeichner ersetzt, der aus der Art des Widgets und einer Zahl besteht. Der zweite Platzhalter wird durch die Art des Widgets ersetzt, beispielsweise `widget_recent_entries` für das Widget, das die neuesten Beiträge anzeigt.

Mit den Werten für `before_title` und `after_title` kann HTML-Code angegeben werden, der vor und hinter dem Titel des Widgets ausgegeben werden soll.

5.10 Der Footer des Themes

Zu diesem frühen Stadium der Entwicklung kommt in die `footer.php` nur das Notwendige. Später finden hier noch ein Menü für Social-Media-Icons, ein normales Menü sowie ein Link zum Theme-Autor Platz. In Listing 27 ist die gesamte Datei abgebildet. Das schließende `div`-Tag gehört zu dem `div`-Element mit der ID `content`, das in der `header.php` geöffnet wurde.

[145] vgl. WordPress: Code Reference › Functions › register_sidebar
[146] vgl. WordPress: Code Reference › Functions › __

Im Anschluss muss vor dem schließenden `body`- und `html`-Tag noch die Funktion `wp_footer()` aufgerufen werden, mit der Skripte und Stylesheets eingebunden werden, die im Fußbereich der Website platziert werden sollen.[147]

```
</div>
<?php wp_footer(); ?>
</body>
</html>
```

Listing 27: footer.php

5.11 Archiv-Ansicht

Um sämtliche Archiv-Arten etwas anders darzustellen als mit der `index.php`, ohne für jede Variante eine eigene Datei anzulegen, muss die `archive.php` erstellt werden. Der einzig größere Unterschied zu der `index.php` ist, dass der Titel und die Beschreibung des Archivs angezeigt werden.

```
<?php get_header(); ?>
   <main role="main">
      <?php if ( have_posts() ) { ?>
         <header class="page-header">
            <?php the_archive_title(
                  '<h1 class="archive-title">', '</h1>'
            );
            the_archive_description(
                  '<div class="taxonomy-description">', '</div>'
            ); ?>
         </header>
         <?php while ( have_posts() ) {
            the_post();
            get_template_part( 'content', get_post_format() );
         }
      }
      the_posts_pagination( array( 'type' => 'list' ) ); ?>
   </main>
<?php get_sidebar();
get_footer();
```

Listing 28: archive.php

In Listing 28 ist die `archive.php` abgebildet, die, wie bereits erwähnt, der in Kapitel 5.7 behandelten `index.php` recht ähnlich ist. Statt gegebenenfalls den Titel der Seite auszugeben, wird in der `archive.php` innerhalb des `header`-Elements mit `the_archive_title()` der Titel des Archives ausgegeben.[148] Im Fall einer Kategorie

[147] vgl. WordPress: Code Reference › Functions › wp_footer
[148] vgl. WordPress: Code Reference › Functions › the_archive_title

wäre das beispielsweise *Kategorie: Porträt*. Als ersten Parameter erwartet die Funktion Markup, das vor dem Titel angezeigt werden soll – als zweiten Parameter wird das Markup hinter dem Titel übergeben.

Zusätzlich zum Titel kann der Theme-Nutzer für Kategorien und Tags noch optional eine Beschreibung vergeben. Diese Beschreibung kann mit der Funktion `the_archive_description()` angezeigt werden.[149] Auch dieser Funktion kann über die Parameter jeweils ein String für die Anzeige vor und nach der Beschreibung übergeben werden.

5.12 Suchergebnis-Template

Für die Anzeige der Suchergebnisse ist die Datei `search.php` verantwortlich. Die Auflistung der Beiträge gleicht der im Blog oder anderen Archiv-Ansichten, es wird jedoch vorher noch einmal der Suchbegriff aufgegriffen und im Fall von keinem Ergebnis ein Hinweis sowie ein Suchformular angezeigt.

```
<?php get_header(); ?>
   <main role="main">
      <?php if ( have_posts() ) { ?>
         <header class="page-header">
            <h1><?php printf( __( 'Search Results for: %s', 'hannover' ),
               esc_html( get_search_query() ) ); ?></h1>
         </header>
         <?php while ( have_posts() ) {
            the_post();
            get_template_part( 'template-parts/content',
               get_post_format() );
         }
      } else { ?>
         <header class="page-header">
            <h1><?php printf( __( 'Nothing found for “%s”', 'hannover' ),
               esc_html( get_search_query() ) ); ?></h1>
         </header>
         <?php get_search_form();
      }
      the_posts_pagination( array( 'type' => 'list' ) ); ?>
   </main>
<?php get_sidebar();
get_footer();
```

Listing 29: search.php

[149] vgl. WordPress: Code Reference › Functions › the_archive_description

Listing 29 zeigt den Code für die Datei `search.php`. Sie ähnelt der `archive.php` aus Listing 28 auf Seite 44 mit einigen Neuerungen. Innerhalb des `h1`-Elements wird eine Nachricht für den Nutzer angezeigt, die besagt `Search Results for: Suchbegriff`. Der Suchbegriff kann mit der `get_search_query()`-Funktion ermittelt werden und wird durch `printf()` in den String eingesetzt.[150] Anschließend wird die Schleife durchlaufen und die Beiträge werden ausgegeben.

Falls keine Beiträge vorhanden sind, wird als `h1`-Headline der Hinweis ausgegeben, dass für den Begriff oder die Begriffe nichts gefunden wurde. Anschließend wird `get_search_form()` aufgerufen, die ein Suchformular einbindet.[151] Die Funktion sucht nach einer Suchformular-Datei in dem Theme oder gegebenenfalls dem Eltern-Theme – stellt das Theme kein Formular bereit, wird das Standard-Formular eingebunden.

5.13 Einzelansicht von Seiten

Um die Einzelansicht einer Seite darzustellen, gibt es in WordPress die Datei `page.php`, die in Listing 30 abgebildet ist.

```
<?php get_header(); ?>
   <main role="main">
      <?php while ( have_posts() ) {
         the_post();
         get_template_part( 'template-parts/content', 'page' );
         if ( comments_open() || get_comments_number() ) {
            comments_template( '', true );
         }
      } ?>
   </main>
<?php get_sidebar();
get_footer();
```

Listing 30: page.php

Zuerst muss mit `get_header()` die `header.php` eingebunden werden.[152] Innerhalb der Schleife wird im Anschluss die `content-page.php` aus dem `template-parts`-Ordner eingebunden. Abschließend wird innerhalb der `while`-Schleife mit `comments_open()` und `get_comments_number()` geprüft, ob Kommentare erlaubt und/oder vorhanden sind.[153, 154] Wenn die Bedingung erfüllt ist, wird mit `comments_template()` die

[150] vgl. WordPress: Code Reference › Functions › get_search_query
[151] vgl. WordPress: Code Reference › Functions › get_search_form
[152] vgl. WordPress: Code Reference › Functions › get_header
[153] vgl. WordPress: Code Reference › Functions › get_comments_number

`comments.php` des Themes eingebunden.[155] Da die Kommentare und Trackbacks getrennt aufgeführt werden sollen, wird als zweiter Parameter der Wert `true` übergeben. Abschließend werden Sidebar und Footer angezeigt.

```
<article <?php post_class(); ?> id="post-<?php the_ID(); ?>">
   <header class="entry-header">
      <?php hannover_the_title( 'h1', false ); ?>
   </header>
   <div class="entry-content">
      <?php hannover_the_content();
      wp_link_pages(); ?>
   </div>
</article>
```

Listing 31: content-page.php

In Listing 31 ist die `content-page.php` des Themes dargestellt. Sie ähnelt der `content.php` aus Kapitel 5.8, mit dem Unterschied, dass der Titel innerhalb einer `h1`-Überschrift ausgegeben und nicht verlinkt wird. Außerdem werden weder das Datum noch die Meta-Informationen ausgegeben, da diese auf einer statischen Seite wie etwa dem Impressum irrelevant sind. Darüber hinaus wird zusätzlich noch die `wp_link_pages()`-Funktion aufgerufen, um gegebenenfalls die Pagination auszugeben, die mit dem `<!--nextpage-->`-Kommentar in Beiträge und Seiten eingefügt werden kann.[156]

5.14 Einzelansicht von Beiträgen

Für die Einzelansicht eines Beitrags gibt es in WordPress die `single.php`, die beim Hannover-Theme in großen Teilen der `page.php` ähnelt.

```
<?php get_header(); ?>
   <main role="main">
      <?php while ( have_posts() ) {
         the_post();
         get_template_part( 'template-parts/content-single',
            get_post_format() );
         the_post_navigation();
         if ( comments_open() || get_comments_number() ) {
            comments_template( '', true );
         }
      } ?>
   </main>
<?php get_sidebar();
get_footer();
```

[154] vgl. WordPress: Code Reference › Functions › comments_open
[155] vgl. WordPress: Code Reference › Functions › comments_template
[156] vgl. WordPress: Code Reference › Functions › wp_link_pages

Listing 32: single.php

Listing 32 zeigt die `single.php`, deren Unterschiede zur `page.php` darin bestehen, dass sie über die `get_template_parts()`-Funktion die `content-single.php` einbindet oder, soweit vorhanden, eine entsprechende Datei mit einem Post-Format-Suffix, wie beispielsweise die `content-single-gallery.php`, sodass diese Ansichten anders dargestellt werden können. Darüber hinaus wird mit der `the_post_navigation()`-Funktion vor dem Kommentarbereich ein Link auf den vorherigen und nächsten Beitrag angezeigt.[157]

```
<article <?php post_class(); ?> id="post-<?php the_ID(); ?>">
   <header class="entry-header">
      <?php the_post_thumbnail( 'large' );
      hannover_the_title( 'h1', false ); ?>
      <span><?php hannover_the_date(); ?></span>
   </header>
   <div class="entry-content">
      <?php hannover_the_content();
      wp_link_pages(); ?>
   </div>
   <footer>
      <p><?php hannover_entry_meta() ?></p>
   </footer>
</article>
```

Listing 33: content-single.php

Die `content-single.php` aus Listing 33 unterscheidet sich nur im Detail von der `content.php`. Die Überschrift wird nicht verlinkt und als `h1` ausgegeben, auch das Datum wird nicht verlinkt – außerdem wird wie in der `content-page.php` die Funktion `wp_link_pages()` aufgerufen.

5.15 Kommentarbereich

Über die `comments_template()`-Funktion wird die `comments.php` des Themes aufgerufen, oder – falls keine vom Theme bereitgestellt wird – die Datei des aktuellen Standard-Themes. Zuerst muss, wie in Listing 34 zu sehen, mit `post_password_required()` geprüft werden, ob der Beitrag mit einem Passwort geschützt ist und ob das richtige Passwort *nicht* vorliegt.[158] Ist das der Fall, wird der Durchlauf der Datei abgebrochen.

[157] vgl. WordPress: Code Reference › Functions › the_post_navigation
[158] vgl. WordPress: Code Reference › Functions › post_password_required

```php
<?php if ( post_password_required() ) {
   return;
} ?>
```

Listing 34: Prüfung auf Passwortschutz in der comments.php

Die Kommentare und Trackbacks sollen getrennt voneinander angezeigt werden. Durch die Übergabe des booleschen Wertes `true` als zweiten Parameter an `comments_template()` steht die Variable `$comments_by_type` mit demselben Inhalt zur Verfügung, wie aus Listing 22 bekannt.

```php
<div id="comments" class="comments-area">
   <?php if ( have_comments() ) {
      if ( ! empty( $comments_by_type['comment'] ) ) {
         $comment_number = count( $comments_by_type['comment'] ); ?>
         <h2 id="comments-title">
            <?php /* translators: Title for comment list.
            1=comment number, 2=post title */
            printf( _n(
               '%1$s Comment on "%2$s"',
               '%1$s Comments on "%2$s"',
               $comment_number,
               'hannover'
            ), number_format_i18n( $comment_number ),
               get_the_title() ) ?>
         </h2>
```

Listing 35: Anfang der Kommentar-Anzeige in der comments.php

In Listing 35 ist der nächste Teil der `comments.php` abgebildet. Nach Öffnen des `div`-Tags, das den gesamten Bereich mit Kommentaren, Trackbacks und Kommentarfeld umschließen wird, prüft `if ( have_comments() )`, ob Reaktionen irgendwelchen Typs ausgegeben werden müssen.[159] Ist das der Fall, wird geprüft, ob Kommentare vorhanden sind und anschließend die Anzahl der Kommentare als Überschrift ausgegeben. Dieses Vorgehen ist weitgehend aus Listing 23 auf Seite 41 bekannt. Der größte Unterschied ist an dieser Stelle, dass über einen zweiten Platzhalter noch der Titel des Beitrags in die Überschrift eingefügt wird.

```php
   <ol class="commentlist">
      <?php wp_list_comments( array(
         'callback' => 'hannover_comments',
         'style'    => 'ol',
         'type'     => 'comment'
      ) ); ?>
   </ol>
<?php }
```

Listing 36: Ausgabe der Kommentare in der comments.php

[159] vgl. WordPress: Code Reference › Functions › have_comments

Listing 36 sorgt für die Ausgabe der Kommentarliste. Da `wp_list_comments()` kein umschließendes `ol`-Element zurückgibt, wird es händisch eingetragen. Als Options-Array wird der Funktion die Callback-Funktion übergeben, die die Darstellung der Kommentare steuert. Wenn hier kein Wert angegeben wird, kümmert sich WordPress intern um die Anzeige der Liste. Als Listenstil soll eine `ol`-Liste angezeigt werden und die angezeigten Elemente müssen vom Typ `comment` sein.[160] Die Callback-Funktion wird, genau wie die zur Anzeige von Trackbacks, später in diesem Kapitel erläutert.

```
if ( ! empty( $comments_by_type['pings'] ) ) {
   $trackback_number = count( $comments_by_type['pings'] ); ?>
   <h2 id="trackbacks-title">
      <?php /* translators: Title for trackback list. 1=trackback number,
      2=post title */
      printf( _n(
         '%1$s Trackback on "%2$s"',
         '%1$s Trackbacks on "%2$s"',
         $trackback_number,
         'hannover'
      ), number_format_i18n( $trackback_number ), get_the_title() ) ?>
   </h2>
```

Listing 37: Ausgabe der Überschrift für die Trackback-Liste in der comments.php

In der `comments.php` geht es weiter mit dem Code aus Listing 37, der im Prinzip dasselbe für Trackbacks erledigt, was Listing 35 für Kommentare macht. Statt auf den `comment`-Teil des `$comments_by_type`-Arrays wird hier auf `pings` geprüft. Dementsprechend wird die Zählung im Vergleich zu den Kommentaren angepasst, ebenso wie der Inhalt der Überschrift.

```
   <ol class="commentlist">
      <?php wp_list_comments( array(
         'callback'   => 'hannover_trackbacks',
         'type'       => 'pings'
      ) ); ?>
   </ol>
<?php }
```

Listing 38: Ausgabe der Trackbacks in der comments.php

Um die Liste der Trackbacks mit dem Code aus Listing 38 anzuzeigen, kann wieder `wp_list_comments()` eingesetzt werden. Als Callback-Funktion wird `hannover_trackbacks()` übergeben und als Typ `pings`.

```
      the_comments_navigation();
      if ( ! comments_open() && get_comments_number() ) { ?>
         <p class="nocomments">
```

[160] vgl. WordPress: Code Reference › Functions › wp_list_comments

```
            <?php _e( 'Comments are closed.', 'hannover' ); ?>
        </p>
    <?php }
}
comment_form( array(
        'label_submit' => __( 'Submit Comment', 'hannover' )
) ); ?>
</div>
```

Listing 39: Letzter Teil der comments.php

In Listing 39 sind die letzten Code-Zeilen der `comments.php` abgebildet. Mit der Funktion `the_comments_navigation()` kann seit WordPress 4.4 eine Navigation für Kommentare angezeigt werden, falls diese nicht alle auf einer Seite angezeigt werden sollen.[161] Anschließend wird geprüft, ob Kommentare geschlossen sind und ob es bereits Kommentare gibt. Nur in diesem Fall wird ein Hinweis darauf ausgegeben, dass Kommentare geschlossen sind.

Abschließend wird mit `comment_form()` das Kommentar-Formular ausgegeben und mit dem Wert für den `label_submit`-Schlüssel die Standard-Beschriftung des Absende-Buttons verändert.[162] Um die Anzeige der Reaktionen abzuschließen, fehlen noch die zwei Callback-Funktionen für die Liste der Kommentare sowie der Trackbacks.

5.15.1 Anzeige der Kommentare

```
function hannover_comments( $comment, $args, $depth ) { ?>
   <li <?php comment_class(); ?>
   id="li-comment-<?php comment_ID(); ?>">
   <article id="comment-<?php comment_ID(); ?>" class="comment">
      <header class="comment-meta comment-author vcard clearfix">
         <?php echo get_avatar( $comment, 50 ); ?>
         <cite class="fn"><?php comment_author_link(); ?></cite>
         <?php printf(
            '<a href="%1$s"><time datetime="%2$s">%3$s</time></a>',
            get_comment_link( $comment->comment_ID ),
            get_comment_time( 'c' ),
            /* translators: 1=date 2=time */
            sprintf( __( '%1$s @ %2$s', 'hannover' ),
                get_comment_date(),
                get_comment_time()
            )
         ); ?>
      </header>
```

Listing 40: Ausgabe des Kommentar-Headers in hannover_comments() in der functions.php

[161] vgl. Obenland: Core-Trac › Ticket #30589 – Comments navigation template tags
[162] vgl. WordPress: Code Reference › Functions › comment_form

Der `hannover_comments()`-Funktion werden die Parameter `$comment`, `$args` sowie `$depth` übergeben. `$comment` enthält das Kommentar-Objekt, mit dem der Autor, Inhalt des Kommentars und Ähnliches zugänglich sind. Mit `$args` werden die Argumente vorgehalten, die an `wp_list_comments()` übergeben wurden, wie etwa die maximale Verschachtelungstiefe der Kommentare. Zu guter Letzt beinhaltet `$depth` die Ebene des gerade angezeigten Kommentars als Zahl.

Zunächst wird in der Funktion ein Listenelement geöffnet, das mit `comment_class()` nützliche Klassen zugewiesen bekommt – ähnlich der bereits besprochenen Funktionen `post_class()` und `body_class()`. Dieses Listenelement muss später nicht geschlossen werden, da WordPress das von sich aus erledigt.[163]

Für eine eindeutige Kommentar-ID wird in dem `article`-Element `comment_ID()` genutzt, um die ID des Kommentars auszugeben.[164] Mit `get_avatar()` wird das Gravatar-Bild des Kommentators eingebunden, sofern die E-Mail-Adresse mit dem Dienst verbunden ist.[165] Der erste Parameter für die Funktion ist das Kommentar-Objekt, der zweite die Größe des quadratischen Bildes. Mit der Funktion `comment_author_link()` wird der Name des Kommentators ausgegeben, der auf die Website verlinkt, die im Kommentarformular eingegeben wurde.[166]

Nach dem Namen des Autors soll als Permalink für den Kommentar das Datum innerhalb des `time`-Elements ausgegeben werden.[167] Um den Link des Kommentars zu ermitteln, wird dessen ID an die `get_comment_link()`-Funktion übergeben.[168] Für das `datetime`-Attribut wird mit `get_comment_time()` und dem Parameter `c` das Datum und die Uhrzeit im ISO-8601-Format ausgegeben.[169, 170] Angezeigt werden soll das Datum nach dem Muster `Uhrzeit @ Datum`, wobei das jeweilige Format aus dem Backend übernommen werden soll. Das Format aus dem Backend wird für das Datum mit der Funktion `get_comment_date()` ermittelt, für die Zeit mit `get_comment_time()`, wobei jeweils kein Parameter übergeben werden darf.[171]

---

[163] vgl. WordPress: Code Reference › Functions › comment_class
[164] vgl. WordPress: Code Reference › Functions › comment_ID
[165] vgl. WordPress: Code Reference › Functions › get_avatar
[166] vgl. WordPress: Code Reference › Functions › comment_author_link
[167] vgl. Sari u.a. (2015): MDN › HTML › <time>
[168] vgl. WordPress: Code Reference › Functions › get_comment_link
[169] vgl. WordPress: Code Reference › Functions › get_comment_time
[170] vgl. PHP Group: date
[171] vgl. WordPress: Code Reference › Functions › get_comment_date

```
<?php if ( '0' == $comment->comment_approved ) { ?>
   <p class="comment-awaiting-moderation">
      <?php
      _e( 'Your comment is awaiting moderation.', 'hannover' );
      ?>
   </p>
<?php } ?>
```

Listing 41: Ausgabe eines Moderationshinweises in hannover_comments()

Um dem Anwender einen Hinweis anzuzeigen, wenn sein Kommentar noch freigeschaltet werden muss, wird in Listing 41 überprüft, ob die Eigenschaft `comment_approved` des Kommentar-Objekts `0` ist, und in dem Fall ein Hinweis ausgegeben.

```
<div class="comment-content comment">
   <?php comment_text(); ?>
   <?php edit_comment_link( __( 'Edit', 'hannover' ), '<p class="edit-
link">', '</p>' ); ?>
</div>
```

Listing 42: Ausgabe des Kommentar-Inhalts in hannover_comments()

Der Code in Listing 42 kümmert sich um die Ausgabe des Kommentars – mit der Funktion `comment_text()` wird der Inhalt ausgegeben.[172] Um berechtigten Nutzern gleich einen Link ins Backend anzuzeigen, wo sie den Kommentar bearbeiten können, wird dieser nach dem Inhalt mit `edit_comment_link()` angezeigt.[173] Der erste Parameter für die Funktion ist der Text, der verlinkt angezeigt werden soll, gefolgt von dem Markup davor und dahinter.

```
      <div class="reply">
         <?php comment_reply_link(
            array(
               'reply_text' => __( 'Reply', 'hannover' ),
               'depth'      => $depth,
               'max_depth'  => $args['max_depth']
            )
         ); ?>
      </div>
   </article>
   <?php
}
```

Listing 43: Ausgabe des Antworten-Links in hannover_comments()

Listing 43 zeigt das Ende der `hannover_comments()`-Funktion. Um die direkte Antwort auf einen anderen Kommentar zu ermöglichen, falls verschachtelte Kommentare auf der Website aktiviert sind, wird mit `comment_reply_link()` ein entsprechender Link aus-

[172] vgl. WordPress: Code Reference › Functions › comment_text
[173] vgl. WordPress: Code Reference › Functions › edit_comment_link

gegeben.[174] In dem Options-Array wird mit `reply_text` der anzuzeigende Text festgelegt, mit `depth` die Tiefe der Kommentarebene und über `max_depth` die Ebene, bis zu der Antworten möglich sind.

Standardmäßig ist es nun so, dass nach einem Klick auf den Link für die Antwort auf einen Kommentar die Seite komplett neu geladen wird. WordPress stellt allerdings ein Skript zur Verfügung, das nach einem Klick das Kommentarfeld unter dem entsprechenden Kommentar erscheinen lässt und die Antwort ohne Neuladen der Seite entgegennimmt. Alle Skripte und Stylesheets, die das Theme benötigt, werden in der `hannover_scripts_styles()`-Funktion in der `functions.php` eingebunden. Dafür gibt es von WordPress die Funktionen `wp_enqueue_script()` für Skripte sowie `wp_enqueue_style()` für Stylesheets.[175, 176] Vorteil dieser Funktionen gegenüber einer festen Integrierung in der `header.php` ist beispielsweise, dass Abhängigkeiten von Skripten berücksichtigt werden können.

```
function hannover_scripts_styles() {
   if ( is_singular() && comments_open()
      && get_option( 'thread_comments' ) ) {
      wp_enqueue_script( 'comment-reply' );
   }
}

add_action( 'wp_enqueue_scripts', 'hannover_scripts_styles' );
```

Listing 44: Einbinden des Comment-Reply-Skripts in hannover_scripts_styles()

In Listing 44 ist die Funktion `hannover_scripts_styles()` zu sehen, die bisher nur das Skript für Kommentarantworten einbindet. Zunächst wird geprüft, ob das Skript überhaupt eingebunden werden muss, also ob sich der Nutzer auf einer Einzelseite befindet, deren Kommentare offen sind und ob verschachtelte Kommentare erlaubt sind. Ist das der Fall, wird mit `wp_euqueue_script()` das Skript mit dem Bezeichner `comment-reply` eingebunden. Da das Skript bereits im WordPress-Core registriert wird, reicht hier der erste Parameter aus, um das Skript einzubinden. Später, wenn das Theme beispielsweise eine Lightbox integriert, werden auch die übrigen Parameter zum Einsatz kommen.

[174] vgl. WordPress: Code Reference › Functions › comment_reply_link
[175] vgl. WordPress: Code Reference › Functions › wp_enqueue_script
[176] vgl. WordPress: Code Reference › Functions › wp_enqueue_style

5.15.2 Anzeige der Trackbacks

```
function hannover_trackbacks( $comment ) { ?>
<li <?php comment_class(); ?> id="comment-<?php comment_ID(); ?>">
   <?php _e( 'Trackback:', 'hannover' ); ?>
   <?php comment_author_link(); ?>
   <?php edit_comment_link(
            __( '(Edit)', 'hannover' ),
            '<span class="edit-link">', '</span>' );
}
```

Listing 45: hannover_trackbacks() in der functions.php

Die Trackbacks sollen in einer Liste angezeigt werden – vorne wird der übersetzbare Begriff *Trackback* angezeigt, gefolgt von dem Titel und einem Bearbeiten-Link für berechtigte Nutzer. Listing 45 zeigt die `hannover_trackbacks()`-Funktion, die dafür zuständig ist. Zuerst wird – wie aus der Funktion für die Kommentare bekannt – das Listenelement geöffnet. Anschließend wird `Trackback:` angezeigt, gefolgt von dem Titel des verlinkenden Beitrags, der mit `comment_author_link()` ausgegeben wird. Abschließend wird mit `edit_comment_link()` ein Link ins Backend angezeigt, über den berechtigte Benutzer direkt zu der Bearbeitungsansicht des Trackbacks gelangen können.

5.16 404-Template

Wenn es eine eingegebene Adresse nicht gibt, und WordPress daraus auch keine Weiterleitung auf einen bestehenden Inhalt erstellen kann, wird nach einer `404.php` im Theme gesucht. Hier kann ein kurzer Hinweis platziert sowie das Suchformular angezeigt werden, damit der Nutzer sein Glück damit versuchen kann.

```
<?php get_header(); ?>
   <main role="main">
      <article id="post-0" class="post no-results not-found">
         <header class="entry-header">
            <h1 class="entry-title">
               <?php _e( 'Nothing Found', 'hannover' ); ?>
            </h1>
         </header>
         <div class="entry-content">
            <p><?php _e( 'Apologies, but no results were found for the
               requested archive. Perhaps searching will help find a
               related post.', 'hannover' ); ?></p>
            <?php get_search_form(); ?>
         </div>
      </article>
   </main>
```

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

Listing 46: 404.php

In Listing 46 ist die gesamte `404.php` abgebildet. Nachdem auf altbekannte Art und Weise der Header des Themes eingebunden wurde, wird in dem `article`-Element ein kurzer Hinweis ausgegeben. Anschließend wird `get_search_form()` aufgerufen, um das Suchformular einzubinden.[177]

5.17 Portfolio-Übersicht

Wie bereits in Kapitel 2.1 ausgeführt, soll der Nutzer eine Seite festlegen können, auf der alle Arbeiten angezeigt werden. Dafür muss zuerst festgelegt werden können, welche Galerien und Bild-Beiträge er als Portfolio-Elemente nutzen möchte. Hier sollen zwei verschiedene Ansätze umgesetzt werden, um dem Nutzer mehr Flexibilität zu bieten: Entweder werden alle Beiträge vom Typ *Galerie* und *Bild* als Portfolio-Elemente behandelt, oder der Nutzer wählt eine Kategorie aus, der alle Portfolio-Elemente angehören müssen. Diese Einstellung soll im Customizer vorgenommen werden.

5.17.1 Option zur Auswahl der Portfolio-Kategorie

Für die Auswahl der Kategorie wird eine `select`-Liste erstellt, aus der der Nutzer wählen kann. Um diese Auswahl anstatt aller Galerie- und Bild-Beiträge zu nutzen, wird darüber eine Checkbox eingefügt, die der Nutzer anhaken muss. Die Funktion für den Customizer wird in die `customizer.php` innerhalb des `inc`-Ordners ausgelagert, damit die `functions.php` übersichtlicher ist. Um die Datei in die `functions.php` einzubinden, ist die Code-Zeile aus Listing 47 notwendig.

```
require get_template_directory() . '/inc/customizer.php';
```

Listing 47: Einbinden der customizer.php in die functions.php

Mit dem `require`-Statement aus PHP wird die Datei eingebunden.[178] Den Pfad zum Verzeichnis des Hannover-Themes gibt die `get_template_directory()`-Funktion aus.[179]

[177] vgl. WordPress: Code Reference › Functions › get_search_form
[178] vgl. PHP Group: Kontrollstrukturen › require
[179] vgl. WordPress: Code Reference › Functions › get_template_directory

Code für die Checkbox

```
function hannover_customize_register( $wp_customize ) {
   $wp_customize->add_setting(
      'portfolio_from_category', array(
         'sanitize_callback' => 'hannover_sanitize_checkbox'
      )
   );

   $wp_customize->add_control(
      'portfolio_from_category', array(
         'label'    => __( 'Use a category instead of all gallery and
         image posts for portfolio elements', 'hannover' ),
         'type'     => 'checkbox',
         'section'  => 'portfolio_elements',
         'settings' => 'portfolio_from_category'
      )
   );

   $wp_customize->add_panel(
      'theme_options', array(
         'title' => __( 'Theme Options', 'hannover' ),
      )
   );

   $wp_customize->add_section(
      'portfolio_elements', array(
         'title' => __( 'Portfolio elements', 'hannover' ),
         'panel' => 'theme_options'
      )
   );
}

add_action( 'customize_register', 'hannover_customize_register' );
```

Listing 48: Erstellung der Portfolio-Kategorie-Checkbox für den Customizer in der customizer.php

In Listing 48 ist der Code abgebildet, der die Checkbox in den Customizer einfügt. `add_setting()` erwartet als ersten Parameter eine ID und danach ein Array von Optionen. Als Optionen können beispielsweise ein Standard-Wert sowie der Typ der Einstellung übergeben werden.[180] An dieser Stelle wird kein Standard benötigt und die übrigen Default-Parameter können ebenso übernommen werden. Als `sanitize_callback` muss eine Funktion angegeben werden, die den eingegebenen Wert überprüft.

Die `add_control()`-Funktion, die für die Darstellung des Formularelements zuständig ist, erwartet ebenfalls als ersten Parameter eine ID. Als Parameter wird an `label` die Beschriftung übergeben, die im Customizer sichtbar ist. Darüber hinaus muss mit `type` der

[180] vgl. WordPress: Theme Handbook › Advanced Theme Topics › Theme Options – The Customizer API › Settings

Formular-Typ angegeben werden – WordPress unterstützt hier die gängigen Typen.[181] Um eine Checkbox zu erzeugen, muss der Schlüssel `checkbox` übergeben werden. Mit `section` wird angegeben, in welchem Teil des Customizers die Option dargestellt werden soll. Hier können entweder Core-Bereiche oder ein eigener angegeben werden. Der Wert für den `settings`-Schlüssel muss mit der ID aus der `add_setting()`-Methode übereinstimmen, für die das Formular-Element ist – wenn dieser Schlüssel nicht angegeben wird, dann nutzt WordPress die ID aus der `add_control()`-Methode.

Um die Theme-Optionen übersichtlich in einem eigenen Bereich zu bündeln, werden sie in einem *Panel* angeordnet. Innerhalb dieses Panels können mit *Sections* wiederum Unterbereiche erstellt werden, innerhalb derer die Formular-Elemente angezeigt werden. Die `add_panel()`-Funktion erwartet als ersten Parameter eine ID und im Anschluss ein Options-Array, in dem der Titel, eine Beschreibung sowie die Priorität des Panels übergeben werden können. An dieser Stelle wird nur der Titel angegeben.

Um den Bereich zu erstellen, in dem die Checkbox angezeigt werden soll, wird mit der `add_section()`-Methode der Portfolio-Bereich erstellt. Die ID muss dabei mit dem Wert übereinstimmen, der in `add_control()` für den `section`-Schlüssel übergeben wurde. Neben dem Titel wird die ID des Panels übergeben, in dem die Section angezeigt werden soll.

Als `sanitize_callback` wurde `hannover_sanitize_checkbox` angegeben, diese Funktion ist in Listing 49 abgebildet, die aus der hilfreichen Code-Sammlung des Theme-Review-Teams übernommen ist.[182] Innerhalb von `return` wird überprüft, ob der übergebene Wert gesetzt und `true` ist und in diesem Fall `true` zurückgegeben, andernfalls `false`.

```
function hannover_sanitize_checkbox( $checked ) {
   return ( ( isset( $checked ) && true == $checked ) ? true : false );
}
```

Listing 49: hannover_sanitize_checkbox() in der customizer.php

[181] vgl. WordPress: Theme Handbook › Advanced Theme Topics › Theme Options – The Customizer API › Controls
[182] vgl. Bennett: WPTRT/code-examples/customizer/sanitization-callbacks.php

Code für die Kategorie-Liste

Damit der Nutzer die Kategorie auswählen kann, aus der die Portfolio-Elemente entnommen werden sollen, muss eine `select`-Liste erstellt werden, die alle Kategorien der WordPress-Installation enthält.

```
$categories     = get_categories();
$category_array = array();
if ( ! empty( $categories ) ) {
   foreach ( $categories as $category ) {
      $category_array[ $category->term_id ] = $category->name;
   }
}

$wp_customize->add_setting(
   'portfolio_category', array(
      'sanitize_callback' => 'hannover_sanitize_select'
   )
);

$wp_customize->add_control(
   'portfolio_category', array(
      'label'           => __( 'Portfolio category', 'hannover' ),
      'type'            => 'select',
      'section'         => 'portfolio_elements',
      'settings'        => 'portfolio_category',
      'choices'         => $category_array,
      'active_callback' => 'hannover_use_portfolio_category_callback'

   )
);
```

Listing 50: Kategorie-Liste in hannover_customize_register() in der customizer.php

Der Code aus Listing 50 gehört in die `hannover_customize_register()`-Funktion. Um im Customizer ein `select`-Formularelement erstellen zu können, muss ein Array mit den Auswahloptionen übergeben werden, das als Schlüssel das `value`-Attribut und als jeweiligen Wert die sichtbare Beschriftung enthält. Um ein Array für die Kategorien zu erstellen, werden mit `get_categories()` sämtliche Kategorien als Array in `$categories` gespeichert.[183] Anschließend wird ein leeres Array erzeugt, das später das Ergebnisarray sein wird. In einer Schleife werden die Kategorien durchlaufen und das Array gefüllt. Als Schlüssel wird immer die `term_id` der Kategorie festgelegt, als Wert der Name.

Um die Einstellung hinzuzufügen, wird erneut die `add_setting()`-Methode verwendet, deren Callback-Funktion später besprochen wird. Um das Formular-Element zu erzeugen,

[183] vgl. WordPress: Code Reference › Functions › get_categories

werden dem Array der `add_control()`-Methode die bereits aus Listing 48 bekannten Schlüssel übergeben. Der einzige Unterschied ist der Schlüssel `choices`, der bei einem Element von Typ `select` übergeben werden muss. Dieser `choices`-Schlüssel erwartet das Array mit den Auswahloptionen, das im Fall der Kategorie-Liste mit dem Array in der Variable `$category_array` vorbereitet wurde. Um das `select`-Element nur anzuzeigen, wenn die Checkbox angewählt ist, wird ein `active_callback` übergeben.

Die Auswahl des Nutzers wird von `hannover_sanitize_select()` als Callback-Funktion geprüft, die in Listing 51 abgebildet ist und wie die Funktion für die Checkbox dem Code-Beispiel des Theme-Review-Teams entnommen ist.[184]

```
function hannover_sanitize_select( $input, $setting ) {
   $input   = sanitize_key( $input );
   $choices = $setting->manager->get_control( $setting->id )->choices;
   return (
   array_key_exists( $input, $choices ) ? $input : $setting->default
   );
}
```

Listing 51: hannover_sanitize_select() in der customizer.php

Dabei wird der Funktion als erster Parameter der Wert des gewählten Eintrags übergeben und als zweiter das Settings-Objekt. Zuerst wird mit `sanitize_key()` sichergestellt, dass der gewählte Wert nur aus kleinen alphanumerischen Zeichen, Bindestrichen und Unterstrichen besteht.[185] Im Anschluss werden die Auswahlmöglichkeiten des Formularelements in der `$choices`-Variable gespeichert und danach überprüft, ob der Wert aus `$input` als Schlüssel in dem Array der vorhandenen Auswahlmöglichkeiten vorkommt. Ist das der Fall, wird der Wert zurückgegeben, andernfalls der Standard-Wert.

Die `hannover_use_portfolio_category_callback()`-Funktion, mit der geprüft wird, ob das `select`-Element angezeigt werden muss, ist in Listing 52 abgebildet.

```
function hannover_use_portfolio_category_callback( $control ) {
   if ( $control->manager->get_setting(
         'portfolio_from_category'
      )->value() == 'checked' ) {
      return true;
   } else {
      return false;
   }
}
```

Listing 52: hannover_use_portfolio_category_callback() in der customizer.php

[184] vgl. Bennett: WPTRT/code-examples/customizer/sanitization-callbacks.php
[185] vgl. WordPress: Code Reference › Functions › sanitize_key

Wenn der Wert der `portfolio_from_category`-Checkbox `checked` entspricht, wird als Ergebnis `true` zurückgegeben, andernfalls `false`. Die erstellten Einstellungen im Customizer sind in Abbildung 7 zu sehen. Wenn der Haken nicht gesetzt ist, wird auch das Auswahl-Element für die Kategorie nicht angezeigt.

Abbildung 7: Customizer-Einstellungen für die Auswahl der Portfolio-Kategorie

5.17.2 Option zum Ausschluss der Portfolio-Elemente vom Blog

Als weitere Funktion muss dem Nutzer ermöglicht werden, die Portfolio-Elemente vom Blog auszuschließen. Auch diese Funktion wird über eine einfache Checkbox gelöst, die im Customizer angezeigt wird. Listing 53, das den Code für diese Funktion anzeigt, ähnelt dem ersten Code-Teil aus Listing 48. Als Callback-Funktion wird die bereits verwendete `hannover_sanitize_callback()` eingesetzt und der Titel des Formularelements sowie die ID, die an `add_setting()` und `add_control()` übergeben wird, sind angepasst.

```
$wp_customize->add_setting(
   'exclude_portfolio_elements_from_blog', array(
      'sanitize_callback' => 'hannover_sanitize_checkbox'
   )
);

$wp_customize->add_control(
   'exclude_portfolio_elements_from_blog', array(
      'label'    =>
         __( 'Exclude the portfolio elements from the blog',
             'hannover' ),
      'type'     => 'checkbox',
      'section'  => 'portfolio_elements',
      'settings' => 'exclude_portfolio_elements_from_blog'
```

```
    )
);
```

Listing 53: Portfolio-Elemente vom Blog ausschließen in hannover_register_customize()

5.17.3 Option zum Festlegen der Anzahl von Portfolio-Elementen auf einer Seite

Um dem Theme-Nutzer auch die Möglichkeit zu geben, die Portfolio-Ansicht mit einer Pagination zu versehen, muss er die Anzahl von Elementen für eine Seite angeben können.

```
$wp_customize->add_setting(
   'portfolio_elements_per_page', array(
      'default'           => 0,
      'sanitize_callback' => 'absint'
   )
);

$wp_customize->add_control(
   'portfolio_elements_per_page', array(
      'label'    =>
         __( 'Number of portfolio elements to show on one page
              (0 to show all elements on one page).', 'hannover' ),
      'type'     => 'number',
      'section'  => 'portfolio_elements',
      'settings' => 'portfolio_elements_per_page',
   )
);
```

Listing 54: Option zum Festlegen der Anzahl von Portfolio-Elementenen pro Seite

Für diese Option wird ein Nummernfeld erstellt, das standardmäßig den Wert `0` hat. In Listing 54 ist der entsprechende Code zu sehen. Als Callback-Funktion wird `absint()` genutzt.[186] Als `type` muss der `add_control()`-Methode der Wert `number` übergeben werden.

5.17.4 Portfolio-Elemente auf einer Seite anzeigen

Um die Portfolio-Elemente auf einer Seite anzuzeigen, wird ein Seiten-Template erstellt, das der Nutzer beim Bearbeiten einer Seite im Backend auswählen kann. Um die Ordnerstruktur übersichtlich zu halten, werden die Seiten-Templates im Ordner `page-`

[186] vgl. WordPress: Code Reference › Functions › absint

`templates` platziert, die WordPress neben der obersten Theme-Ordner-Ebene nach Seiten-Templates durchsucht.[187]

Das Seiten-Template

Um ein Seiten-Template zu erstellen, muss am Anfang der PHP-Datei ein Kommentar nach bestimmtem Schema geschrieben werden, damit WordPress erkennt, dass es sich um ein Seiten-Template handelt.[188]

```
<?php
/**
 * Template Name: Portfolio page
 */
```

Listing 55: Kommentar zur Identifizierung des Seiten-Templates in der portfolio-page.php

In Listing 55 ist der Anfang der `portfolio-page.php`-Datei mit dem entsprechenden Kommentar zu sehen. Wichtig ist der Teil `Template Name:`, der WordPress mitteilt, dass es sich um ein Seiten-Template handelt.

```
get_header(); ?>
   <main role="main">
      <header>
         <h1 class="page-title"><?php single_post_title(); ?></h1>
      </header>
```

Listing 56: Seitentitel auf der Portfolio-Übersicht in der portfolio-page.php

Ebenso unspektakulär geht es in der Datei mit dem Inhalt von Listing 56 weiter. Die `header.php` wird eingebunden und innerhalb des `main`-Elements sowie einem `header` wird der Titel der Seite ausgegeben.

Um nur Beiträge zu erhalten, die bestimmten Bedingungen entsprechen, muss eine angepasste Anfrage an die Datenbank gestellt werden. Dazu muss zunächst das Array erstellt werden, das die Bedingungen für die Abfrage enthält. Da es ganz grundsätzlich die zwei möglichen Fälle gibt, dass die Portfolio-Beiträge anhand der Kategorie oder nur anhand des Post-Formats ermittelt werden, müssen zwei Arrays erstellt werden – abhängig von der Wahl im Customizer wird dann das korrekte für die Anfrage verwendet.

[187] vgl. WordPress: Theme Handbook › Template Files Section › Page Template Files › Page Templates › File Organization of Page Templates
[188] vgl. WordPress: Theme Handbook › Template Files Section › Page Template Files › Page Templates › Creating Custom Page Templates for Global Use

```php
<?php
$use_portfolio_category = get_theme_mod( 'portfolio_from_category' );
$portfolio_category  = get_theme_mod( 'portfolio_category' );
$elements_per_page = get_theme_mod( 'portfolio_elements_per_page',
                                    0 );
$paged = ( get_query_var( 'paged' ) ) ? get_query_var( 'paged' ) : 1;
if ( $elements_per_page == 0 ) {
   $elements_per_page = - 1;
}
if ( $use_portfolio_category == 'checked' && $portfolio_category !== '' )
{
   $args = array(
      'posts_per_page' => $elements_per_page,
      'paged'          => $paged,
      'tax_query'      => array(
         'relation' => 'AND',
         array(
            'taxonomy' => 'category',
            'field'    => 'term_id',
            'terms'    => array( $portfolio_category ),
         ),
         array(
            'taxonomy' => 'post_format',
            'field'    => 'slug',
            'terms'    => array(
               'post-format-gallery',
               'post-format-image'
            ),
         ),
      )
   );
} else {
   $args = array(
      'posts_per_page' => $elements_per_page,
      'tax_query'      => 'tax_query'      => array(
         'taxonomy' => 'post_format',
         'field'    => 'slug',
         'terms'    => array(
            'post-format-gallery',
            'post-format-image'
         ),
      ),
      'paged'          => $paged
   );
}
```

Listing 57: Vorbereitung der Argument-Arrays für die QP_Query

In Listing 57 werden zunächst die Werte aus dem Customizer ermittelt, mit denen der Nutzer bisher die Portfolio-Übersicht beeinflussen kann. Dabei kommt immer die Funktion `get_theme_mod()` zum Einsatz, die als ersten Parameter die ID erwartet, die an die

`add_setting()`-Methode übergeben wurde.[189] Als zweiter Parameter kann ein Standard-Wert übergeben werden, wie bei der Anzahl der Portfolio-Elemente.

Damit die Pagination trotz angepasster Abfrage funktioniert, muss in der Datei `portfolio-page.php` als nächstes der Wert für den Schlüssel `paged` vorbereitet werden, der in `$paged` gespeichert wird.[190] Im Anschluss wird überprüft, ob der Nutzer bei der Anzahl der Elemente pro Seite eine `0` eingestellt hat, um keine Pagination zu erhalten. In diesem Fall wird der Wert auf `-1` geändert, da das als Wert bei dem Schlüssel `posts_per_page` für unendlich steht.

Danach wird geprüft, ob der Nutzer die Checkbox angehakt hat, um eine Portfolio-Kategorie zu verwenden und ob die Kategorie gesetzt ist. In dem Fall wird das Argument-Array für die Abfrage wie folgt zusammengebaut: Für die Beiträge pro Seite wird der Wert eingesetzt, der in `$elements_per_page` gespeichert ist. Für die korrekte Pagination bekommt der Schlüssel `paged` den vorbereiteten Wert aus der `$paged`-Variablen. Um Bedingungen zu erstellen, die die Taxonomien der Beiträge betreffen, gibt es den Schlüssel `tax_query`.[191] Innerhalb dieses Schlüssels sind auch Beziehungen zwischen verschiedenen Bedingungen möglich, die mit dem `relation`-Schlüssel angegeben werden. In der `portfolio-page.php` sollen die Beiträge geholt werden, die sowohl in der Kategorie, die der Nutzer ausgewählt hat, als auch vom Typ Galerie oder Bild sind.

Es müssen also die zwei Bedingungen der Kategorie und der Post-Formate mit `AND` verknüpft werden – dieser Wert wird an den Schlüssel `relation` übergeben. Im Anschluss wird die Kategorie-Bedingung formuliert, indem an das Array für den Schlüssel `taxonomy` der Wert `category` übergeben wird, dessen `term_id` der ID entsprechen soll, die die Kategorie besitzt, die vom Nutzer im Customizer ausgewählt wurde. Um auf bestimmte Post-Formate einzuschränken, wird im nächsten Array der Wert `post_format` genutzt. Der Slug dieser Taxonomie soll für den Beitrag entweder `post-format-gallery` oder `post-format-image` entsprechen.[192]

Wenn der Nutzer hingegen keine Kategorie ausgewählt hat, wird als Wert für `tax_query` nur das Array für die Post-Format-Bedingung übergeben, sodass alle Beiträge vom Typ *Galerie* und *Bild* unabhängig von der Kategorie ausgegeben werden.

---

[189] vgl. WordPress: Code Reference › Functions › get_theme_mod
[190] vgl. Bennett: How to fix pagination for custom loops? (2013)
[191] vgl. WordPress: Codex › Class Reference › WP_Query › Taxonomy Parameters
[192] vgl. WordPress: Codex › Class Reference › WP_Query › Taxonomy Parameters

```
$portfolio_query = new WP_Query( $args );
$temp_query      = $wp_query;
$wp_query        = null;
$wp_query        = $portfolio_query;
if ( $portfolio_query->have_posts() ) {
   while ( $portfolio_query->have_posts() ) {
      $portfolio_query->the_post();
      get_template_part('template-parts/content','portfolio-element');
   }
}
```

Listing 58: Durchführung der angepassten Query-Loop in der portfolio-page.php

In Listing 58 ist die Durchführung und Vorbereitung der Beitragsschleife zu sehen. Um die Abfrage mit dem entsprechenden Argumente-Array durchzuführen, wird in `$portfolio_query` ein Objekt der `WP_Query`-Klasse erstellt, dem das Array übergeben wird. Um später wieder auf die Ursprungs-Query zugreifen zu können, wird diese in `$temp_query` zwischengespeichert und das Standard-Query-Objekt von WordPress, `$wp_query`, auf `Null` gesetzt. Im Anschluss wird dieser Variable das angepasste Query-Objekt aus `$portfolio_query` zugewiesen, damit die Funktion für die Pagination, die mit `$wp_query` arbeitet, später das korrekte Ergebnis ausgibt.

Nun können die bereits bekannten Methoden `have_posts()` und `the_post()` von dem Query-Objekt `$portfolio_query` genutzt werden, um die Beiträge in einer Schleife zu durchlaufen. Für die Ausgabe wird die Datei `content-portfolio-element.php` eingebunden – für eine zweite Variante der Ausgabe muss dann lediglich diese Zeile mit dem `get_template_part()`-Aufruf verändert werden.

```
      wp_reset_postdata();
      the_posts_pagination( array( 'type' => 'list' ) );
      $wp_query = null;
      $wp_query = $temp_query; ?>
   </main>
<?php get_footer();
```

Listing 59: Ende der portfolio-page.php

Im Anschluss wird, wie in Listing 59 zu sehen, mit `wp_reset_postdata()` das globale `$post`-Objekt zurückgesetzt und danach mit `the_posts_pagination()` die Pagination ausgegeben.[193, 194] Zum Schluss wird `$wp_query` wieder das ursprüngliche Query-Objekt zugewiesen. Diese komplette Lösung mit der Query für die Pagination ist – entsprechend

[193] vgl. WordPress: Code Reference › Functions › wp_reset_postdata
[194] vgl. WordPress: Code Reference › Functions › the_posts_pagination

angepasst – der umfangreichen Antwort von Chip Bennett auf eine ähnliche Frage bei wordpress.stackexchange.com entnommen.[195]

Ausgabe der Portfolio-Elemente in content-portfolio-page.php

```
<article <?php post_class( 'element' ); ?> id="post-<?php the_ID();
   ?>">
   <a href="<?php the_permalink(); ?>">
      <h2 class="screen-reader-text"><?php the_title(); ?></h2>
      <?php hannover_image_from_gallery_or_image_post(
         'thumbnail', $post
      ); ?>
   </a>
</article>
```

Listing 60: content-portfolio-page.php

Listing 60 zeigt die Datei, die standardmäßig für die Ausgabe der Portfolio-Elemente zuständig ist. Durch den Parameter für die `post_class()`-Funktion wird eine zusätzliche Klasse eingefügt, damit die Elemente nachher mit CSS eindeutig von dem alternativen Layout abgegrenzt werden können.[196] Innerhalb des `article`-Elements wird der verlinkte Titel für Screen-Reader-Nutzer ausgegeben sowie `hannover_image_from_gallery_or_image_post()` aufgerufen, um das erste Bild aus dem Galerie- oder Bild-Beitrag anzuzeigen. Damit diese Funktion etwas flexibler ist und eventuell noch an weiteren Stellen genutzt werden kann, wird als erster Parameter die gewünschte Größe des Bildes übergeben, gefolgt von dem Post-Objekt.

```
function hannover_image_from_gallery_or_image_post( $size, $post ) {
   if ( has_post_thumbnail() ) {
      the_post_thumbnail( $size );
   } else {
```

Listing 61: Anfang der hannover_image_from_gallery_or_image_post() in der functions.php

In Listing 61 sind die ersten Zeilen dieser Funktion zu sehen. Der Nutzer soll ein eigenes Vorschaubild über die Beitragsbild-Funktion festlegen können, weshalb zuerst mit `has_post_thumbnail()` geprüft wird, ob bei dem Beitrag ein Beitragsbild gesetzt ist.[197] Ist das der Fall, wird dies in der gewünschten Größe ausgegeben.

```
$post_format = get_post_format( $post );
if ( $post_format == 'gallery' ) {
   $images  = hannover_get_gallery_images( $post->ID );
   $image   = array_shift( $images );
```

[195] vgl. Bennett: How to fix pagination for custom loops? (2013)
[196] vgl. WordPress: Code Reference › Functions › post_class
[197] vgl. WordPress: Code Reference › Functions › has_post_thumbnail

```
    $img_tag = wp_get_attachment_image( $image->ID, $size );
}
```

Listing 62: Ausgabe des ersten Galerie-Bildes

Wenn kein Beitragsbild gesetzt ist, müssen die beiden Post-Formate *Bild* und *Galerie* unterschiedlich behandelt werden, um jeweils das erste Bild zu ermitteln. Um zu prüfen, welches Post-Format der aktuelle Beitrag besitzt, wird in Listing 62 die Funktion `get_post_format()` genutzt, der das Post-Objekt übergeben wird. Wenn das Ergebnis des Aufrufs `gallery` ist, es sich bei dem Beitrag also um eine Galerie handelt, werden die Bilder der Galerien von diesem Beitrag über die `hannover_get_gallery_images()`-Funktion ermittelt, die ein Array von Bildern zurückgibt. Da nur das erste Bild benötigt wird, wird mit `array_shift()` alles außer dem ersten Array-Element gelöscht.[198] Anschließend wird über die Funktion `wp_get_attachment_image()` das `img`-Tag des Bildes in `$img_tag` gespeichert – als Parameter erwartet die Funktion die ID des Bildes sowie die Größe.[199] Die Funktion `hannover_get_gallery_images()` wird nach Behandlung der aktuellen Funktion erläutert.

```
elseif ( $post_format == 'image' ) {
    $first_img_url = hannover_get_first_image_from_post_content();
    $pattern       = '/-\d+x\d+(\.\wWordPress$)/i';
    $first_img_url = preg_replace( $pattern, '${1}', $first_img_url );
    $first_img_id  = attachment_url_to_postid( $first_img_url );
    if ( $first_img_id == 0 ) {
        $attachments = get_children( array(
            'post_parent'    => $post->ID,
            'post_status'    => 'inherit',
            'post_type'      => 'attachment',
            'post_mime_type' => 'image',
            'order'          => 'ASC',
            'orderby'        => 'menu_order',
        ) );
        $first_img   = array_shift( $attachments );
        $img_tag     = wp_get_attachment_image( $first_img->ID, $size );
    } else {
        $img_tag = wp_get_attachment_image( $first_img_id, $size );
    }
}
```

Listing 63: Ausgabe des ersten Bildes eines Bild-Beitrags

In Listing 63 ist der Code zu sehen, der das erste Bild eines *Bild*-Beitrags ausgibt. `hannover_get_first_image_from_post_content()` liefert die URL des ersten Bildes aus dem Beitrag zurück – auch diese Funktion wird nach Besprechung der aktuellen

[198] vgl. PHP Group: array_shift
[199] vgl. WordPress: Code Reference › Functions › wp_get_attachment_image

erläutert. Um wie bei dem ersten Bild der Galerie die Funktion `wp_get_attachment_image()` nutzen zu können, ist die ID des Bildes notwendig.

Seit WordPress 4.2 gibt es `attachment_url_to_postid()`, um von einer URL auf die ID zu kommen.[200] Diese Funktion arbeitet nur dann korrekt, wenn die URL der vollständigen Bildgröße übergeben wird – nicht die einer kleineren Größe, die WordPress beim Upload eines Bildes generiert. Aus diesem Grund muss zuerst sichergestellt werden, dass die URL einer generierten Bildgröße angepasst wird, sodass sie der URL von der vollen Größe entspricht. WordPress fügt bei den generierten Größen vor die Dateiendung einen Teil nach dem Schema `-BreitexHöhe`, also beispielsweise `-800x430`, ein. Dieser Teil inklusive der folgenden Dateiendung kann durch den regulären Ausdruck `/-\d+x\d+(\.\wWordPress$)/i` beschrieben werden. Dieser Ausdruck wird mit der `preg_replace()`-Funktion auf die URL angewendet. Als ersten Parameter erwartet die Funktion den regulären Ausdruck, als zweiten das, womit die gefundene Stelle ersetzt werden soll und als dritten den String, der durchsucht werden soll. Ersetzt werden soll der Treffer durch die Dateiendung, weshalb diese im regulären Ausdruck von runden Klammern umschlossen ist, sodass sie mit `${1}` angesprochen werden kann. Falls das Bild in voller Größe eingefügt wurde, wird der reguläre Ausdruck nichts finden und die URL unverändert bleiben.

Danach wird die URL an `attachment_url_to_postid()` übergeben und die ID in `$first_img_id` gespeichert. Wenn diese Funktion kein Bild findet, wird `0` zurückgegeben – das kann beispielsweise der Fall sein, wenn der Nutzer kein Bild in den Beitrag eingefügt hat oder das Bild im Beitrag nicht in die eigene WordPress-Installation hochgeladen wurde. In diesem Fall wird versucht, mit `get_children()` die Bilder zu ermitteln, die zu dem Beitrag hochgeladen wurden. Hier werden dann auch Bilder gefunden, die nicht in den Beitrag eingefügt, aber dazu hochgeladen wurden. Dafür wird als Wert für `post_parent` die ID des Beitrags übergeben. Für `post_status` soll derselbe Wert genutzt werden, den der Beitrag hat. Gesucht werden nur die Anhänge vom Typ `image`, die in aufsteigender Reihenfolge sortiert werden sollen.

Auch hier wird ein Array von Objekten zurückgegeben, von dem wieder nur das erste gebraucht wird. Nach Einsatz von `array_shift()` wird die ID des Objekts an `wp_get_attachment_image()` übergeben. Wenn das Ergebnis aus dem

[200] vgl. WordPress: Code Reference › Functions › attachment_url_to_postid

`attachment_url_to_postid()`-Aufruf nicht `0` ist, kann gleich die `wp_get_attachment_image()`-Funktion aufgerufen werden.

```
        echo $img_tag;
    }
}
```

Listing 64: Ende der hannover_image_from_gallery_or_image_post()

Abschließend wird das `img`-Element in der `$img_tag`-Variable ausgegeben, wie in Listing 64 dargestellt.

```
function hannover_get_gallery_images( $post_id ) {
    $post = get_post( $post_id );
    if ( ! $post || empty ( $post->post_content ) ) {
        return array();
    }
    $galleries = get_post_galleries( $post, false );
    if ( empty ( $galleries ) ) {
        return array();
    }
    $ids = array();
    foreach ( $galleries as $gallery ) {
        if ( ! empty ( $gallery['ids'] ) ) {
            $ids = array_merge( $ids, explode( ',', $gallery['ids'] ) );
        }
    }
    $ids = array_unique( $ids );
    if ( empty ( $ids ) ) {
        $attachments = get_children( array(
            'post_parent'    => $post_id,
            'post_status'    => 'inherit',
            'post_type'      => 'attachment',
            'post_mime_type' => 'image',
            'order'          => 'ASC',
            'orderby'        => 'menu_order',
        ) );
        if ( empty ( $attachments ) ) {
            return array();
        }
    }
    $images = get_posts(
        array(
            'post_type'      => 'attachment',
            'post_mime_type' => 'image',
            'orderby'        => 'post__in',
            'numberposts'    => 999,
            'include'        => $ids
        )
    );
    if ( ! $images && ! $attachments ) {
        return array();
    } elseif ( ! $images ) {
        $images = $attachments;
    }
```

```
    return $images;
}
```

Listing 65: Ermitteln der Galerie-Bilder in hannover_get_gallery_images()

In Listing 65 ist die Funktion abgebildet, die die Bilder aus den Galerien eines Galerie-Beitrags herausholt – zum Großteil entspricht die Funktion einer Lösung von Thomas Scholz, die leicht angepasst wurde. Zunächst werden mit `get_post()` die Daten des Beitrags zur übergebenen ID in der `$post`-Variable gespeichert.[201] Im Anschluss wird überprüft, ob der Beitrag nicht vorhanden oder leer ist. In beiden Fällen wird von der Funktion ein leeres Array zurückgegeben.

Mit `get_post_galleries()` können die Galerien aus einem Beitrag ermittelt werden.[202] Zurückgegeben wird von der Funktion ein Array, das für jeden Galerie-Shortcode wiederum ein Array enthält, welches die Daten zu den Bildern bereitstellt. Danach wird geprüft, ob keine Galerien gefunden wurden und in dem Fall ein leeres Array zurückgegeben. Nun werden nur die IDs der Bilder benötigt, die als Galerie eingefügt wurden – dafür wird ein leeres Array angelegt und danach die Galerien nacheinander in einer `foreach`-Schleife durchlaufen. Wenn der Wert zu dem Schlüssel `ids` nicht leer ist, werden die mit Komma getrennten IDs an den Kommata zu Arrays aufgespalten. Dafür wird die `explode()`-Funktion genutzt, die einen String an einem übergebenen Zeichen – in diesem Fall das Komma – auftrennt und die einzelnen Bestandteile als Werte eines Arrays zurückgibt.[203] Dieses Ergebnis-Array wird mit der `array_merge()`-Funktion an das bereits vorhandene Array `$ids` angehängt.[204]

Anschließend werden mit `array_unique()` alle doppelt vorhandenen IDs aus dem zusammengesetzten Array entfernt.[205] Falls das IDs-Array leer ist, wird mit derselben Vorgehensweise wie in Listing 63 versucht, Bilder zu ermitteln, die zu dem Beitrag hochgeladen wurden. Falls auch hier kein Ergebnis erzielt wird, wird ein leeres Array zurückgegeben. Um alle Informationen zu den Bildern zu bekommen, von denen die IDs ermittelt wurden, wird `get_posts()` aufgerufen.[206] In dem Argumente-Array wird als Post-Typ `attachment` angegeben, als Mime-Typ `image`. Sortiert werden sollen die Bilder in der Reihenfolge, in der die IDs aus der `$ids`-Variable dem `include`-Schlüssel übergeben werden – das entspricht der Reihenfolge, wie sie in dem Beitrag vorkommen. Wenn da-

[201] vgl. WordPress: Code Reference › Functions › get_post
[202] vgl. WordPress: Code Reference › Functions › get_post_galleries
[203] vgl. PHP Group: explode
[204] vgl. WordPress: Code Reference › Functions › array_merge
[205] vgl. WordPress: array_unique
[206] vgl. WordPress: Code Reference › Functions › get_posts

nach weder Werte für die Variable `$images` noch `$attachments` vorliegen, wird ein leeres Array zurückgegeben. Wenn nur `$images` leer ist, wurden keine Bild-Objekte aus dem `get_posts()`-Aufruf zurückgegeben und der `$images`-Variable wird der Wert von `$attachments` zugewiesen. Abschließend werden die Bilder zurückgegeben.

```
function hannover_get_first_image_from_post_content() {
   global $post;
   $first_img = '';
   $output    = preg_match( '/<img.+src=[\'"]([^\'"]+)[\'"].*>/i',
      $post->post_content, $matches );
   if ( ! empty( $matches[1] ) ) {
      $first_img = $matches[1];
   }

   return $first_img;
}
```

Listing 66: hannover_get_first_image_from_post_content() in functions.php

In Listing 66 ist die Funktion abgebildet, die die URL des ersten Bildes aus dem Beitragsinhalt extrahiert. Dafür wird zuerst das Post-Objekt zugänglich gemacht und eine leere Variable angelegt. Mit der Funktion `preg_match()` kann ein String nach dem ersten Vorkommen eines bestimmten Musters durchsucht werden – hier ein `img`-Element.[207] Die URL wird in dem regulären Ausdruck in Klammern geschrieben, sodass dieser Teil in der `$matches`-Variable gespeichert wird. Als zu durchsuchender String ist mit `$post->post_content` der Inhalt des Beitrags angegeben.

Im Anschluss wird überprüft, ob der Schlüssel `1` des Arrays einen Wert, nämlich die URL des ersten `img`-Elements, enthält. Ist das der Fall, wird dieser Wert in `$first_img` gespeichert und die URL zurückgegeben.

5.17.5 Portfolio-Elemente vom Blog ausschließen

Um die Portfolio-Elemente auszuschließen, muss in der Blog- und Archiv-Übersicht die Query angepasst werden – ungefähr in der Art und Weise, wie das in der `portfolio-page.php` geschehen ist. Dafür muss vor `if ( have_posts() ) {` aus Listing 14 auf Seite 33 der Code eingefügt werden, der nachfolgend schrittweise erläutert wird.

```
$exclude_portfolio_elements = get_theme_mod(
```

[207] vgl. PHP Group: preg_match

```
   'exclude_portfolio_elements_from_blog' );
if ( $exclude_portfolio_elements == 'checked' ) {
   $use_portfolio_category = get_theme_mod(
      'portfolio_from_category' );
   $portfolio_category     = get_theme_mod( 'portfolio_category' );
   $paged                  = ( get_query_var( 'paged' ) ) ?
                               get_query_var( 'paged' ) : 1;
   if ( $use_portfolio_category == 'checked'
      && $portfolio_category !== '' ) {
      $args = array(
         'category__not_in' => array( $portfolio_category ),
         'paged'            => $paged,
      );
   } else {
      $args = array(
         'paged'     => $paged,
         'tax_query' => array(
            array(
               'taxonomy' => 'post_format',
               'field'    => 'slug',
               'terms'    => array(
                  'post-format-gallery', 'post-format-image' ),
               'operator' => 'NOT IN'
            ),
         )
      );
   }
```

Listing 67: Vorbereitung der Index-Query ohne Portfolio-Objekte in der index.php

Listing 67 zeigt den Code, der die Argument-Arrays für `WP_Query` vorbereitet, wie es in etwa bereits aus Listing 57 von Seite 64 bekannt ist. Zuerst wird mit der Funktion `get_theme_mod()` und dem übergebenen Parameter der Wert der Checkbox ermittelt. Wenn die Checkbox angehakt ist, muss die Query verändert werden. Nach diesem Test werden die Werte benötigt, ob das Portfolio anhand einer Kategorie erstellt werden soll und welche Kategorie dafür ausgewählt ist. Der `$paged`-Variable wird der aus Listing 57 bekannte Wert zugewiesen, um die korrekte Funktion der Pagination zu gewährleisten.

Danach wird überprüft, ob eine Kategorie für die Portfolio-Elemente genutzt werden soll und ob eine Kategorie gewählt ist. Wenn das der Fall ist, wird an das Argument-Array neben dem bekannten Wert für `paged` die Kategorie-ID an den Schlüssel `category__not_in` übergeben. So werden alle Artikel ausgeschlossen, die der Kategorie angehören, die der Nutzer im Customizer für die Portfolio-Elemente ausgewählt hat. Wenn keine Kategorie gewählt ist oder die Portfolio-Elemente alle *Bild*- und *Galerie*-Beiträge sein sollen, kommt wieder der Schlüssel `tax_query` zum Einsatz. Neben den bereits bekannten Schlüsseln wird mit der Angabe von `NOT IN` für `operator` festgelegt, dass diese

Taxonomien nicht aufgeführt werden sollen. Damit sind die Beiträge mit den beiden entsprechenden Post-Formaten ausgeschlossen.

```
   $index_query = new WP_Query( $args );
   $temp_query  = $wp_query;
   $wp_query    = null;
   $wp_query    = $index_query;
   if ( $index_query->have_posts() ) {
      if ( is_home() && ! is_front_page() ) { ?>
         <header>
            <h1 class="page-title screen-reader-text">
               <?php single_post_title(); ?>
            </h1>
         </header>
      <?php }
      while ( $index_query->have_posts() ) {
         $index_query->the_post();
         get_template_part( 'template-parts/content',
            get_post_format() );
      }
   }
   wp_reset_postdata();
   the_posts_pagination( array( 'type' => 'list' ) );
   $wp_query = null;
   $wp_query = $temp_query;
} else {
```

Listing 68: Angepasste Query in der index.php für Ausgabe ohne Portfolio-Elemente

Im Anschluss wird die Query in Listing 68 nach demselben Prinzip umgesetzt, wie bereits in Listing 58 und Listing 59 auf Seite 66 beschrieben.

5.18 Archiv-Ansicht

5.18.1 Optionen für die Archiv-Funktion

Der Nutzer soll Elemente für das Archiv durch Zuweisen einer Kategorie festlegen können, wobei standardmäßig kein Archiv verwendet wird. Um die Optionen im Customizer umzusetzen, wird eine Lösung mit Radio-Buttons und abhängig von der Auswahl eingeblendeten Formular-Elementen umgesetzt, vom Prinzip ähnlich der Lösung für die Portfolio-Kategorie.

Radio-Buttons

```
$wp_customize->add_setting(
   'portfolio_archive', array(
      'default'           => 'no_archive',
```

```
      'sanitize_callback' => 'hannover_sanitize_select'
   )
);

$wp_customize->add_control(
   'portfolio_archive', array(
      'label'    => __(
         'Choose if you want to select
         portfolio elements for the archive.',
         'hannover'
      ),
      'type'     => 'radio',
      'section'  => 'portfolio_archive',
      'settings' => 'portfolio_archive',
      'choices'  => array(
         'no_archive'       => 'No archive',
         'archive_category' => 'Archive category',
      )
   )
);
```

Listing 69: Radio-Buttons im Customizer für das Portfolio-Archiv

In Listing 69 wird für die `add_setting()`-Methode diesmal mit `no_archive` auch ein Standard-Wert vergeben, damit ein Radio-Button ausgewählt ist. Als Sanitization-Callback kann dieselbe Funktion verwendet werden, wie für die Select-Liste. In `add_control()` wird als Typ `radio` angegeben und als Section `portfolio_archive`. An den Schlüssel `choices` werden als Array die zwei Radio-Buttons mit ihrer Beschriftung und dem Schlüssel übergeben. Der Code für die neue Section ist in Listing 70 zu sehen und vom Prinzip bereits aus Kapitel 5.17.1 bekannt.

```
$wp_customize->add_section(
   'portfolio_archive', array(
      'title' => __( 'Portfolio archive', 'hannover' ),
      'panel' => 'theme_options'
   )
);
```

Listing 70: Portfolio-Archive-Section im Customizer

Archiv-Kategorie

Die Einstellung für die Archiv-Kategorie ähnelt deutlich der für die Auswahl der Portfolio-Kategorie und ist in Listing 71 abgebildet.

```
$wp_customize->add_setting(
   'portfolio_archive_category', array(
      'sanitize_callback' => 'hannover_sanitize_select'
   )
);

$wp_customize->add_control(
```

```
    'portfolio_archive_category', array(
        'label'           => __( 'Archive category', 'hannover' ),
        'type'            => 'select',
        'section'         => 'portfolio_archive',
        'settings'        => 'portfolio_archive_category',
        'choices'         => $category_array,
        'active_callback' => 'hannover_choice_callback'
    )
);
```

Listing 71: Archiv-Kategorie im Customizer

Ein wichtiger Unterschied ist die angepasste Funktion für `active_callback`.

```
function hannover_choice_callback( $control ) {
    $radio_setting = $control->manager->
    get_setting( 'portfolio_archive' )->value();
    $control_id    = $control->id;
    if ( $control_id == 'portfolio_archive_category'
         && $radio_setting == 'archive_category' ) {
        return true;
    }

    return false;
}
```

Listing 72: hannover_choice_callback() in der customizer.php

In Listing 72 ist die Funktion zu sehen, die für die situationsabhängige Anzeige der Formularelemente für die Archiv-Kategorie zuständig ist. In `$radio_setting` wird der Wert des ausgewählten Radio-Buttons und in `$control_id` der eindeutige Bezeichner des Formularelements, von dem der Callback aufgerufen wurde, gespeichert. Anschließend wird in einer `if`-Bedingung geprüft, ob das Element für die Archiv-Kategorie den Callback aufgerufen hat und das entsprechende Radio-Element gewählt ist – in dem Fall wird `true` zurückgegeben, ansonsten `false`.

Zum Schluss soll dem Nutzer noch ermöglicht werden, äquivalent zu der Zahl von Portfolio-Elementen pro Seite auch die Anzahl von archivierten Elementen je Seite einzustellen. Dafür wird einfach der Code aus Listing 54 von Seite 62 übernommen und die Beschreibung etwas angepasst sowie der eindeutige Bezeichner von Setting und Control auf `portfolio_archive_elements_per_page` geändert – das Ergebnis ist in Listing 73 zu sehen.

```
$wp_customize->add_setting(
    'portfolio_archive_elements_per_page', array(
        'default'           => 0,
        'sanitize_callback' => 'absint'
    )
);
```

```
$wp_customize->add_control(
   'portfolio_archive_elements_per_page', array(
      'label'    => __( 'Number of archived portfolio elements to show on
         one page (0 to show all elements on one page).', 'hannover' ),
      'type'     => 'number',
      'section'  => 'portfolio_archive',
      'settings' => 'portfolio_archive_elements_per_page',
   )
);
```

Listing 73: Option für Beitrags-Anzahl in Archiv-Ansicht

5.18.2 Archiv-Elemente auf einer Seite anzeigen

Um die Archiv-Elemente auf einer Seite anzuzeigen, wird ein ähnliches Seiten-Template angelegt wie das, welches für die Portfolio-Übersicht erstellt wurde.

```
<?php
/**
 * Template Name: Portfolio archive page
 */
get_header(); ?>
   <main role="main">
      <header>
         <h1 class="page-title"><?php single_post_title(); ?></h1>
      </header>
      <?php $use_portfolio_category =
               get_theme_mod( 'portfolio_from_category' );
      $portfolio_category = get_theme_mod( 'portfolio_category' );
      $archive_type = get_theme_mod( 'portfolio_archive' );
      $archive_category = get_theme_mod(
                              'portfolio_archive_category' );
      $elements_per_page = get_theme_mod(
         'portfolio_archive_elements_per_page', 0 );
      $paged =
         ( get_query_var( 'paged' ) ) ? get_query_var( 'paged' ) : 1;
      $args = array();
      if ( $elements_per_page == 0 ) {
         $elements_per_page = - 1;
      }
```

Listing 74: Anfang der Archiv-Seite in der portfolio-archive-page.php

Listing 74 zeigt den Anfang der Archiv-Seite des Themes, das in der Datei `portfolio-archive-page.php` in dem `page-templates`-Ordner Platz findet. Zuerst kommt der Kommentar, damit WordPress das Seiten-Template erkennt. Anschließend werden der Header eingebunden und im `main`-Element die notwendigen Informationen aus dem Customizer geholt. Wenn als Anzahl an Elementen pro Seite `0` gesetzt ist, wird der Wert wieder in `-1` umgewandelt, damit alle Elemente auf einer Seite angezeigt werden.

```
if( $archive_category !== '' && $archive_type == 'archive_category' ){
   if ( $use_portfolio_category == 'checked'
         && $portfolio_category !== '' ) {
      $args = array(
         'posts_per_page' => $elements_per_page,
         'paged'          => $paged,
         'tax_query'      => array(
            'relation' => 'AND',
            array(
               'taxonomy' => 'category',
               'field'    => 'term_id',
               'terms'    => array( $portfolio_category ),
            ),
            array(
               'taxonomy' => 'category',
               'field'    => 'term_id',
               'terms'    => array( $archive_category ),
            ),
            array(
               'taxonomy' => 'post_format',
               'field'    => 'slug',
               'terms'    => array(
                  'post-format-gallery',
                  'post-format-image'
               ),
            ),
         )
      );
   } else {
```

Listing 75: Vorbereitung der Query-Argumente für das Archiv bei gewählter Portfolio-Kategorie

Nachdem in Listing 75 geprüft wurde, ob eine Archiv-Kategorie gewählt und der entsprechende Radio-Button ausgewählt wurde, wird zuerst der Fall behandelt, dass die Portfolio-Elemente ebenfalls über eine Kategorie ausgewählt werden. Dafür wird geprüft, ob eine Kategorie ausgewählt wurde und die Checkbox gesetzt ist. Wenn das der Fall ist, dürfen nur Elemente angezeigt werden, die sowohl der Portfolio-Kategorie, die in `$portfolio_category` gespeichert ist, als auch der Archiv-Kategorie aus `$archive_category` zugewiesen sind. Darüber hinaus müssen die Beiträge entweder vom Post-Format *Bild* oder *Galerie* sein. Um diese Bedingung kümmert sich wieder der `tax_query`-Schlüssel, dem für jede der Einzelbedingung ein Array übergeben wird, die mit `AND` verknüpft werden.

```
      $args = array(
         'posts_per_page' => $elements_per_page,
         'tax_query'      => array(
            'relation' => 'AND',
            array(
               'taxonomy' => 'category',
               'field'    => 'term_id',
               'terms'    => array( $archive_category ),
```

```
            ),
            array(
               'taxonomy' => 'post_format',
               'field'    => 'slug',
               'terms'    => array(
                  'post-format-gallery',
                  'post-format-image'
               ),
            ),
         ),
         'paged'          => $paged
      );
   }
}
```

Listing 76: Vorbereitung der Query-Argumente für das Archiv ohne Portfolio-Kategorie

Listing 76 zeigt die Vorbereitung für das Argument-Array, wenn keine Kategorie für die Portfolio-Elemente angegeben ist. In diesem Fall kann das Array für die Portfolio-Kategorie in dem `tax_query`-Schlüssel wegfallen, sodass alle Elemente angezeigt werden, die in der Archiv-Kategorie sind und das Format *Bild* oder *Galerie* haben.

```
      $archive_query = new WP_Query( $args );
      $temp_query    = $wp_query;
      $wp_query      = null;
      $wp_query      = $archive_query;
      if ( $archive_query->have_posts() ) {
         while ( $archive_query->have_posts() ) {
            $archive_query->the_post();
            get_template_part( 'template-parts/content',
               'portfolio-element' );
         }
      }
      wp_reset_postdata();
      the_posts_pagination( array( 'type' => 'list' ) );
      $wp_query = null;
      $wp_query = $temp_query; ?>
   </main>
<?php get_footer();
```

Listing 77: Loop für die Archiv-Ansicht

Der Code in Listing 77 entspricht dem aus Listing 58 und Listing 59 auf Seite 66 und gibt die Elemente aus.

5.18.3 Archiv-Elemente aus der Portfolio-Übersicht ausschließen

Um die Funktion des Archivs zu vervollständigen, dürfen Archiv-Elemente nicht auf der Portfolio-Seite angezeigt werden. Die Query muss demnach dahingehend verändert werden, dass Elemente, die der Archiv-Kategorie zugewiesen sind, nicht ausgegeben werden.

```
   <h1 class="page-title"><?php single_post_title(); ?></h1>
</header>
<?php $archive_type = get_theme_mod( 'portfolio_archive' );
$archive_category   = get_theme_mod( 'portfolio_archive_category' );
```

Listing 78: Ergänzung in der portfolio-page.php, um die Archiv-Customizer-Werte zu ermitteln

Zusätzlich zu den bereits ermittelten Customizer-Werten müssen am Anfang der `portfolio-page.php` noch die für das Archiv gespeichert werden. Dafür zuständig sind die beiden letzten Zeilen aus Listing 78, die ergänzt wurden – bereits vorhandener Code ist ausgegraut.

```
if ( $archive_type !== 'archive_category' ) {
   $archive_category = '';
}
if ( $elements_per_page == 0 ) {
   $elements_per_page = - 1;
```

Listing 79: Leeren der Archiv-Kategorie, wenn kein Archiv genutzt wird

Es kann vorkommen, dass ein Nutzer zwar den Radio-Button gewählt hat, kein Archiv anzulegen, obwohl er vorher eins hatte. In diesem Fall ist die Kategorie noch gespeichert. In Listing 79 wird geprüft, ob bei den Radio-Buttons etwas anderes gewählt ist als das Kategorie-Archiv und dann der eventuell vorhandene Wert der Variablen, die in Listing 78 die Archiv-Kategorie speichert, geleert.

```
'tax_query'      => array(
   'relation' => 'AND',
   array(
      'taxonomy' => 'category',
      'field'    => 'term_id',
      'terms'    => array( $archive_category ),
      'operator' => 'NOT IN'
   ),
   array(
      'taxonomy' => 'category',
      'field'    => 'term_id',
      'terms'    => array( $portfolio_category ),
   ),
```

Listing 80: Anpassung des Argument-Arrays für die Portfolio-Query mit Kategorie

In Listing 80 wird das Argument-Array für den Fall, dass die Portfolio-Elemente via Kategorie festgelegt werden, durch einen Zusatz innerhalb von `tax_query` erweitert. Durch die Angabe von `NOT IN` für `operator` werden alle Beiträge ausgeschlossen, die in der Kategorie für die Archive vorhanden sind.

5.19 Kategorie-Seiten für Portfolio-Elemente

Der Nutzer soll neben der Portfolio-Übersicht die Möglichkeit bekommen, Portfolio-Elemente in Kategorien aufzuteilen und diese auf je einer Seite anzuzeigen. Wenn der Nutzer beispielsweise Portfolio-Elemente der Kategorie *Sports* zugewiesen hat, dann soll es möglich sein, eine Seite zu erstellen, auf der nur diese Portfolio-Elemente angezeigt werden.

Auch diese Funktion wird mit einem Seiten-Template umgesetzt. Im Customizer werden automatisch für jede Seite, die dieses Seiten-Template hat, drei Einstellungen eingefügt.

5.19.1 Optionen für Portfolio-Seiten

Um jeweils ein Formularelement pro Seite mit dem entsprechenden Seiten-Template anzuzeigen, müssen zunächst diese Seiten ermittelt werden. In den nachfolgenden Code-Blöcken sind die bereits vorhandenen Zeilen wieder ausgegraut, die neuen hervorgehoben.

```
$wp_customize->add_control(
   'portfolio_archive_elements_per_page', array(
      'label'    => __( 'Number of archived portfolio elements to show on
         one page (0 to show all elements on one page).', 'hannover' ),
      'type'     => 'number',
      'section'  => 'portfolio_archive',
      'settings' => 'portfolio_archive_elements_per_page',
   )
);

$portfolio_category_pages = get_posts(
   array(
      'post_type'  => 'page',
      'meta_key'   => '_wp_page_template',
      'meta_value' => 'page-templates/portfolio-category-page.php'
   )
);
```

Listing 81: Ermitteln der Seiten mit dem Kategorie-Seiten-Template

In Listing 81 werden mit der `get_posts()`-Funktion die Seiten des entsprechenden Seiten-Templates ermittelt und in `$portfolio_category_pages` gespeichert.[208] Um das Ergebnis auf Seiten einzuschränken, wird als `post_type` der Wert `page` übergeben. Um das Seiten-Template zu prüfen, können die beiden Datenbankspalten `meta_key` und `meta_value` genutzt werden. In der Zeile mit dem `meta_key`-Wert `_wp_page_template` wird das Template der Seite gespeichert. Wenn kein spezielles Template gewählt ist, ist der dazugehörende Wert der `meta_value`-Spalte `standard`. Sonst entspricht der Wert dem Dateinamen des Seiten-Templates, gegebenenfalls mit dem Pfad, wie in diesem Fall: `page-templates/portfolio-category-page.php`.

```
if ( ! empty( $portfolio_category_pages ) ) {
   foreach ( $portfolio_category_pages as $portfolio_category_page ) {
      $id = $portfolio_category_page->ID;
      $wp_customize->add_setting(
         "portfolio_category_page_$id", array(
            'sanitize_callback' => 'hannover_sanitize_select'
         )
      );
```

Listing 82: Setting für jede Portfolio-Kategorie-Seite im Customizer

Listing 82 zeigt den Anfang der Schleife, die im Anschluss an Listing 81 für jede Seite mit dem entsprechenden Seiten-Template die Customizer-Einstellungen baut. Zuerst muss allerdings geprüft werden, ob das Ergebnis aus `get_posts()` nicht leer ist. Wenn das so ist, startet eine `foreach`-Schleife für jede der einzelnen Seiten. Darin wird die ID der Seite in der Variable `$id` gespeichert, damit sie später platzsparend ausgegeben werden kann. Mit der `add_setting()`-Methode wird, wie bereits aus früheren Kapiteln bekannt, eine Einstellung für den Customizer angelegt.[209] Damit die ID auch bei mehreren Seiten mit dem Seiten-Template eindeutig ist, wird an ein Präfix die ID der aktuellen Seite angehängt. Wichtig ist, dass die ID statt in eine einfache Prime in eine doppelte eingefasst wird, damit der PHP-Code ausgeführt wird.

```
/* translators: Label for portfolio category pages customizer control.
s=page title */
$label = sprintf(
   __( 'Choose portfolio category to show on “%s”', 'hannover' ),
   esc_html( $portfolio_category_page->post_title )
);
```

Listing 83: Generierung des Labels für die Kategorie-Auswahlliste

[208] vgl. WordPress: Code Reference › Functions › get_posts
[209] vgl. WordPress: Code Reference › Classes › WP_Customize_Manager › WP_Customize_Manager::add_setting

Damit der Nutzer genau weiß, für welche Seite er gerade die anzuzeigende Kategorie auswählt, muss die Beschriftung des `select`-Elements entsprechend dynamisch sein. Um die noch folgende `add_control()`-Methode übersichtlicher zu halten, wird die Beschriftung in Listing 83 generiert. Mit `sprintf()` wird der Platzhalter `%s` in der Beschriftung durch den Titel der Seite ersetzt, der mit `$portfolio_category_page->post_title` ermittelt wird.

```
$wp_customize->add_control(
   "portfolio_category_page_$id", array(
      'label'    => $label,
      'type'     => 'select',
      'section'  => 'portfolio_category_pages',
      'settings' => "portfolio_category_page_$id",
      'choices'  => $category_array,
   )
);
```

Listing 84: Hinzufügen des Formularelements für Kategorieauswahl

In Listing 84 wird mit der `add_control()`-Methode das Formular-Element für das bereits erstellte Setting eingefügt.[210] Um die Einzigartigkeit der ID zu sichern, wird vorgegangen wie in Listing 82 – nach derselben Art und Weise wird der Wert für `setting` gesetzt. Als Beschriftung wird der String aus Listing 83 angegeben, als Typ `select` und für den `choices`-Schlüssel kann das Array genutzt werden, das bereits für die Auswahl der Portfolio- und Archiv-Kategorie zum Einsatz gekommen ist.

Damit der Nutzer für jede der Seiten auch die Anzahl der anzuzeigenden Elemente pro Seite festlegen kann, wird innerhalb der Schleife des Weiteren ein Zahlen-Element erstellt.

```
      $wp_customize->add_setting(
         "portfolio_category_page_elements_per_page_$id", array(
            'default'           => 0,
            'sanitize_callback' => 'absint'
         )
      );

      $wp_customize->add_control(
         "portfolio_category_page_elements_per_page_$id", array(
            'label'    => __( 'Number of elements to show on one page
                (0 to show all).', 'hannover' ),
            'type'     => 'number',
            'section'  => 'portfolio_category_pages',
            'settings' =>
               "portfolio_category_page_elements_per_page_$id",
         )
```

[210] vgl. WordPress: Code Reference › Classes › WP_Customize_Manager › WP_Customize_Manager::add_control

```
        );
    }
}
```

Listing 85: Erstellen des Zahlen-Elements für die Anzahl auf Portfolio-Kategorie-Seiten

Listing 85 zeigt den dafür verantwortlichen Code. Für die Eindeutigkeit wird wieder die ID der Seite genutzt, ansonsten gleicht er dem Code aus Listing 54 auf Seite 62.

Als Section wird für beide Controls `portfolio_category_pages` angegeben, die in Listing 86 nach bekanntem Muster erstellt wird. Neu ist hier lediglich der Array-Schlüssel `description`, an den eine Beschreibung übergeben werden kann, die oberhalb der Elemente aus der Section angezeigt wird – dieser Code wird außerhalb der Schleife notiert.

```
$wp_customize->add_section(
    'portfolio_category_pages', array(
        'title'       => __( 'Portfolio category pages', 'hannover' ),
        'description' => __(
            'Here you can choose the category to display on the
            respective portfolio category page. The category must
            include portfolio elements–otherwise all portfolio elements
            will be displayed.', 'hannover' ),
        'panel'       => 'theme_options'
    )
);
```

Listing 86: Erstellung der Customizer-Section portfolio_category_pages

5.19.2 Portfolio-Elemente einer Kategorie anzeigen

Die Datei für das Seiten-Template heißt `portfolio-category-page.php` und kommt zu den bereits erstellten Templates in den Ordner `page-templates`.

```
<?php
/**
 * Template Name: Portfolio category page
 */
get_header(); ?>
    <main role="main">
        <header>
            <h1 class="page-title"><?php single_post_title(); ?></h1>
        </header>
        <?php $page_id = $post->ID;
        $archive_type = get_theme_mod( 'portfolio_archive' );
        $archive_category = get_theme_mod('portfolio_archive_category');
        $use_portfolio_category = get_theme_mod(
            'portfolio_from_category' );
        $portfolio_category = get_theme_mod( 'portfolio_category' );
        $portfolio_category_page_category = get_theme_mod(
            "portfolio_category_page_$page_id" );
        $elements_per_page = get_theme_mod(
```

```
        "portfolio_category_page_elements_per_page_$page_id", 0 );
    $paged = (get_query_var('paged')) ? get_query_var('paged') : 1;
    if ( $archive_type !== 'archive_category' ) {
        $archive_category = '';
    }
    if ( $elements_per_page == 0 ) {
        $elements_per_page = - 1;
    }
```

Listing 87: Anfang der portfolio-category-page.php

Listing 87 zeigt den Anfang der `portfolio-category-page.php`. Zunächst wird die ID der Seite ermittelt und in `$page_id` gespeichert. Danach werden die notwendigen Werte aus dem Customizer abgerufen – neu im Vergleich zu Listing 74 auf Seite 77 sind hier lediglich die Werte für die Kategorie, die auf der Seite angezeigt werden soll, sowie die Elemente pro Seite. Bei beiden wird mit dem Zusatz `$page_id` der Wert aus dem Customizer geholt, der zur aktuellen Seite gehört. Auch die übrige Datei ähnelt stark der für die Portfolio-Übersicht. Neu ist lediglich, dass in den `tax_query`-Bereichen der Argument-Arrays eine Bedingung hinzukommt, nämlich die Kategorie, die der Nutzer für die aktuelle Seite im Customizer gewählt hat.

```
if ( $use_portfolio_category == 'checked' &&
    $portfolio_category !== '' ) {
    $args = array(
        'posts_per_page' => $elements_per_page,
        'paged'          => $paged,
        'tax_query'      => array(
            'relation' => 'AND',
            array(
                'taxonomy' => 'category',
                'field'    => 'term_id',
                'terms'    => array( $archive_category ),
                'operator' => 'NOT IN'
            ),
            array(
                'taxonomy' => 'category',
                'field'    => 'term_id',
                'terms'    => array( $portfolio_category ),
            ),
            array(
                'taxonomy' => 'category',
                'field'    => 'term_id',
                'terms'    => array( $portfolio_category_page_category ),
            ),
            array(
                'taxonomy' => 'post_format',
                'field'    => 'slug',
                'terms'    => array(
                    'post-format-gallery',
                    'post-format-image'
                ),
            ),
```

```
        )
    );
```

Listing 88: Argument-Array mit Portfolio-Elementen aus einer Kategorie in der portfolio-category-page.php

In Listing 88 ist damit im Vergleich zu dem Array aus der `portfolio-page.php` lediglich der fett gedruckte Teil dazugekommen – der Rest ist bereits bekannt.

```
} else {
    $args = array(
        'posts_per_page' => $elements_per_page,
        'paged'          => $paged,
        'tax_query'      => array(
            'relation' => 'AND',
            array(
                'taxonomy' => 'category',
                'field'    => 'term_id',
                'terms'    => array( $archive_category ),
                'operator' => 'NOT IN'
            ),
            array(
                'taxonomy' => 'category',
                'field'    => 'term_id',
                'terms'    => array( $portfolio_category_page_category ),
            ),
            array(
                'taxonomy' => 'post_format',
                'field'    => 'slug',
                'terms'    => array(
                    'post-format-gallery',
                    'post-format-image'
                ),
            ),
        )
    );
}
```

Listing 89: Argument-Array ohne Portfolio-Kategorie in der portfolio-category-page.php

Ähnlich sieht es in Listing 89 aus – auch hier ist das Argument-Array identisch mit dem aus der Portfolio-Übersicht, lediglich der Teil für die Seiten-Kategorie ist dazugekommen.

```
        $portfolio_query = new WP_Query( $args );
        $temp_query      = $wp_query;
        $wp_query        = null;
        $wp_query        = $portfolio_query;
        if ( $portfolio_query->have_posts() ) {
            while ( $portfolio_query->have_posts() ) {
                $portfolio_query->the_post();
                get_template_part( 'template-parts/content',
                    'portfolio-element' );
            }
        }
        wp_reset_postdata();
        the_posts_pagination( array( 'type' => 'list' ) );
```

```
        $wp_query = null;
        $wp_query = $temp_query; ?>
    </main>
<?php get_footer();
```

Listing 90: Angepasste Query in der portfolio-category-page.php

Auch Listing 90 hält keine Neuerungen bereit. Die Query entspricht Eins zu Eins der aus Listing 58 und Listing 59 von Seite 66.

5.20 Alternatives Layout für Portfolio-Elemente

Ein anfangs formuliertes Ziel war, dass der Nutzer das Layout der Portfolio-Elemente in der Übersicht anpassen kann. Neben der bereits realisierten Version, dass nur das erste Bild angezeigt wird, soll eine weitere Layout-Version neben dem Bild zusätzlich den Titel und eine Beschreibung anzeigen.

Dafür muss der Nutzer zuerst im Customizer die Möglichkeit haben, das alternative Layout auszuwählen. Diese Möglichkeit soll ihm unabhängig voneinander für die Portfolio-Übersicht, die Archiv-Ansicht und jede Kategorie-Seite gegeben werden. Als Formularelement wird in allen Fällen eine einfache Checkbox eingesetzt.

5.20.1 Alternatives Layout in Portfolio-Übersicht

```
$wp_customize->add_setting(
    'portfolio_alt_layout', array(
        'sanitize_callback' => 'hannover_sanitize_checkbox'
    )
);

$wp_customize->add_control(
    'portfolio_alt_layout', array(
        'label'    => __( 'Use alternative layout for portfolio
            overview', 'hannover' ),
        'type'     => 'checkbox',
        'section'  => 'portfolio_elements',
        'settings' => 'portfolio_alt_layout'
    )
);

$wp_customize->add_setting(
    'portfolio_archive', array(
```

Listing 91: Option für alternatives Layout auf Portfolio-Übersicht

In Listing 91 ist der Code abgebildet, der die Einstellung für den Layout-Wechsel in der Portfolio-Übersicht im Customizer erstellt, der fast identisch mit dem ersten Teil aus Listing 48 von Seite 57 ist. Angepasst sind hier lediglich die ID und Beschriftung.

Innerhalb der `portfolio-page.php` muss nur ein geringfügiger Teil angepasst werden, um das andere Layout der Portfolio-Elemente anzuzeigen, wenn der Nutzer die Checkbox angehakt hat.

```
$wp_query              = $portfolio_query;
$portfolio_alt_layout = get_theme_mod( 'portfolio_alt_layout' );
if ( $portfolio_query->have_posts() ) {
   while ( $portfolio_query->have_posts() ) {
      $portfolio_query->the_post();
      if ( $portfolio_alt_layout == 'checked' ) {
         get_template_part( 'template-parts/content',
            'portfolio-element-alt' );
      } else {
         get_template_part( 'template-parts/content',
            'portfolio-element' );
      }
   }
}
```

Listing 92: Angepasste Loop für alternatives Layout der Elemente in der Portfolio-Übersicht

In Listing 92 sind die neu hinzugekommenen Code-Zeilen bunt dargestellt, die bereits vorhandenen ausgegraut. Vor dem Start der Loop muss mit der Funktion `get_theme_mod()` der Wert der `portfolio_alt_layout`-Einstellung ermittelt werden.[211] Im Anschluss wird innerhalb der Loop mit einer `if`-Bedingung abgefragt, ob die Checkbox angehakt ist. Wenn das der Fall ist, wird die noch zu erstellende `content-portfolio-element-alt.php` aus dem `template-parts`-Ordner eingefügt, andernfalls die Datei, die nur das Bild anzeigt.

In dem alternativen Layout sollen neben dem ersten Bild auch der Titel sowie der Auszug sichtbar sein, der vom Autor in der Bearbeitungsansicht eines Beitrags festgelegt werden kann. Wenn keiner angelegt wurde, werden lediglich das Bild und der Titel angezeigt.

```
<article <?php post_class( 'element-alt' ); ?>
   id="post-<?php the_ID(); ?>">
   <a href="<?php the_permalink(); ?>">
      <?php hannover_image_from_gallery_or_image_post( 'medium_large',
         $post ); ?>
   </a>
   <div>
      <a href="<?php the_permalink(); ?>">
```

[211] vgl. WordPress: Code Reference › Functions › get_theme_mod

```
            <h2><?php the_title(); ?></h2>
        </a>
        <?php if ( has_excerpt() ) {
            the_excerpt();
        } ?>
    </div>
</article>
```

Listing 93: Alternatives Layout in der content-portfolio-element-alt.php

In Listing 93 ist der gesamte Code dieser Datei zu sehen. Innerhalb des Links wird mit `hannover_image_from_gallery_or_image_post()` das erste Bild ausgegeben, was bereits ab Listing 61 auf Seite 67 erläutert wurde. Als Größe wird hier `medium_large` angegeben, die seit WordPress 4.4 verfügbar ist. Im Anschluss werden innerhalb eines `div`-Elements der verlinkte Titel sowie der Auszug ausgegeben – mit `has_excerpt()` wird geprüft, ob ein Auszug vorhanden ist und mit `the_excerpt()` wird er angezeigt.[212, 213]

5.20.2 Alternatives Layout im Archiv

Die Integration der Einstellungsmöglichkeit für das alternative Layout in der Archiv-Ansicht verläuft ähnlich wie bei der Portfolio-Übersicht.

```
$wp_customize->add_setting(
    'portfolio_archive_alt_layout', array(
        'sanitize_callback' => 'hannover_sanitize_checkbox'
    )
);

$wp_customize->add_control(
    'portfolio_archive_alt_layout', array(
        'label'    => __( 'Use alternative layout for archive',
            'hannover' ),
        'type'     => 'checkbox',
        'section'  => 'portfolio_archive',
        'settings' => 'portfolio_archive_alt_layout'
    )
);

$portfolio_category_pages = get_posts(
    array(
```

Listing 94: Option für alternatives Layout auf der Archiv-Seite

[212] vgl. WordPress: Code Reference › Functions › has_excerpt
[213] vgl. WordPress: Code Reference › Functions › the_excerpt

Listing 94 zeigt den Code für die Customizer-Option, die der aus Listing 91 gleicht. Auch die Anpassung in der `portfolio-archive-page.php`, die in Listing 95 abgebildet ist, zeigt nichts Neues.

```
$archive_alt_layout = get_theme_mod( 'portfolio_archive_alt_layout' );
if ( $archive_query->have_posts() ) {
   while ( $archive_query->have_posts() ) {
      $archive_query->the_post();
      if ( $archive_alt_layout == 'checked' ) {
         get_template_part( 'template-parts/content', 'portfolio-element-
alt' );
      } else {
         get_template_part( 'template-parts/content',
            'portfolio-element' );
      }
   }
}
```

Listing 95: Angepasste Loop für alternatives Layout des Archivs

5.20.3 Alternatives Layout für Portfolio-Kategorie-Seiten

Auch für jede einzelne Seite vom Typ *Portfolio category page* soll der Nutzer das alternative Layout aktivieren können. Dafür muss die zusätzliche Option in die `foreach`-Schleife aus Listing 82 von Seite 82 integriert werden.

```
$wp_customize->add_setting(
   "portfolio_category_page_alt_layout_$id", array(
      'sanitize_callback' => 'hannover_sanitize_checkbox'
   )
);

$wp_customize->add_control(
   "portfolio_category_page_alt_layout_$id", array(
      'label'    => __( 'Use alternative layout', 'hannover' ),
      'type'     => 'checkbox',
      'section'  => 'portfolio_category_pages',
      'settings' => "portfolio_category_page_alt_layout_$id"
   )
);
```

Listing 96: Option für alternatives Layout auf den Kategorie-Seiten

Der Code aus Listing 96 steht am Ende dieser Schleife und macht im Prinzip nichts anderes als die beiden Optionen, die bereits besprochen wurden. Mit der Seiten-ID, die in `$id` gespeichert ist, wird nach demselben Schema wie bei den anderen Settings und Controls aus der Schleife eine eindeutige ID erzeugt.

Auch die Wahl der richtigen Template-Datei in der Seiten-Template-Datei, die in Listing 97 abgebildet ist, ist identisch mit der aus den vorangehenden zwei Unterkapiteln.

```
$portfolio_category_alt_layout = get_theme_mod(
"portfolio_category_page_alt_layout_$page_id" );
if ( $portfolio_query->have_posts() ) {
   while ( $portfolio_query->have_posts() ) {
      $portfolio_query->the_post();
      if ( $portfolio_category_alt_layout == 'checked' ) {
         get_template_part( 'template-parts/content',
            'portfolio-element-alt' );
      } else {
         get_template_part( 'template-parts/content',
            'portfolio-element' );
      }
   }
}
```

Listing 97: Angepasste Loop für alternatives Layout der Portfolio-Kategorie-Seiten

5.21 Einzelansicht von Portfolio-Elementen

Die Portfolio-Elemente sind Beiträge und werden mit der `single.php` dargestellt. Es soll allerdings auf der Einzelansicht keine Sidebar angezeigt werden und auch keine Navigation zum nächsten und vorherigen Beitrag, wenn die Portfolio-Elemente vom Blog ausgeschlossen sind. Die Datei `single.php` muss dafür etwas angepasst werden.

```
<?php get_header();
$portfolio_post = false;
$format = get_post_format( $post_id );
if ( $format == 'gallery' || $format == 'image' ) {
   $use_portfolio_category = get_theme_mod(
      'portfolio_from_category' );
   if ( $use_portfolio_category == 'checked' ) {
      $portfolio_category = get_theme_mod( 'portfolio_category' );
      if ( has_category( $portfolio_category, $post ) ) {
         $portfolio_post = true;
      }
   } else {
      $portfolio_post = true;
   }
} ?>
   <main role="main"<?php if ( $portfolio_post ) {
      echo ' class="portfolio-element"';
   } ?>>
      <?php while ( have_posts() ) {
         the_post();
         if ( $portfolio_post ) {
            $exclude_portfolio_elements = get_theme_mod(
               'exclude_portfolio_elements_from_blog' );
            get_template_part(
```

```
                'template-parts/content-single-portfolio' );
            if ( $exclude_portfolio_elements != 'checked' ) {
                the_post_navigation();
            }
        } else {
            get_template_part(
                'template-parts/content-single', get_post_format() );
            the_post_navigation();
        }
        if ( comments_open() || get_comments_number() ) {
            comments_template( '', true );
        }
    } ?>
</main>
<?php if ( ! $portfolio_post ) {
    get_sidebar();
}
get_footer();
```

Listing 98: Anpassung der single.php für die Einzelansicht von Portfolio-Beiträgen

Zunächst muss in Listing 98 ermittelt werden, ob es sich bei dem angezeigten Beitrag um ein Portfolio-Element handelt. Dafür wird zuerst eine Variable mit dem booleschen Wert `false` gefüllt, die später – sollte es sich um ein Portfolio-Element handeln – auf `true` gesetzt wird. Darüber hinaus wird das Post-Format des Beitrags über die `get_post_format()`-Funktion ermittelt, der das Post-Objekt übergeben wird.[214]

Wenn es sich um einen Galerie- oder Bild-Beitrag handelt, wird mit `get_theme_mod()` der Wert der Portfolio-Kategorie-Checkbox ermittelt – als Wert wird die ID des Settings übergeben.[215] Wenn die Checkbox angehakt ist, wird die ID der Kategorie ermittelt und mit der `has_category()`-Funktion überprüft, ob der aktuelle Beitrag dieser Kategorie zugewiesen ist – dafür wird als erster Parameter die ID und als zweiter das Post-Objekt übergeben.[216] Ist die Bedingung wahr, wird der Wert von `$portfolio_post` auf `true` geändert.

Wenn das Portfolio nicht anhand einer Kategorie gebildet wird, es sich bei dem Beitrag aber um einen Bild- oder Galerie-Post handelt, wird die Variable ohne weitere Prüfungen auf `true` gesetzt. Damit die Einzelansicht später per CSS anders angesprochen werden kann, wird dem `main`-Element die Klasse `portfolio-element` mitgegeben, wenn `$portfolio_post` den Wert `true` enthält.

[214] vgl. WordPress: Code Reference › Functions › get_post_format
[215] vgl. WordPress: Code Reference › Functions › get_theme_mod
[216] vgl. WordPress: Code Reference › Functions › has_category

Statt innerhalb der Schleife nun einfach die normale Datei für die Einzelansicht einzubinden, wird überprüft, ob es sich um ein Portfolio-Element handelt und in dem Fall ermittelt, ob die Portfolio-Elemente vom Blog ausgeschlossen werden sollen. Danach wird die `content-single-portfolio.php` aus dem `template-parts`-Ordner eingebunden. Wenn die Blog-Beiträge *nicht* von den Portfolio-Elementen getrennt sind, wird mit `the_post_navigation()` die Navigation zum vorherigen und nächsten Beitrag ausgegeben.[217] Wenn der Beitrag kein Portfolio-Element ist, wird die `content-single.php` eingebunden.

Die letzte Anpassung ist im Bereich der Sidebar zu finden: Diese wird nur dann eingebunden, wenn es sich bei dem Beitrag nicht um ein Portfolio-Element handelt.

```
<article <?php post_class(); ?> id="post-<?php the_ID(); ?>">
   <header class="entry-header">
      <?php hannover_the_title( 'h1', false ); ?>
      <span><?php hannover_the_date(); ?></span>
   </header>
   <div class="entry-content">
      <?php hannover_the_content();
      wp_link_pages(); ?>
   </div>
</article>
```

Listing 99: content-single-portfolio.php

Die `content-single-portfolio.php` ähnelt der `content-single.php` aus Listing 33 auf Seite 48. Der einzige Unterschied ist, dass weder das Beitragsbild noch der Footer eingebunden werden.

5.22 Portfolio- und Archiv-Kategorien vom Widget ausschließen

Eventuell möchten die Nutzer die Kategorien für Archiv und Portfolio nicht in dem Kategorie-Widget angezeigt haben – vor allem dann nicht, wenn sie die Portfolio-Elemente vom Blog ausschließen. Für beide Kategorien wird jeweils eine Option in Form einer Checkbox im Customizer angelegt.

[217] vgl. WordPress: Code Reference › Functions › the_post_navigation

5.22.1 Optionen für Ausschluss der Portfolio- und Archiv-Kategorie im Customizer

```
$wp_customize->add_setting(
   'portfolio_remove_category_from_cat_widget', array(
      'sanitize_callback' => 'hannover_sanitize_checkbox'
   )
);

$wp_customize->add_control(
   'portfolio_remove_category_from_cat_list', array(
      'label' => __( 'Remove portfolio category from category widget',
         'hannover' ),
      'type' => 'checkbox',
      'section' => 'portfolio_elements',
      'settings' => 'portfolio_remove_category_from_cat_widget',
      'active_callback' => 'hannover_use_portfolio_category_callback'
   )
);

$wp_customize->add_setting(
   'exclude_portfolio_elements_from_blog', array(
```

Listing 100: Option zum Ausschluss der Portfolio-Kategorie

In Listing 100 ist der Code abgebildet, der die Checkbox für den Ausschluss der Portfolio-Kategorie in den Customizer einfügt. Bei Listing 48 auf Seite 57 ist die Nutzung von den Methoden `add_setting()` und `add_control()` bereits näher erläutert worden. Die Callback-Funktion, die dafür sorgt, dass die Option nur angezeigt wird, wenn das Portfolio mit einer Kategorie gebildet werden soll, ist in Listing 52 auf Seite 60 näher beschrieben.

```
$wp_customize->add_setting(
   'portfolio_archive_remove_category_from_cat_widget', array(
      'sanitize_callback' => 'hannover_sanitize_checkbox'
   )
);

$wp_customize->add_control(
   'portfolio_archive_remove_category_from_cat_widget', array(
      'label' => __( 'Remove archive category from category widget',
         'hannover' ),
      'type' => 'checkbox',
      'section' => 'portfolio_archive',
      'settings' =>
         'portfolio_archive_remove_category_from_cat_widget',
      'active_callback' => 'hannover_choice_callback'
   )
);

$wp_customize->add_setting(
   'portfolio_archive_elements_per_page', array(
```

Listing 101: Option zum Ausschluss der Archiv-Kategorie

Der Code aus Listing 101 ist dem aus Listing 100 recht ähnlich. Die Option wird aber in einer anderen Section dargestellt und auch die Callback-Funktion ist nicht dieselbe. Diese in Listing 72 auf Seite 76 erläuterte Funktion muss um einen Teil erweitert werden. Nach dem gleichen Verfahren wie für die andere Option wird in Listing 102 geprüft, ob die Control diejenige zum Ausschluss der Archiv-Kategorie ist und ob der Radio-Button der für die Archiv-Kategorie ist.

```
    return true;
} elseif (
    $control_id == 'portfolio_archive_remove_category_from_cat_widget'
        && $radio_setting == 'archive_category' ) {
    return true;
}

return false;
```

Listing 102: Erweiterung der hannover_choice_callback()-Funktion

5.22.2 Entfernen der Kategorien aus dem Widget

Um die Argumente für die Anzeige des Kategorie-Widgets zu filtern, gibt es den Hook `widget_categories_args`.[218] Für das Ausschließend von Kategorien müssen dem Argument-Array, das dieser Filter verändern kann, die ID oder IDs mit dem Schlüssel `exclude` übergeben werden, der standardmäßig nicht vorhanden ist.

```
function hannover_filter_category_widget( $cat_args ) {
    $use_portfolio_category = get_theme_mod(
        'portfolio_from_category' );
    $archive_type = get_theme_mod( 'portfolio_archive' );
    $exclude_portfolio_cat_from_widget = get_theme_mod(
        'portfolio_remove_category_from_cat_widget' );
    $exclude_archive_cat_from_widget = get_theme_mod(
        'portfolio_archive_remove_category_from_cat_widget' );
    $exclude = '';
    if ( $exclude_portfolio_cat_from_widget == 'checked'
        && $use_portfolio_category == 'checked' ) {
        $portfolio_category = get_theme_mod( 'portfolio_category' );
        $exclude            = $portfolio_category;
    }
    if ( $exclude_archive_cat_from_widget == 'checked'
        && $archive_type == 'archive_category' ) {
        $archive_category = get_theme_mod(
            'portfolio_archive_category' );
        if ( $exclude != '' ) {
            $exclude .= ', ' . $archive_category;
        } else {
            $exclude = $archive_category;
        }
```

[218] vgl. WordPress: Code Reference › Hooks › widget_categories_args

```
    }
    $cat_args['exclude'] = $exclude;

    return $cat_args;
}

add_filter( 'widget_categories_args',
    'hannover_filter_category_widget' );
```

Listing 103: hannover_filter_category_widget() in der functions.php

Listing 103 zeigt den kompletten Code, der für die Filterung verantwortlich ist. Ganz unten ist zu sehen, wie die Funktion an den entsprechenden Filter angehängt wird. Dadurch hat die Funktion `hannover_filter_category_widget()` über den Parameter `$cat_args` Zugriff auf die Argumente, die für die Anzeige des Widgets herangezogen werden und kann diese verändern. Zunächst muss in der Funktion der Zustand der Customizer-Einstellungen ermittelt werden – dafür wird, wie bereits häufiger verwendet, `get_theme_mod()` eingesetzt, an die der Bezeichner des Settings übergeben werden muss.[219] Neben den Werten für den Ausschluss der jeweiligen Kategorie wird auch die Einstellung benötigt, ob überhaupt jeweils eine Kategorie genutzt werden soll. Es kann beispielsweise sein, dass zwar noch der Haken gesetzt ist, um die Portfolio-Kategorie auszuschließen, aber nicht der Haken, dass das Portfolio durch eine Kategorie erzeugt werden soll.

Im Anschluss wird die Variable `$exclude` mit einem leeren String befüllt und geprüft, ob der Haken für das Entfernen der Portfolio-Kategorie vom Widget gesetzt ist und eine Portfolio-Kategorie genutzt werden soll. Ist das der Fall, wird wiederum mit `get_theme_mod()` die ID der Kategorie ermittelt, die für das Portfolio gewählt ist, und danach der leeren `$exclude`-Variable zugewiesen.

Danach wird überprüft, ob die Archiv-Kategorie entfernt werden soll und ein Archiv angelegt ist. Auch hier muss die ID der Archiv-Kategorie ermittelt und unterschieden werden, ob `$exclude` bereits die Kategorie vom Portfolio enthält oder nicht. Wenn `$exclude` nicht leer ist, also eine Portfolio-Kategorie enthält, dann muss vor der Archiv-Kategorie-ID ein Komma eingefügt werden. Wenn die Variable leer ist, kann einfach die ID zugewiesen werden.

Abschließend wird der Wert von `$exclude` dem `exclude`-Schlüssel des Argument-Arrays zugewiesen und dieses zurückgegeben.

[219] vgl. WordPress: Code Reference › Functions › get_theme_mod

5.23 Seiten-Template ohne Sidebar

In Kapitel 5.13 wurde die `page.php` erstellt, die unter anderem eine Sidebar einbindet. Der Nutzer soll aber auch die Möglichkeit bekommen, eine Seite ohne Sidebar zu erstellen, auf der der Inhaltsbereich dann zentriert ist. Um dieses Verhalten umzusetzen, gleicht das Seiten-Template `no-sidebar-page.php` in Listing 104 der `page.php` aus Listing 30 auf Seite 46 bis auf den fehlenden Aufruf von `get_sidebar()`.[220]

```
<?php
/**
 * Template Name: Page without Sidebar
 */
get_header(); ?>
   <main role="main">
      <?php while ( have_posts() ) {
         the_post();
         get_template_part( 'template-parts/content', 'page' );
         if ( comments_open() || get_comments_number() ) {
            comments_template( '', true );
         }
      } ?>
   </main>
<?php get_footer();
```

Listing 104: no-sidebar-page.php

5.24 Seiten-Templates für die Startseite

Auf der Startseite soll es möglich sein, einen Slider oder ein zufälliges Bild anzuzeigen. Am einfachsten ist das mit zwei Seiten-Templates umzusetzen, die jeweils mit einer Galerie arbeiten, welche in die Seite eingefügt wurde. Im Fall des Sliders werden alle Galerie-Bilder zurückgegeben und als Slider dargestellt, im Fall des zufälligen Bildes wird ein Bild aus dem Galerie-Shortcode ausgewählt und angezeigt.

5.24.1 Slider auf der Startseite

Um den Slider umzusetzen, wird das Skript *Owl Carousel* verwendet, das auf *jQuery* basiert.[221] Da jQuery von WordPress eingebunden wird, wenn ein anderes Skript es voraussetzt, muss nur Owl Carousel nachgerüstet werden. Der Nutzer soll dabei angeben können,

[220] vgl. WordPress: Code Reference › Functions › get_sidebar
[221] vgl. Wojciechowski: Owl Carousel

ob der Slider automatisch durchläuft und wie lange die Bilder jeweils angezeigt werden sollen.

```
$wp_customize->add_section(
   'portfolio_elements', array(
      'title' => __( 'Slider settings', 'hannover' ),
      'panel' => 'theme_options'
   )
);
```

Listing 105: Customizer-Section für Slider-Einstellungen

Diese beiden Optionen werden erneut im Customizer umgesetzt, wofür in Listing 105 eine neue Section angelegt wird. Dieser Code kommt ganz ans Ende der Funktion `hannover_customize_register()` in der `customizer.php`.

Für die Option des automatischen Abspielens der eingesetzten Slider wird in Listing 106 eine einfache Checkbox, wie bereits aus Listing 48 auf Seite 57 bekannt, realisiert. Zugewiesen wird die Control der neuen Section `slider_settings`.

```
$wp_customize->add_setting(
   'slider_autoplay', array(
      'sanitize_callback' => 'hannover_sanitize_checkbox',
   )
);

$wp_customize->add_control(
   'slider_autoplay', array(
      'label'    => __( 'Enable autoplay', 'hannover' ),
      'type'     => 'checkbox',
      'section'  => 'slider_settings',
      'settings' => 'slider_autoplay'
   )
);

$wp_customize->add_panel(
   'theme_options', array(
```

Listing 106: Option zur Aktivierung des Auoplays für Slider

Damit der Nutzer die Zeitspanne einstellen kann, die ein Bild beim Autoplay eingeblendet sein soll, kommt ein Formularelement von Typ `number` zum Einsatz. Als Standard wird in Listing 107 `3000` festgelegt, was drei Sekunden entspricht. Ansonsten ähnelt der Code stark dem Listing 54 auf Seite 62.

```
      'settings' => 'slider_autoplay'
   )
);

$wp_customize->add_setting(
```

```
    'slider_autoplay_time', array(
        'default'           => 3000,
        'sanitize_callback' => 'absint'
    )
);

$wp_customize->add_control(
    'slider_autoplay_time', array(
        'label'    => __( 'Time in milliseconds to show each image with
            autoplay', 'hannover' ),
        'type'     => 'number',
        'section'  => 'slider_settings',
        'settings' => 'slider_autoplay_time',
    )
);
```

Listing 107: Option zur Festlegung der Zeit pro Bild beim Autoplay

Das Seiten-Template sieht fast genauso aus, wie die `no-sidebar-page.php` und ist in Listing 108 abgebildet. Ein wichtiger Unterschied ist, dass hier nicht die `content-page.php` eingebunden wird, sondern deren Inhalt ohne die Ausgabe des Headers direkt in die Datei eingefügt wird, damit der Seitentitel nicht angezeigt wird.

```
<?php
/**
 * Template Name: Front page with slider
 */
get_header(); ?>
    <main role="main">
        <?php if ( have_posts() ) {
            while ( have_posts() ) {
                the_post(); ?>
                <article <?php post_class(); ?> id="post-<?php the_ID(); ?>">
                    <div class="entry-content">
                        <?php hannover_the_content();
                        wp_link_pages(); ?>
                    </div>
                </article>
                <?php if ( comments_open() || get_comments_number() ) {
                    comments_template( '', true );
                }
            }
        } else {
            get_template_part( 'template-parts/content', 'none' );
        } ?>
    </main>
<?php get_footer();
```

Listing 108: slider-front-page.php

Um auf der Seite eine `h1`-Überschrift anzuzeigen, wie es aus Gründen der Barrierefreiheit geboten ist, müssen die Bedingungen in der `header.php` erweitert werden, die in Listing 8 und Listing 9 auf Seite 26 und 27 behandelt worden sind. Dabei wurde der Seitentitel

abhängig von der angezeigten Seite entweder als `h1`-Überschrift oder als Absatz ausgezeichnet – für das Seiten-Template mit dem Slider soll er als `h1` ausgegeben werden. Listing 109 zeigt zwei Ausschnitte aus der `header.php`, die diese Änderung umsetzen. Zuerst muss vor der Ausgabe mit der Funktion `get_page_template_slug()` der Pfad des Seiten-Template ermittelt werden, indem der Funktion die ID der Seite übergeben wird.[222] Anschließend wird innerhalb der beiden `if`-Bedingungen neben der bereits vorhandenen Prüfung noch getestet, ob das Seiten-Template den Slug `page-templates/slider-front-page.php` besitzt. Ist das der Fall, wird entweder das Logo oder der Seiten-Titel von einem `h1`-Element umschlossen.

```
<div class="site-branding">
   <?php $page_template = get_page_template_slug( $post->ID );
   if ( get_header_image() ) {
      if ( ( is_front_page() && is_home() ) ||
         $page_template == 'page-templates/slider-front-page.php' ) { ?>
         <h1 class="logo">
[…]
   } else {
      if ( ( is_front_page() && is_home() ) ||
         $page_template == 'page-templates/slider-front-page.php' ) { ?>
         <h1 class="site-title"><?php bloginfo( 'name' ); ?></h1>
```

Listing 109: Ausgabe des Seitentitels als h1 auf der Startseite mit Slider

Das Owl-Carousel-Skript wird in der Datei `owl-carousel.js` innerhalb eines noch anzulegenden `js`-Ordners gespeichert, die CSS-Datei als `owl-carousel.css` in dem Ordner `css`. Um Parameter aus PHP zu JavaScript zu transferieren, damit die Entscheidung des Nutzers aus dem Customizer auch berücksichtigt werden kann, hat Samuel Wood einen hilfreichen Beitrag in seinem Blog verfasst.[223] Er nutzt die Funktion `wp_localize_script()`, die eigentlich für Strings im JavaScript-Code gedacht ist, die übersetzbar sein sollen.[224]

```
global $post;
$page_template = get_page_template_slug( $post->ID );
if ( $page_template == 'page-templates/slider-front-page.php' ) {
   wp_enqueue_style( 'owl-carousel', get_template_directory_uri()
      . '/css/owl-carousel.css' );
   wp_enqueue_script( 'owl-carousel', get_template_directory_uri()
      . '/js/owl-carousel.js', array( 'jquery' ), false, true );
```

Listing 110: Einbinden der CSS- und JS-Datei für Owl Carousel

[222] vgl. WordPress: Code Reference › Functions › get_page_template_slug
[223] vgl. Wood: Passing parameters from PHP to Javascripts in plugins
[224] vgl. WordPress: Code Reference › Functions › wp_localize_script

Das Skript und die CSS-Datei von Owl Carousel werden in der `functions.php` in die bereits bestehende `hannover_scripts_styles()`-Funktion eingebunden, wie Listing 110 zeigt. Zunächst muss allerdings geprüft werden, ob der aktuelle Seitenaufruf das Seiten-Template für den Slider auf der Startseite hat, damit das Skript und die CSS-Datei nur eingebunden werden, wenn es notwendig ist. Dafür wird mit `get_page_template_slug()` der Slug des Seiten-Templates ermittelt und anschließend geprüft, ob er dem Pfad des Seiten-Template entspricht.[225] Ist das der Fall, müssen die Dateien des Slider-Skripts eingebunden werden. Als ersten Parameter wird an `wp_enqueue_style()` ein Bezeichner übergeben, gefolgt von dem Pfad zur Datei.[226] Mit der Funktion `get_template_directory_uri()` wird der Pfad zum Theme-Ordner ausgegeben.[227] Ein wichtiger Unterschied zu `get_stylesheet_directory_uri()` ist, dass die erste Funktion, wenn sie aus einem Child-Theme aufgerufen wird, den Pfad zum Eltern-Theme ausgibt, statt den Pfad zum Child-Theme.[228] So funktioniert diese Referenz auch, wenn ein Child-Theme von Hannover genutzt wird.

An `wp_enqueue_script()` kann zusätzlich zum Bezeichner und dem Pfad noch ein Array aus Skripten angegeben werden, die vor diesem Skript geladen sein müssen – im Fall vom Owl Carousel ist das jQuery. Danach kann optional eine Version angegeben werden und als letztes ein boolescher Wert, der angibt, ob das Skript mit `wp_footer()` statt `wp_head()` ausgegeben werden soll – also im Footer des Themes.

```
        . '/js/owl-carousel.js', array( 'jquery' ), false, true );
    $slider_autoplay      = get_theme_mod( 'slider_autoplay' );
    $slider_autoplay_time = get_theme_mod( 'slider_autoplay_time',
        3000 );
    $params               = array(
        'autoplay'        => $slider_autoplay,
        'autoplayTimeout' => $slider_autoplay_time,
        'prev'            => __( 'Previous Slide', 'hannover' ),
        'next'            => __( 'Next Slide', 'hannover' ),

    );
    wp_localize_script( 'owl-carousel', 'OwlParams', $params );
}
```

Listing 111: Parameter aus dem Customizer an das Slider-Skript übergeben

Um die Lösung aus dem Beitrag von Samuel Wood zu nutzen, damit die Werte aus dem Customizer in der JavaScript-Datei verwendet werden können, wird Listing 111 benötigt.

[225] vgl. WordPress: Code Reference › Functions › get_page_template_slug
[226] vgl. WordPress: Code Reference › Functions › wp_enqueue_style
[227] vgl. WordPress: Code Reference › Functions › get_template_part_directory
[228] vgl. WordPress: Code Reference › Functions › get_stylesheet_directory_uri

Zunächst werden mit `get_theme_mod()` die Werte aus dem Customizer ermittelt – bei der Zeit wird über den zweiten Parameter der Standard-Wert angegeben.[229] Im Anschluss wird ein Array erstellt, in den die Werte eingetragen werden. Für `autoplay` und `autoplayTimeout` werden die entsprechenden Werte des Customizers eingesetzt, `prev` und `next` enthalten die übersetzbaren Beschriftungen für Screen-Reader-Nutzer, die den Vor- und Zurück-Buttons des Sliders zugewiesen werden.

Im nächsten Schritt muss die `wp_localize_script()`-Funktion aufgerufen werden. Als ersten Parameter erwartet sie den Bezeichner des Skripts, an den die Parameter übergeben werden sollen. Der zweite Parameter ist der Name des JavaScript-Objekts, über das die Argumente in der JavaScript-Datei zugänglich sind, der dritte Parameter ist der Name des Argument-Arrays.

```
jQuery(document).ready(function () {
    jQuery('.gallery').wrap("<div class='slider'></div>");
    jQuery('.gallery').owlCarousel({
        items: 1,
        animateOut: "fadeOut",
        animateIn: "fadeIn",
        loop: true,
        autoplay: OwlParams.autoplay,
        autoplayTimeout: OwlParams.autoplayTimeout,
        nav: true,
        navText: [
           '<span class="screen-reader-text">' + OwlParams.prev +
           '</span><span aria-hidden="true">‹</span>',
           '<span aria-hidden="true">›</span>' +
           '<span class="screen-reader-text">'
           + OwlParams.next + '</span>'],
   });
});
```

Listing 112: Initialisierung des Owl Carousel

Um den Slider zu implementieren, muss nun in die Datei `owl-carousel.js` gewechselt werden. Am Ende der Datei wird in Listing 112 zunächst mit der `ready()`-Funktion von jQuery sichergestellt, dass das Dokument fertig geladen ist.[230] Anschließend wird um jedes Element mit der Klasse `gallery` ein `div` mit der Klasse `slider` gelegt – Mittel der Wahl ist hierfür die `wrap()`-Funktion von jQuery, die als Parameter die HTML-Struktur erwartet, mit der das Element umschlossen werden soll.[231] Um den Slider zu initialisieren, wird

[229] vgl. WordPress: Code Reference › Functions › get_theme_mod
[230] vgl. jQuery: .ready
[231] vgl. jQuery: .wrap

für die Elemente mit der `gallery`-Klasse `owlCarousel()` ausgeführt, wie auf der Website des Skripts beschrieben.[232]

Um die Standard-Optionen anzupassen, werden diese überschrieben.[233] So wird mit `items` angegeben, dass nur ein Element angezeigt werden soll. Als Animation soll ein Ein- und Ausblenden genutzt werden und nach dem letzten Bild soll es wieder mit dem ersten weitergehen. Für den Schlüssel `autoplay` wird mit `OwlParams.autoplay` der Wert aus dem Customizer eingetragen, genauso für `autoplayTimeout`. Für `nav` wird `true` übergeben, damit Vor- und Zurück-Links angezeigt werden. Für `navText` wird ein Array aus den zwei übrigen Optionen gebildet, das über `wp_localize_script()` übergeben wurde, damit der Text für die Vor- und Zurück-Links übersetzbar ist. Dabei werden die Beschriftungen in HTML verpackt – im Frontend normal angezeigt werden soll jeweils ein Pfeil, der in einem `span` durch das Attribut `aria-hidden="true"` vor Screen-Readern versteckt wird. Für Screen-Reader-Nutzer hingegen ist das übersetzbare Label, das in einem `span` mit der Klasse `screen-reader-text` platziert wird, um es via CSS im Frontend zu verstecken.

5.24.2 Zufälliges Bild auf der Startseite

Das Seiten-Template für ein zufälliges Bild aus einer Galerie des Beitrags ist schnell erstellt, da alle notwendigen Funktionen bereits im Theme umgesetzt wurden. Für das Template wird die Datei `random-image-front-page.php` angelegt und in den Ordner `page-templates` gelegt.

```
<?php
/**
 * Template Name: Front page with random image
 */
get_header(); ?>
   <main role="main">
      <?php $images = hannover_get_gallery_images( $post->ID );
      shuffle( $images );
      $first_image = array_shift( $images ); ?>
      <figure>
         <?php if ( ! empty( $first_image ) ) {
            echo wp get attachment image( $first image->ID, 'large' );
         }
         if ( ! empty( $first_image->post_title ) ) { ?>
            <figcaption>
               <?php echo $first_image->post_title; ?>
```

[232] vgl. Wojciechowski: Owl Carousel › Installation
[233] vgl. Wojciechowski: Owl Carousel › Options

```
            </figcaption>
         <?php } ?>
      </figure>
   </main>
<?php get_footer();
```

Listing 113: random-image-front-page.php

Die komplette Datei ist in Listing 113 abgebildet. Innerhalb des `main`-Elements muss zunächst die `hannover_get_gallery_images()`-Funktion aufgerufen werden, um die Bilder aus dem oder den Gallery-Shortcodes zu ermitteln – detailliert erläutert wurde diese Funktion in Listing 65 auf Seite 71. Um ein zufälliges Bild aus dem Ergebnis anzuzeigen, muss das Array mit der `shuffle()`-Funktion gemischt und im Anschluss alles bis auf das erste Element mit `array_shift()` entfernt werden. [234, 235] Innerhalb eines `figure`-Elements wird dann durch Aufruf der `wp_get_attachment_image()`-Funktion und Übergabe der Bild-ID sowie der gewünschten Größe das `img`-Tag für das Bild ausgegeben – vorher wird noch überprüft, ob ein Bild vorhanden ist.[236]

Wenn im WordPress-Backend für das Bild ein Titel eingetragen ist, dann soll dieser in einem `figcaption`-Element angezeigt werden. Zugänglich ist der Wert des Bildtitels über den Array-Schlüssel `post_title`, der zunächst auf einen Wert geprüft wird. Ist ein Wert vorhanden, wird dieser in dem Element ausgegeben.

```
if ( get_header_image() ) {
   if ( ( is_front_page() && is_home() ) ||
      $page_template == 'page-templates/slider-front-page.php' ||
      $page_template == 'page-templates/random-image-front-page.php' ){?>
      <h1 class="logo">
[…]
} else {
   if ( ( is_front_page() && is_home() ) ||
      $page_template == 'page-templates/slider-front-page.php' ||
      $page_template == 'page-templates/random-image-front-page.php' ){?>
      <h1 class="site-title"><?php bloginfo( 'name' ); ?></h1>
```

Listing 114: Ausgabe des Seitentitels als h1 auf der Startseite mit zufälligem Bild

Auch für dieses Seiten-Template muss der Seitentitel als `h1`-Überschrift ausgegeben werden – die entsprechende Anpassung in der `header.php` , von der Vorgehensweise bereits aus Listing 109 auf Seite 100 bekannt, ist in Listing 114 abgebildet.

[234] vgl. PHP Group: shuffle
[235] vgl. PHP Group: array_shift
[236] vgl. WordPress: Code Reference › Functions › wp_get_attachment_image

5.25 Galerien als Slider oder einzelne Bilder

Der Nutzer soll entscheiden können, ob sämtliche Galerien auf der Seite als Slider statt als einzelne Bilder dargestellt werden sollen – wie im Seiten-Template mit Slider für die Startseite. Auch diese Option findet ihren Platz wieder im Customizer und wird mit einer Checkbox implementiert.

```
        'settings' => 'slider_autoplay_time',
    )
);

$wp_customize->add_setting(
    'galleries_as_slider', array(
        'sanitize_callback' => 'hannover_sanitize_checkbox',
    )
);

$wp_customize->add_control(
    'galleries_as_slider', array(
        'label'    => __( 'Display all galleries as sliders',
            'hannover' ),
        'type'     => 'checkbox',
        'section'  => 'slider_settings',
        'settings' => 'galleries_as_slider'
    )
);
```

Listing 115: Option, alle Galerien als Slider darzustellen

Listing 115 zeigt den im Prinzip bereits bekannten Code, um die Checkbox in den Customizer einzufügen. Als Section wird dieselbe genutzt, in der auch die Optionen für Autoplay und Zeitspanne dargestellt werden.

```
global $post;
$galleries_as_slider = get_theme_mod( 'galleries_as_slider' );
$page_template       = get_page_template_slug( $post->ID );
if ( $page_template == 'page-templates/slider-front-page.php' ||
     $galleries_as_slider == 'checked'
) {
    wp_enqueue_style( 'owl-carousel', get_template_directory_uri()
        . '/css/owl-carousel.css' );
```

Listing 116: Owl-Carousel-Ressourcen für alle Galerien nutzen, wenn im Customizer gewählt

Um diese Einstellung im Frontend sichtbar zu machen, muss lediglich die Bedingung in der `hannover_scripts_styles()` erweitert werden, die dafür sorgt, dass das Stylesheet und das Skript von Owl Carousel eingebunden werden. Wie diese Erweiterung aussieht, ist in Listing 116 abgebildet. Mit `get_theme_mod()` wird die entsprechende Einstellung aus dem Customizer ermittelt und anschließend in der `if`-Bedingung geprüft, ob

diese Option ausgewählt ist und/oder das Seiten-Template mit dem Slider auf der Startseite angezeigt wird.

5.26 Vorschau eines Galerie- oder Bild-Beitrags

In der Vorschau für einen Beitrag vom Typ *Galerie* oder *Bild* soll über dem Titel das erste Bild und statt des `the_content()`-Aufrufs ein eventuell vorhandener Auszug des Beitrags angezeigt werden. Damit die Ansicht für die beiden Post-Formate richtig ausgegeben wird, müssen die Dateien `content-gallery.php` sowie `content-image.php` im Ordner `template-parts` erstellt werden, die den identischen Inhalt haben.

```
<article <?php post_class(); ?> id="post-<?php the_ID(); ?>">
   <header class="entry-header">
      <?php hannover_image_from_gallery_or_image_post( 'large',
         $post );
      hannover_the_title( 'h2', true ); ?>
   </header>
   <div class="entry-content">
      <?php if ( has_excerpt() ) {
         the_excerpt();
      } ?>
   </div>
   <footer>
      <p><a href="<?php the_permalink(); ?>"
            class="entry-date"><?php hannover_the_date(); ?></a>
         <?php hannover_entry_meta() ?></p>
   </footer>
</article>
```

Listing 117: Vorschau von Galerie- und Bild-Beiträgen

Der komplette Code für beide Dateien ist in Listing 117 abgebildet. Innerhalb des header-Elements wird mit `hannover_image_from_gallery_or_image_post()` das erste Bild der Galerie oder des Beitrags angezeigt – genau erläutert wird die Funktion ab Listing 61 auf Seite 67. Anschließend gibt `hannover_the_title()`, die in Listing 16 auf Seite 36 erklärt wird, den verlinkten Titel auf den Beitrag als `h2`-Überschrift aus. Innerhalb des Inhalt-`div`s wird mit `has_excerpt()` geprüft, ob der Nutzer einen Auszug für den Beitrag eingetragen hat und in dem Fall dieser mit `the_excerpt()` ausgegeben.[237, 238]

[237] vgl. WordPress: Code Reference › Functions › the_excerpt
[238] vgl. WordPress: Code Reference › Functions › has_excerpt

5.27 Read-More-Sprung verhindern

Wenn ein Read-More-Link auf einer Übersichtsseite angezeigt wird, weil der Nutzer ihn in seinem Beitrag eingefügt hat, dann wird standardmäßig eine Sprungmarke eingefügt, sodass auf der Einzelansicht des Beitrags direkt zu dem Teil gesprungen wird, der als nächstes gelesen werden soll. Dieses Verhalten ist etwas verwirrend, da es nicht erwartet wird – daher wird diese Sprungmarke mit dem Code aus Listing 118 von den Read-More-Links entfernt.

```
function hannover_remove_more_link_scroll( $link ) {
   $link = preg_replace( '/#more-[0-9]+/', '', $link );

   return $link;
}

add_filter( 'the_content_more_link',
   'hannover_remove_more_link_scroll' );
```

Listing 118: Entfernen der Sprungmarke beim Read-More-Link

Da die Funktion `hannover_remove_more_link_scroll()` an den Filter `the_content_more_link` übergeben wird, ist im Parameter `$link` der Read-More-Link vorhanden.[239] In dieser URL wird mit der `preg_replace()`-Funktion die Sprungmarke entfernt und die bereinigte URL zurückgegeben.[240]

5.28 Menü

Damit das Menü mit möglichst vielen verschiedenen Methoden genutzt werden kann, muss JavaScript eingesetzt werden – zu diesen Methoden gehört beispielsweise das Öffnen eines DropDown-Menüs mit einem Touchscreen oder die Nutzung des Menüs über eine Tastatur. Auch das Einblenden des Menüs auf kleinen Viewports muss vorbereitet werden.

Da das Rad nicht neu erfunden werden muss, wird die Lösung des diesjährigen WordPress-Standard-Themes *Twenty Sixteen* genutzt – der notwendige JavaScript-Code ist im GitHub-Repository des Themes zu finden.[241] Innerhalb des Codes müssen lediglich einige Zeilen, die nichts mit dem Menü zu tun haben, gelöscht beziehungsweise angepasst werden, damit jQuery die Elemente mit den korrekten Klassen und IDs sucht.

[239] vgl. WordPress: Code Reference › Hooks › the_content_more_link
[240] vgl. PHP Group: preg_replace
[241] vgl. Carlson u.a. (2015): twentysixteen/js/functions.js

Um zwei Screen-Reader-Texte übersetzbar zu integrieren, kommt die Funktion `wp_localize_script()` zum Einsatz, wie bereits in Listing 111 erläutert. Wie die Einbindung in `hannover_scripts_styles()` aussieht, zeigt Listing 119.

```
wp_enqueue_script( 'hannover-menu', get_template_directory_uri()
   . '/js/menu.js', array( 'jquery' ), false, true );

wp_localize_script( 'hannover-menu', 'screenReaderText', array(
   'expand'   => __( 'expand child menu', 'hannover' ),
   'collapse' => __( 'collapse child menu', 'hannover' ),
) );
```

Listing 119: Einbinden des Menü-Skripts

5.29 Footer-Bereich erweitern

Um Links anzuzeigen, die vielleicht nicht in die Hauptnavigation gehören, wie beispielsweise das Impressum, bekommt das Theme ein Menü im Footer. Darüber hinaus soll der Nutzer ein Menü mit Social-Media-Links anlegen können, die dann automatisch als SVG-Grafiken angezeigt werden. Ganz unten in den Footer kommt ein Hinweis auf den Ersteller des Themes.

5.29.1 Social-Media-Menü

Häufig werden zur Anzeige von Icons auf Websites sogenannte Icon Fonts eingesetzt. Das sind Fonts, die statt Buchstaben Icons enthalten und oft durch Einsetzen einer bestimmten Klasse im HTML-Code via CSS eingefügt werden. Das ist nicht besonders semantisch und barrierefrei, im Gegensatz zu SVG, wie Sara Soueidan erläutert.

> „SVGs are semantic. An SVG icon is a graphic — an image — so what better way to mark up an image than to use a (scalable vactor) graphic tag? An SVG element (`<svg>`) represents an icon simply and semantically, while icon fonts usually require non-semantic markup like pseudo-elements and empty `<span>`s to be displayed. For people concerned about semantics and accessibility this introduces a serious issue: these elements don't accommodate screen readers well, not to mention that the markup generally just doesn't make much sense — if any — for an image.“[242]

Um SVG-Icons als Vordergrundbilder zu nutzen, wie es in dem Social-Media-Menü der Fall sein soll, gibt es laut Soueidan die folgende gute Möglichkeit:

```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg"
```

[242] Soueidan (2015), S. 161.

```
   style="display: none;">
   <symbol id="icon-twitter" viewBox="0 0 16 16">
      <title>Twitter</title>
      <!-- Twitter-Icon-Markup -->
   </symbol>

   <symbol id="icon-facebook" viewBox="0 0 16 16">
      <title>Facebook</title>
      <!-- Twitter-Icon-Markup -->
   </symbol>
   <!-- weitere Icons -->
</svg>
```

Listing 120: SVG-Markup für die Social-Media-Icons

Der SVG-Code aus Listing 120 wird in eine eigene SVG-Datei `social-media-icons.svg` in dem `svg`-Ordner eingefügt. Innerhalb des `symbol`-Elements kann für die bessere Barrierefreiheit ein `title`-Element angegeben werden, gefolgt von dem SVG-Markup für das Icon. Es ließe sich auch noch für jedes Symbol eine Beschreibung einfügen, allerdings gibt es scheinbar keine Möglichkeit, diese übersetzbar zu integrieren. Die Icons stammen zum großen Teil aus dem Genericons-Set von Automattic, abgesehen von dem Xing-Icon, das von *Font Awesome* ist. [243, 244] Die komplette SVG-Datei mit den Icons ist im GitHub-Repository des Hannover-Themes zugänglich.[245]

Jedes Icon kann auf der Website mit der Methode aus Listing 121 angezeigt werden. Innerhalb des `svg`-Elements wird in dem `use`-Tag der Pfad zu der SVG-Datei angegeben, inklusive einem Hash-Wert, der mit der `id` des entsprechenden `symbol`-Elements übereinstimmt.[246]

```
<svg class="icon-twitter">
   <use xlink:href=
      "/pfad-zum-theme/svg/social-media-icons.svg#icon-twitter"/>
</svg>
```

Listing 121: SVG auf der Website anzeigen

Diese Ausgabe aus Listing 121 muss statt des normalen Textlinks von der Menü-Funktion ausgegeben werden.

```
'primary' => __( 'Primary Menu', 'hannover' ),
/* translators: Name of menu position for social media icons */
'social'  => __( 'Social Menu', 'hannover' ),
```

Listing 122: Registrierung des Social-Media-Menüs in der functions.php

[243] vgl. automattic: Genericons
[244] vgl. Gandy: Font Awesome
[245] vgl. Brinkmann: Tag „v1.0.8“ – hannover/svg/social-media-icons.svg
[246] vgl. Soueidan (2015), S. 175-176.

Damit die Position für das neue Social-Media-Menü im Backend auswählbar ist, muss die `register_nav_menus()`-Funktion in der `functions.php` um einen Array-Eintrag erweitert werden.[247] Um das Menü anzuzeigen, muss in die bereits erstellte `footer.php` gewechselt werden.

```
<footer id="footer">
   <?php if ( has_nav_menu( 'social' ) ) {
      wp_nav_menu(
         array(
            'theme_location' => 'social',
            'menu_class'     => 'social-menu',
            'container'      => '',
            'walker'         => new Hannover_Social_Menu_Walker(),
            'depth'          => 1
         )
      );
   }
```

Listing 123: Ausgabe des Social-Media-Menüs in der footer.php

Zunächst wird in Listing 123 mit `has_nav_menu()` geprüft, ob die Menüposition `social` mit einem Menü belegt ist.[248] Falls das so ist, gibt `wp_nav_menu()` das entsprechende Menü von dieser Position aus. Wichtig ist im Vergleich zu dem Menü in der `header.php`, dass für `walker` ein Wert angegeben wird, damit die Ausgabe individuell angepasst werden kann. Die Klasse, die dafür verantwortlich ist, kommt in die Datei `class-hannover-social-menu-walker.php` in den `inc`-Ordner. Um sie in die `functions.php` einzubinden, ist Listing 124 zuständig.

```
require get_template_directory() . '/inc/customizer.php';

require get_template_directory()
   . '/inc/class-hannover-social-menu-walker.php';
```

Listing 124: Einbinden der class-hannover-social-menu-walker.php

Die Klasse `Hannover_Social_Menu_Walker` erweitert `Walker_Nav_Menu`, die für die Standard-Ausgabe der Menüs zuständig ist.[249] Überschrieben werden muss lediglich die `start_el()`-Methode, die sich um die Ausgabe der Menüelemente kümmert.[250] Dabei wird der größte Teil der Methode einfach übernommen und lediglich ein kleiner Teil verändert. Dieser angepasste und selbst programmierte Teil wird hier erläutert, die komplette

[247] vgl. WordPress: Code Reference › Functions › register-nav_menus
[248] vgl. WordPress: Code Reference › Functions › has_nav_menu
[249] vgl. WordPress: Code Reference › Classes › Walker_Nav_Menu
[250] vgl. WordPress: Code Reference › Classes › Walker_Nav_Menu › Walker_Nav_Menu::start_el

Klasse ist im Repository auf GitHub zugänglich – der Teil, der aus Gründen der Orientierung aus der übernommenen Klasse abgebildet wird, ist ausgegraut.[251]

```
<?php

class Hannover_Social_Menu_Walker extends Walker_Nav_Menu {
   […]
   public function start_el( &$output, $item, $depth = 0,
      $args = array(), $id = 0 ) {
      […]
      $title = apply_filters( 'nav_menu_item_title', $title, $item,
$args, $depth );

      /**
       * Build array to get the class name for the social network in
            the SVG
       */

      $social_media_channels = array(
         'plus.google.com' => 'icon-google-plus',
         'wordpress.org'   => 'icon-wordpress',
         'wordpress.com'   => 'icon-wordpress',
         'facebook.com'    => 'icon-facebook',
         'twitter.com'     => 'icon-twitter',
         'dribbble.com'    => 'icon-dribbble',
         'pinterest.com'   => 'icon-pinterest',
         'github.com'      => 'icon-github',
         'tumblr.com'      => 'icon-tumblr',
         'youtube.com'     => 'icon-youtube',
         'flickr.com'      => 'icon-flickr',
         'vimeo.com'       => 'icon-vimeo',
         'instagram.com'   => 'icon-instagram',
         'linkedin.com'    => 'icon-linkedin',
         'xing.de'         => 'icon-xing',
         'xing.com'        => 'icon-xing',
         '/feed'           => 'icon-feed',
      );
```

Listing 125: Erster Ausschnitt aus der Hannover_Social_Menu_Walker()-Klasse

Es muss geprüft werden, zu welchem sozialen Netzwerk die URL des gerade durchlaufenen Menüpunkts gehört und dann die entsprechende SVG-ID genutzt werden. Dafür wird in Listing 125 kurz vor Bearbeitung der Menüpunkte ein Array erstellt, das als Schlüssel immer die URL des sozialen Netzwerks und als entsprechenden Wert die ID aus der SVG-Datei enthält.

```
$svg_id = "";

foreach ( $social_media_channels as $key => $value ) {
   $pattern = "|$key|";
   preg_match( $pattern, $atts['href'], $matches );
```

[251] vgl. Brinkmann: Tag „v1.0.8“ – hannover/inc/class-hannover-social-walker.php

```
    if ( ! empty( $matches[0] ) ) {
        $match  = $matches[0];
        $svg_id = $social_media_channels[ $match ];
        break;
    }
}
```

Listing 126: Prüfen der aktuellen Menü-URL und speichern der zugehörenden ID

In Listing 126 wird ein leerer String der Variable `$svg_id` zugewiesen. Danach wird für das Array aus Social-Media-Kanälen eine Schleife durchlaufen, in der der jeweilige Schlüssel des Arrays in der `$key`-Variable und der Wert in `$value` zugänglich ist. In der URL des aktuellen Menüpunkts, die in `$atts['href']` gespeichert ist, muss nach dem Vorkommen von `$key` gesucht werden. Dafür wird die Funktion `preg_match()` eingesetzt, der als erster Parameter der Ausdruck übergeben wird, nach dem gesucht werden soll, als zweiter der String, der durchsucht wird und als dritter eine Variable, in der der Treffer gespeichert wird.

Wenn bei dem Durchlauf des Social-Media-Arrays eine Übereinstimmung gefunden wird, ist `$matches[0]` nicht mehr leer, sondern enthält den Schlüssel aus dem Array, also die URL des sozialen Netzwerks. In diesem Fall wird der Wert der Variable `$match` zugewiesen und im nächsten Schritt die ID aus dem Array geholt, indem als Schlüssel der Wert aus `$match` übergeben wird. Anschließend wird der Schleifendurchlauf abgebrochen, damit die Schleife so selten wie möglich für jeden Menüpunkt durchlaufen werden muss.

```
if ( $svg_id != "" ) {
    $icon_url    = get_template_directory_uri()
        . '/svg/social-media-icons.svg#' . $svg_id;
    $item_output = $args->before;
    $item_output .= '<a' . $attributes . '>';
    $item_output .= '<svg class="' . $svg_id . '"><use xlink:href="'
        . $icon_url . '"></use></svg>';
    $item_output .= '</a>';
    $item_output .= $args->after;
} else {
    $item_output = $args->before;
    $item_output .= '<a' . $attributes . '>';
    $item_output .= $args->link_before . $title . $args->link_after;
    $item_output .= '</a>';
    $item_output .= $args->after;
}
```

Listing 127: Zusammensetzen des Menüpunkts in Hannover_Social_Menu_Walker

Wenn `$svg_id` nicht leer ist, kann die passende SVG ausgegeben werden. In `$icon_url` wird die URL gespeichert, die mit `get_template_directory_uri()`, dem statischen Teil `/svg/social-media-icons.svg#` und der ID aus `$svg_id` zusammengesetzt wird. Die `$item_output`-Anweisungen, die ausgegraut sind, wurden aus der Methode der

Oberklasse übernommen. Die Zeile, die eigentlich den Titel ausgibt und das in dem `else`-Zeig auch tut, ist im `if`-Zweig durch die Ausgabe des `svg`-Elements ersetzt worden.

Browser-Unterstützung für externe SVGs

Da Internet Explorer 9 bis 11 keine externen SVGs anzeigen, die mit dem `use`-Element eingebunden sind, wird das Skript `svg4everybody` eingesetzt.[252, 253] Die Datei `svg4everybody.js` findet Platz in dem `js`-Ordner und wird hier nicht extra abgebildet, da die einzige Anpassung der Datei aus dem GitHub-Repository das Einfügen des `svg4everybody()`-Aufrufs ist.[254]

Damit das Skript nur eingebunden wird, wenn die Menüposition `social` verwendet wird, wird erneut die `has_nav_menu()`-Funktion genutzt. In Listing 128 ist der Code abgebildet, der dafür zuständig ist.

```
wp_enqueue_script( 'hannover-menu', get_template_directory_uri()
   . '/js/menu.js', array( 'jquery' ), false, true );

if ( has_nav_menu( 'social' ) ) {
   wp_enqueue_script( 'hannover-svg4everybody',
      get_template_directory_uri() . '/js/svg4everybody.js',
      array( 'jquery' ), false, true );
}
```

Listing 128: Einbinden von svg4everybody.js in der functions.php

5.29.2 Footer-Menü und Theme-Autor-Hinweis

```
'primary' => __( 'Primary Menu', 'hannover' ),
/* translators: Name of menu position in the footer */
'footer'  => __( 'Footer Menu', 'hannover' ),
/* translators: Name of menu position for social media icons */
```

Listing 129: Registrieren des Footer-Menüs in der functions.php

Um die Position des Footer-Menüs im Backend einzublenden, muss sie wie beim Social-Media-Menü zunächst registriert werden. Der Teil ist in Listing 129 abgebildet und vom Prinzip her aus Listing 12 von Seite 30 bekannt.

```
<?php if ( has_nav_menu( 'footer' ) ) { ?>
   <nav>
      <h2 class="screen-reader-text">
         <?php /* translators: hidden screen reader headline for the
```

[252] vgl. Can I use: SVG (basic support)
[253] vgl. Neal: svg4everybody
[254] vgl. Brinkmann: Tag „v1.0.8“ – hannover/js/svg4everybody.js

```
            footer navigation */
            _e( 'Footer navigation', 'hannover' ); ?>
        </h2>
        <?php wp_nav_menu(
            array(
                'theme_location' => 'footer',
                'menu_class'     => 'footer-menu',
                'container'      => ''
            )
        ); ?>
    </nav>
<?php } ?>
```

Listing 130: Einfügen der Footer-Navigation in der footer.php

Um das Menü von der Menüposition `footer` anzuzeigen, kommt erneut die Funktion `has_nav_menu()` zum Einsatz, damit sichergestellt wird, dass es etwas zum Anzeigen gibt. Innerhalb des `nav`-Elements wird für Screen-Reader-Nutzer eine Überschrift eingefügt und anschließend die Navigation mit `wp_nav_menu()` ausgegeben.

```
    <p class="theme-author">
        <?php _e( 'Theme: Hannover by <a rel="nofollow"
            href="https://florianbrinkmann.de">Florian Brinkmann</a>',
            'hannover' ) ?>
    </p>
</footer>
```

Listing 131: Hinweis auf den Theme-Autoren in der footer.php

In Listing 131 ist der abschließende Hinweis auf den Autoren des Themes abgebildet, der ganz ans Ende des sichtbaren Footers kommt.

5.30 Design

Das Design in Form des CSS-Codes wird nicht Teil dieser Bacherlorarbeit sein, da eine detaillierte Besprechung aller CSS-Regeln erstens zu umfangreich wäre und zweitens sich diese Arbeit auf die WordPress-Funktionen konzentrieren soll. Der komplette CSS-Code des Themes wird in die `hannover.css` innerhalb des `css`-Ordners geschrieben. Theoretisch böte sich dafür auch die `style.css` an – das Problem mit dem Code in dieser Datei ist aber, dass er sich nicht komplett minimieren ließe, da der Kommentar mit den Meta-Daten für das Theme nicht minimiert werden darf.

```
wp_enqueue_style( 'hannover-style', get_template_directory_uri()
    . '/css/hannover.css', array(), null );
```

Listing 132: Einbinden der CSS-Datei

Wie alle anderen Skripte und Stylesheets wird auch die CSS-Datei des Themes innerhalb der `hannover_scripts_styles()`-Funktion eingebunden – der dafür notwendige Code ist in Listing 132 abgebildet. Für das Einbinden wird die Funktion `wp_enqueue_style()` verwendet, deren Parameter bereits erläutert wurden.[255]

5.30.1 Notwendige CSS-Selektoren

Es gibt einige CSS-Selektoren, auf die das Theme-Check-Plugin prüft.[256] So müssen in jedem Fall die folgenden Selektoren in der CSS-Datei angesprochen werden: `.wp-caption` und `.wp-caption-text`, die bei Bildbeschriftungen zum Einsatz kommen. `.sticky`, das dem oder den Beiträgen zugewiesen wird, die in der Blog-Übersicht oben gehalten werden sollen und `.gallery-caption`, das dem Element zugewiesen ist, welches die Bildbeschriftungen in einer Galerie enthält.

`.bypostauthor` spricht Kommentare an, die vom selben Benutzer geschrieben wurden, der den Beitrag verfasst hat. `.alignright`, `.alignleft` und `.aligncenter` sorgen jeweils für die entsprechende Ausrichtung von Elementen – wenn im WordPress-Editor beispielsweise ein Bild rechts ausgerichtet ist, wird die Klasse `.alignright` zugewiesen. Der Selektor `.screen-reader-text` spricht Elemente an, die im Frontend versteckt sind und für Screen-Reader-Nutzer gedacht sind.

5.30.2 Lightbox einbinden

Damit Bilder durch einen Klick in einer Lightbox geöffnet werden können und dafür nicht extra ein Plugin installiert werden muss, wird die Lösung von Osvaldas Valutis eingesetzt.[257] Standardmäßig bindet sie keine Navigationselemente ein, um nicht vom angezeigten Inhalt abzulenken. Auf der Demo-Seite des Projekts sind die Navigationselemente jedoch umgesetzt, weshalb der entsprechende Code von dort übernommen werden konnte – es waren lediglich leichte Anpassungen notwendig.[258]

```
wp_enqueue_script( 'hannover-lightbox', get_template_directory_uri()
   . '/js/lightbox.js', array( 'jquery' ), false, true );
```

Listing 133: Einbinden des Lightbox-Skripts

[255] vgl. WordPress: Code Reference › Functions › wp_enqueue_style
[256] Wood/Obenland (2015): theme-check/checks/style_needed.php
[257] vgl. Valutis: Image Lightbox, Responsive and Touch-friendly
[258] vgl. Valutis: Image Lightbox › Demo

Der komplette Code für die Lightbox kommt in die `lightbox.js` im `js`-Verzeichnis und wird eingebunden, wie in Listing 133 abgebildet. Die CSS-Angaben werden mit in die `hannover.css` geschrieben.

5.30.3 Schrift

Als Schrift kommt in dem Theme *Noto Sans* von Google zum Einsatz. Den Font gibt es im Schrift-Dienst von Google, den *Google Web Fonts*, und kann deshalb direkt von dort in das Theme eingebunden werden. Um die Schrift zu nutzen, muss lediglich die URL, die nach Wahl der Schriften und Schriftschnitte bereitgestellt wird, wie ein Stylesheet eingebunden werden.

```
wp_enqueue_style( 'hannover-fonts', '//fonts.googleapis.com/css
   ?family=Noto+Sans:400,700,400italic,700italic', array(), null );
```

Listing 134: Einbinden der Schriftart

Listing 134 zeigt den Code, der dafür notwendig ist. Wie alle anderen Funktionen zum Einbinden von Skripten und Stylesheets kommt auch dieser Teil in die `hannover_scripts_styles()`-Funktion in der `functions.php`. An die Funktion `wp_enqueue_style()` wird einfach als zweiter Parameter die URL aus den Google Web Fonts übergeben.

5.30.4 Inhaltsbreite definieren und große Bildgröße anpassen

Nach der groben Umsetzung des Designs stellt sich heraus, dass der Inhaltsbereich maximal 845 Pixel breit ist. Um anzugeben, dass der eingefügte Inhalt höchstens diese Breite haben soll, gibt es in WordPress die Variable `$content_width`.[259] In dieser Breite werden beispielsweise oEmbed-Inhalte wie etwa Tweets eingebunden und auch Plugins können auf diesen Wert zugreifen, um ihre Inhalte bestmöglich an das Theme anzupassen.

Für die Festlegung dieses Wertes reicht es aus, in der `functions.php` den Code aus Listing 135 einzufügen.

```
if ( ! isset( $content_width ) ) {
   $content_width = 845;
}
```

Listing 135: Definieren der Inhaltsbreite in der functions.php

[259] vgl. WordPress: Codex › Content Width

Um den Standard-Wert der großen Bildgröße dahingehend anzupassen, dass er 845 Pixeln entspricht, gibt es die `update_option()`-Funktion.[260] Als erster Parameter wird der Name der Option übergeben, die aktualisiert werden soll, gefolgt von einem oder mehreren Parametern, die den neuen Wert angeben. Mit dem ersten Parameter `large_size_w` lässt sich die Breite der großen Bildgröße anpassen. Listing 136 zeigt die Anpassung des Wertes auf eine Breite von 845 Pixeln.

```
update_option( 'large_size_w', 845 );
```

Listing 136: Anapassen der Breite von der großen Bildgröße in der functions.php

5.30.5 Screenshot

Für die Präsentation im Backend, muss eine `screenshot.png` im Root-Vereichnis des Themes platziert werden. Laut Theme-Review-Handbook soll dieser Screenshot nicht größer als 1.200 x 900 Pixel sein.[261] Auch bei den Bildern auf dem Screenshot muss wie bei allen anderen Ressourcen auf die Lizenz geachtet werden. Bilder mit einer geeigneten Lizenz finden sich beispielsweise auf unsplash.com. Der finale Screenshot des Themes ist in Abbildung 8 zu sehen.

[260] vgl. WordPress: Code Reference › Functions › update_option
[261] vgl. WordPress: Theme Review Handbook › Required › Screenshot

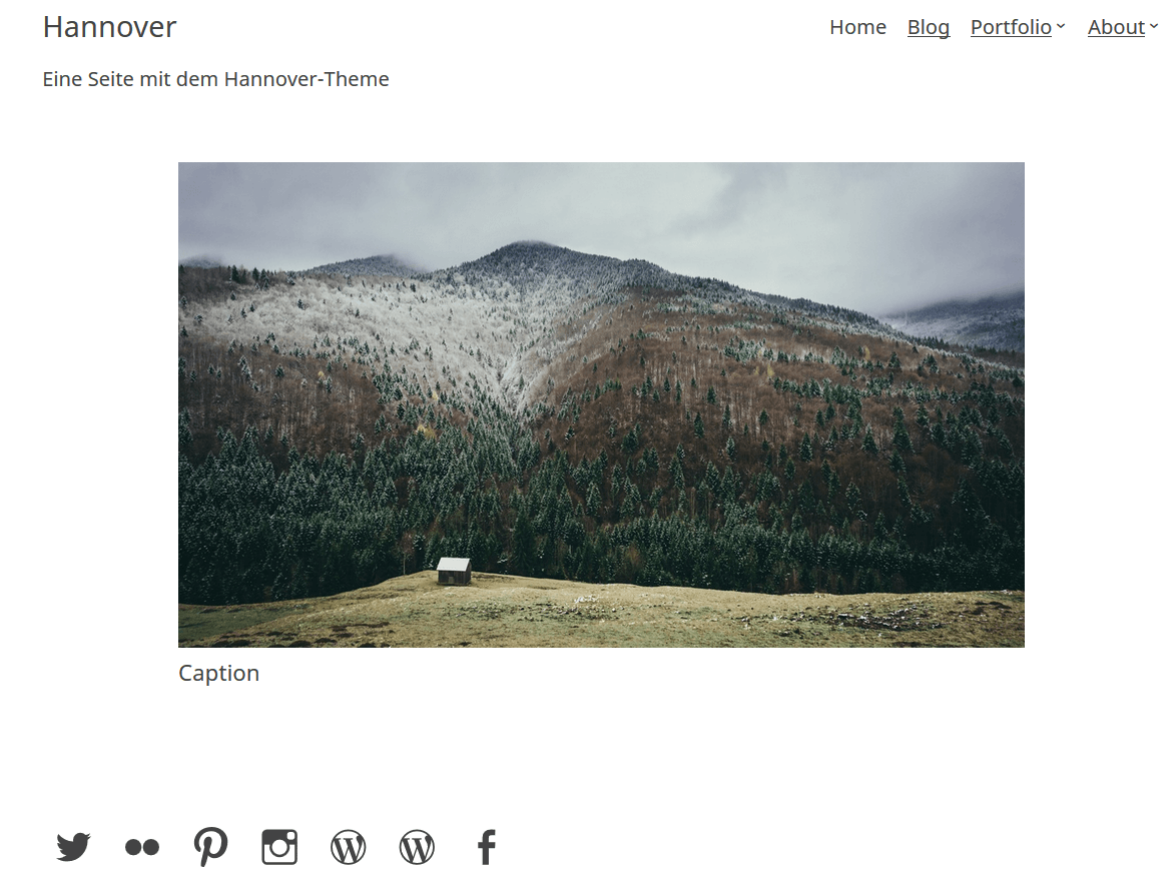


Abbildung 8: Screenshot des Themes, der im Backend und im Theme-Verzeichnis angezeigt wird

5.31 Readme

In die Readme-Datei eines Themes sollte idealerweise ein Changelog und in jedem Fall Hinweise zu den Lizenzen verwendeter Ressourcen eingefügt werden. Im Fall des Hannover-Themes sind folgende externe Ressourcen verwendet worden: Der Font *Noto Sans*, das Skript *Owl Carousel* für den Slider, das Lightbox-Script von Osvaldas Valutis, das JavaScript von Twenty Sixteen für das Menü, *Genericons* für die Social-Media-Icons und *Font Awesome* für das Xing-Icon, *svg4everybody* für den Browser-Fallback und das Bild im Screenshot.

Listing 137 zeigt die komplette Readme des Themes inklusive Changelog und Copyright-Hinweis ganz am Ende.

```
== Changelog ==

Version 1.0 - 26.12.2015
---
- initial release
```

```
== Copyright ==

Noto Sans
Licence: Apache License, version 2.0
Source: https://www.google.com/fonts/specimen/Noto+Sans

Owl Carousel
Licence: MIT License
Source: http://www.owlcarousel.owlgraphic.com/

Lightbox Script
Licence: MIT License
Source: http://osvaldas.info/image-lightbox-responsive-touch-friendly

Menu JS
Licence: GPLv2 or later
Source:
https://github.com/WordPress/twentysixteen/blob/master/js/functions.js

Genericons icon font, Copyright 2013-2015 Automattic.com
License: GNU GPL, Version 2 (or later)
Source: http://www.genericons.com

Xing Icon from Font Awesome 4.5
Licence: SIL OFL 1.1
Source: http://fontastic.me/

svg4everybody
Licence: CC0 1.0 Universal License
Source: https://github.com/jonathantneal/svg4everybody

Image from Screenshot
Licence: CC0 1.0 Universal
Source: https://unsplash.com/photos/a0YaUXLjKaY

---
Hannover WordPress Theme, Copyright 2015 Florian Brinkmann
Hannover is distributed under the terms of the GNU GPL
```

Listing 137: Readme-Datei

5.32 Übersetzung

Durch die konsequente Nutzung der Gettext-Funktionen ist das Theme auf die Übersetzung vorbereitet. Seit einiger Zeit können alle Themes des Theme-Verzeichnisses auf dem *GlotPress* von WordPress, das unter translate.wordpress.org erreichbar ist, übersetzt werden. Theme-Autoren können jedoch eigene Übersetzungen mitliefern, die die jeweilige Sprache dann überschreiben. Das Hannover-Theme soll von diesen Vorteilen aber profitieren, weshalb keine Übersetzung beigefügt wird.

Um Übersetzungen zu laden, muss die `load_theme_textdomain()`-Funktion genutzt werden, der als Parameter die Text-Domain des Themes übergeben wird.[262] Um die Übersetzungen nur zu laden, wenn es notwendig ist, wird der Aufruf von einigen Bedingungen abhängig gemacht.

```
function hannover_load_translation() {
   if ( ( ! defined( 'DOING_AJAX' ) && ! 'DOING_AJAX' ) ||
      ! hannover_is_login_page() || ! hannover_is_wp_comments_post() ) {
      load_theme_textdomain( 'hannover' );
   }
}

add_action( 'after_setup_theme', 'hannover_load_translation' );

function hannover_is_login_page() {
   return in_array( $GLOBALS['pagenow'], array( 'wp-login.php',
      'wp-register.php' ) );
}

function hannover_is_wp_comments_post() {
   return in_array( $GLOBALS['pagenow'], array( 'wp-comments-post.php' )
);
}
```

Listing 138: Laden der Übersetung in der functions.php

Listing 138 zeigt den kompletten Code, der das Laden der Übersetzungen bewerkstelligt. In der Funktion `hannover_load_translation()` wird mit `load_theme_textdomain()` die Übersetzung geladen. Vorher wird geprüft, ob ein AJAX-Request ausgeführt wurde – in diesem Fall muss die Übersetzung des Themes nicht neu geladen werden. Dafür gibt es in WordPress die Konstante `DOING_AJAX`. Ebenfalls nicht geladen werden soll die Übersetzung auf der Backend-Login-Seite und der Seite, die aufgerufen wird, wenn ein Kommentar geschrieben wurde.

Dafür wird jeweils eine Hilfsfunktion erstellt, die einen booleschen Wert zurückgibt. Mit der Funktion `hannover_is_login_page()` wird überprüft, ob die gerade aufgerufene Seite die `wp-login.php` oder `wp-register.php` ist. Dabei kommt die `in_array()`-Funktion zum Einsatz, die überprüft, ob der Wert des ersten Parameters in dem Array vorhanden ist, das als zweiter Parameter übergeben wird – die Globale Variable `pagenow` liefert die aktuelle Seite.[263]

Ähnlich wird in der Funktion `hannover_is_wp_comments()` verfahren. Hier wird geprüft, ob die aktuelle Seite der `wp-comments-post.php` entspricht.

[262] vgl. WordPress: Code Reference › Functions › load_theme_textdomain
[263] vgl. PHP Group: in_array

5.33 Dokumentation

Um für andere Nutzer das Theme gut zu dokumentieren, wird über jeder Funktion ein erklärender Kommentar eingefügt, der gegebenenfalls auch die Parameter der Funktion aufführt sowie den Typ des Rückgabewertes. Im Fall der Sanitization-Funktion für die Checkboxen im Customizer sieht das beispielhaft so aus, wie in Listing 139 zu sehen.

```
/**
 * Sanitizes checkbox input
 *
 * @param $checked
 *
 * @return bool
 */
function hannover_sanitize_checkbox( $checked ) {
   return ( ( isset( $checked ) && true == $checked ) ? true : false );
}
```

Listing 139: Beispiel eines dokumentierenden Kommentars

Zunächst kommt in den PHP-Kommentar eine kurze Beschreibung der Funktion. Hinter `@param` wird anschließend aufgelistet, welche Parameter die Funktion erwartet und mit `@return` wird angegeben, welcher Datentyp zurückgegeben wird. Im Fall der obigen Funktion ist das immer ein boolescher Wert.

5.34 Child-Theme-Check

Um Anpassungen an einem Theme vorzunehmen, das von jemand anderem entwickelt wurde, werden immer Child-Themes empfohlen, damit die Änderungen bei einem Theme-Update nicht überschrieben werden. Das Problem hierbei ist jedoch, dass eventuelle Sicherheitslücken des Parent-Themes mit in das Child-Theme übernommen und darin konserviert werden. Wenn das Parent-Theme die Lücke schließt, bleibt sie im Child-Theme – und damit auf der Live-Seite – erhalten.

Torsten Landsiedel hat dieses „Dilemma“ in einem Vortrag auf dem WordCamp Cologne 2015 angesprochen und gleich eine Lösung präsentiert: Das Child-Theme-Check-Plugin.[264, 265] Damit kann der Nutzer eines Child-Themes nach einem Update die Änderungen im Parent-Theme direkt neben dem Code des Child-Themes anschauen und so entscheiden, ob wichtige Änderungen dabei sind, die übernommen werden sollten, oder nicht.

[264] vgl. Landsiedel (2015)
[265] vgl. Landsiedel u.a.

Um das Plugin als Theme-Autor bestmöglich zu unterstützen, muss in jeder PHP-Datei am Anfang in einem Kommentar eine `@version`-Angabe eingefügt werden, die dann bei einer Änderung in dieser Datei angepasst wird. Wie das beispielsweise in der `index.php` aussieht, ist in Listing 140 abgebildet.

```
<?php
/**
 * Main template file
 *
 * @version 1.0
 */
get_header(); ?>
   <main role="main">
```

Listing 140: @version-Hinweis in der index.php

6 Upload und Review-Korrekturen des Themes

Nach dem ersten Upload eines Themes in das Theme-Verzeichnis wird dieses einer Review durch ein Mitglied des Theme-Review-Teams unterzogen. Den wenigen Freiwilligen in dem Team geschuldet, dauert es normalerweise einige Zeit, bis ein neues Theme mit der Review an der Reihe ist – durch die besondere Dringlichkeit im Fall des Hannover-Themes war es aber möglich, schneller einen Theme-Reviewer zu bekommen.

6.1 Upload ins Theme-Verzeichnis

Nachdem das Theme ein letztes Mal mit dem Theme-Check-Plugin geprüft und keine Fehler oder Warnungen gemeldet wurden, wird es in ein ZIP-Archiv gepackt und über den Link *Upload Your Theme* in das Theme-Verzeichnis geladen.[266] Bevor das Formular für den Upload aufgerufen wird, gelangt der Theme-Autor auf eine Seite, auf der auf die Richtlinien hingewiesen wird, die ein Theme erfüllen muss.

Nachdem das Theme hochgeladen wurde, wird im Theme-Trac ein neues Ticket erstellt – beim Hannover-Theme die Nummer 29486.[267] Hier findet die Kommunikation mit dem Reviewer statt. Wenn eine neue Version des Themes mit Korrekturen hochgeladen wird und bereits ein Ticket für das Theme offen ist, wird die neue Version in dieses Ticket integriert.

[266] vgl. WordPress: Theme Directory › Getting Started
[267] vgl. WordPress: Make WordPress Themes › Theme: Hannover

6.2 Korrekturen

Die Korrekturen sind hier nach den Dringlichkeitsstufen gruppiert, nicht nach den Kommentaren der Reviewer. Aufgeführt werden lediglich die Punkte, die auch angepasst wurden – ein paar konnten nach kurzer Diskussion ausgeräumt werden und erhalten bleiben.

6.2.1 Notwendig

Die Core-Option zur Änderung der Header-Farbe im Customizer muss von jedem Theme-Autor selbst integriert werden, da der Core das Markup für die CSS-Selektoren nicht kennen kann. Diese Umsetzung wurde bisher nicht getätigt und muss deshalb nun nachgeholt werden. Notwendig ist dafür eine kleine Funktion in der `customizer.php`.

```
function hannover_header_textcolor() {
   if ( get_theme_mod( 'header_textcolor' ) ) { ?>
      <style type="text/css">
         #header,
         #header a {
            color: <?php echo '#' . get_theme_mod('header_textcolor');
?>;
         }
      </style>
   <?php }
}

add_action( 'wp_head', 'hannover_header_textcolor' );
```

Listing 141: hannover_header_textcolor() in der customizer.php

Listing 141 zeigt die Funktion, in der zuerst geprüft wird, ob eine Header-Textfarbe eingetragen wurde – dafür wird `get_theme_mod()` mit dem Parameter `header_textcolor` verwendet.[268] Im Anschluss wird ein `style`-Element geöffnet und darin die Selektoren angegeben, die alle Texte in dem Header ansprechen. Als Wert für `color` wird der Wert der Customizer-Einstellung übergeben.

Eine weitere notwendige Korrektur ist der Einsatz von `if ( have_posts() )`, bevor mit `while ( have_posts() )` die Post-Schleife durchlaufen wird. Diese Anpassung ist in den Dateien `single.php`, `page.php`, `slider-front-page.php` sowie `no-sidebar-page.php` umzusetzen – beispielhaft für alle vier zeigt Listing 142 diese Anpassung in der `page.php`.

[268] vgl. WordPress: Code Reference › Functions › get_theme_mod

```
<?php if ( have_posts() ) {
   while ( have_posts() ) {
      the_post();
      get_template_part( 'template-parts/content', 'page' );
      if ( comments_open() || get_comments_number() ) {
         comments_template( '', true );
      }
   }
}
```

Listing 142: Prüfung von have_posts() vor Schleifendurchlauf in der page.php

Zudem ist in diesem Bereich die Bereitstellung einer Nachricht, wenn keine Beiträge gefunden wurden, `if ( have_posts() )` also zu keinem Ergebnis geführt hat, notwendig. Dafür wird eine neue Datei `content-none.php` im `template-parts`-Ordner erstellt, die lediglich eine Nachricht für den Nutzer anzeigt – der Inhalt dieser Datei ist in Listing 143 abgebildet.

```
<?php
/**
 * Template part for no posts found with have_posts()
 *
 * @version 1.0.6
 */
?>
<article class="post">
   <p><?php _e( 'Sorry, no posts matched your criteria.',
         'hannover' ); ?></p>
</article>
```

Listing 143: content-none.php

Um die Meldung auszugeben, wenn keine Beiträge gefunden wurden, muss in allen Dateien mit `if ( have_posts() )` ein `else`-Zweig eingefügt werden, der beispielhaft erneut an der `page.php` gezeigt wird.

```
<?php if ( have_posts() ) {
   while ( have_posts() ) {
      the_post();
      get_template_part( 'template-parts/content', 'page' );
      if ( comments_open() || get_comments_number() ) {
         comments_template( '', true );
      }
   }
} else {
   get_template_part( 'template-parts/content', 'none' );
} ?>
```

Listing 144: Einfügen des content-none-Templates in der page.php

In Listing 144 wird mit `get_template_part()` die erstellte Datei eingebunden, wenn keine Beiträge gefunden werden.[269]

Die letzte notwendige Korrektur aus der ersten Review betrifft die doppelten Anführungszeichen in Text-Strings, die vom Theme-Check-Plugin als spezielle Zeichen gekennzeichnet werden und durch " ersetzt werden sollen.

Anpassung der Lösung für den Ausschluss der Portfolio-Elemente vom Blog

Bei der Admin-Review hat Justin Tadlock angemerkt, dass die genutzte Lösung für den Ausschluss der Portfolio-Elemente vom Blog so nicht umgesetzt werden kann, da sie mit Plugins kollidieren wird, die für die Blog-Ausgabe die Main-Query erwarten und keine Custom-Query, die das Hannover-Theme bisher in der `index.php` einsetzt. Die richtige Lösung ist hier die Nutzung des `pre_get_posts`-Hooks, der vor Durchführung der Query ausgeführt wird.[270,271]

Durch diese veränderte Lösung kann die `index.php` deutlich schlanker dargestellt werden, wie der abgebildete Ausschnitt in Listing 145 zeigt. Die gesamte Logik für den Ausschluss der Portfolio-Elemente konnte entfernt werden, es verbleibt die normale Durchführung der Loop.

```
<?php if ( have_posts() ) {
   if ( is_home() && ! is_front_page() ) { ?>
      <header>
         <h1 class="page-title screen-reader-text">
            <?php single_post_title(); ?></h1>
      </header>
   <?php }
   while ( have_posts() ) {
      the_post();
         get_template_part( 'template-parts/content',
            get_post_format() );
      }
   } else {
      get_template_part( 'template-parts/content', 'none' );
```

Listing 145: Ausschnitt der korrigierten index.php nach Admin-Review

Um die Portfolio-Elemente dennoch ausschließen zu können, kommt wie bereits erwähnt der Hook `pre_get_posts` zum Einsatz. Die komplette Funktion ist in Listing 146 zu sehen.

[269] vgl. WordPress: Code Reference › Functions › get_template_part
[270] vgl. Tadlock: Make WordPress Themes › Theme: Hannover › Comment 29
[271] vgl. WordPress: Code Reference › Hooks › pre_get_posts

```
function hannover_exlude_portfolio_elements_from_blog( $query ) {
   if ( $query->is_home() && $query->is_main_query() ) {
      $exclude_portfolio_elements = get_theme_mod(
         'exclude_portfolio_elements_from_blog' );
      if ( $exclude_portfolio_elements == 'checked' ) {
         $use_portfolio_category = get_theme_mod(
            'portfolio_from_category' );
         $portfolio_category     = get_theme_mod( 'portfolio_category' );
         if ( $use_portfolio_category == 'checked'
            && $portfolio_category != '' ) {
            $query->set( 'cat', "-$portfolio_category" );
         } else {
            $tax_query = array(
               array(
                  'taxonomy' => 'post_format',
                  'field'    => 'slug',
                  'terms'    => array( 'post-format-gallery',
                     'post-format-image' ),
                  'operator' => 'NOT IN'
               )
            );
            $query->set( 'tax_query', $tax_query );
         }
      }
   }
}

add_action( 'pre_get_posts',
   'hannover_exlude_portfolio_elements_from_blog' );
```

Listing 146: hannover_exclude_portfolio_elemente_from_blog() in der functions.php

Über den `$query`-Parameter ist das Query-Objekt zugänglich, das nach Ausführung des `pre_get_posts`-Hooks zur Anzeige der Beiträge genutzt wird. Mit `$query->is_home()` und `$query->is_main_query()` wird zunächst geprüft, ob die Query auf der Blog-Seite ausgeführt wird und die Main-Query ist.[272, 273] Anschließend wird mit bekanntem Code aus Listing 67 von Seite 73 ermittelt, ob Portfolio-Elemente vom Blog ausgeschlossen werden sollen. Wenn das der Fall ist und die Elemente durch eine Kategorie gekennzeichnet sind, wird wie im Codex beschrieben mit `$query->set( 'cat', "-$portfolio_category" );` die Portfolio-Kategorie mit einem vorangestellten `-` in den `cat`-Schlüssel des Argument-Arrays für die Query eingefügt.[274] Nach demselben Verfahren kann der `tax_query`-Schlüssel gefüllt werden, wenn alle Bild- und Galerie-Beiträge die Portfolio-Elemente bilden.

[272] vgl. WordPress: Code Reference › Functions › is_home
[273] vgl. WordPress: Code Reference › Functions › is_main_query
[274] vgl. WordPress: Codex › Plugin API › Action Reference › pre_get_posts › Exclude categories on your main page

6.2.2 Stark Empfohlen

Die Art und Weise, wie das Theme die Portfolio-Funktion integriert, ist ohne kurze Anleitung für einen neuen Nutzer nicht einfach zu erfassen, weshalb der Reviewer empfohlen hat, eine Anleitung in die Readme-Datei einzufügen, um den Einstieg zu erleichtern.

In die Readme wurde daraufhin eine kurze nummerierte Liste eingefügt, die Schritt für Schritt eine mögliche Einrichtung des Themes erläutert, gefolgt von einer ausführlicheren Erklärung und den erweiterten Möglichkeiten durch den Customizer.

6.2.3 Empfohlen

Ähnliche Logik aus mehreren Dateien in eine Funktion auslagern

Die Logik der Argumentvorbereitung für die eigenen Querys in den Page-Template-Dateien `portfolio-page.php`, `portfolio-category-page.php` und `portfolio-archive-page.php` ist sehr ähnlich, weshalb sie sich besser in eine eigene Funktion auslagern lässt, die dann jeweils aus den Template-Dateien aufgerufen wird. Um das richtige Array für jedes der Seiten-Templates in einer Funktion zusammenzusetzen, muss nach den Seiten-Templates unterschieden werden – der Funktionsaufruf gibt neben dem Post-Objekt als String deshalb noch das Seiten-Template als String an.

```
function hannover_page_template_query_args( $post,
   $template = 'portfolio' ) {
   $archive_type           = get_theme_mod( 'portfolio_archive' );
   $archive_category       = get_theme_mod( 'portfolio_archive_category'
);
   $use_portfolio_category = get_theme_mod( 'portfolio_from_category' );
   $portfolio_category     = get_theme_mod( 'portfolio_category' );
   if ( $template == 'portfolio' ) {
      $elements_per_page = get_theme_mod( 'portfolio_elements_per_page',
         0 );
   } elseif ( $template == 'portfolio-archive' ) {
      $elements_per_page = get_theme_mod(
         'portfolio_archive_elements_per_page', 0 );
   } else {
      $elements_per_page = get_theme_mod(
         "portfolio_category_page_elements_per_page_$post->ID", 0 );
      $portfolio_category_page_category = get_theme_mod(
         "portfolio_category_page_$post->ID" );
   }
   $paged = ( get_query_var( 'paged' ) ) ? get_query_var( 'paged' ) : 1;
```

Listing 147: Anfang von hannover_page_template_query_args() in functions.php

In Listing 147 sind die ersten Zeilen dieser Funktion abgebildet. Für alle drei Seiten-Templates werden Informationen benötigt, ob und mit welcher Kategorie ein Archiv angelegt ist und ob die Portfolio-Elemente mit einer Kategorie gekennzeichnet wurden und mit welcher – die Informationen aus dem Customizer werden auf dieselbe Art ermittelt, wie bereits aus den verschiedenen Seiten-Template-Dateien bekannt. Die Anzahl der Elemente pro Seite muss für jedes Template anders ermittelt werden, weshalb der Wert von `$template` überprüft und je nach Wert die richtige Customizer-Einstellung ausgelesen wird – im Fall der Kategorie-Seiten wird für die Bildung der Einstellungs-ID mit `$post->ID` auf die Seiten-ID zugegriffen und neben der Anzahl pro Seite noch die Kategorie ermittelt, die angezeigt werden soll. Abschließend wird in dem Code-Block der Wert für den `paged`-Schlüssel, der bei allen Seiten-Templates gleich ist, in einer Variable gespeichert.

```
$tax_query = array( 'relation' => 'AND' );
if ( $elements_per_page == 0 ) {
   $elements_per_page = - 1;
}
$tax_query[] = array(
   'taxonomy' => 'post_format',
   'field'    => 'slug',
   'terms'    => array(
      'post-format-gallery',
      'post-format-image'
   ),
);
if ( $template != 'portfolio-archive' &&
   $archive_type == 'archive_category' ) {
   $tax_query[] = array(
      'taxonomy' => 'category',
      'field'    => 'term_id',
      'terms'    => array( $archive_category ),
      'operator' => 'NOT IN'
   );
}
if ( $template == 'portfolio-archive' &&
   $archive_category !== '' && $archive_type == 'archive_category' ) {
   $tax_query[] = array(
      'taxonomy' => 'category',
      'field'    => 'term_id',
      'terms'    => array( $archive_category ),
   );
}
if ( $template == 'portfolio-category-page' ) {
   $tax_query[] = array(
      'taxonomy' => 'category',
      'field'    => 'term_id',
      'terms'    => array( $portfolio_category_page_category ),
   );
}
if ( $use_portfolio_category == 'checked'
```

```
        && $portfolio_category !== '' ) {
        $tax_query[] = array(
            'taxonomy' => 'category',
            'field'    => 'term_id',
            'terms'    => array( $portfolio_category ),
        );
    }
    $args = array(
        'posts_per_page' => $elements_per_page,
        'paged'          => $paged,
        'tax_query'      => $tax_query
    );

    return $args;
}
```

Listing 148: Ende der hannover_page_template_query_args()

Listing 148 zeigt den zweiten Teil der Funktion. Die Beziehung zwischen den verschiedenen Unter-Arrays des `tax_query`-Schlüssels ist bei allen Seiten-Templates `AND`, weshalb dieser Wert ohne weitere Prüfung in `$tax_query` gespeichert werden kann. Wenn als Anzahl der Elemente pro Seite eine `0` ermittelt wurde, wird dieser Wert in `-1` verändert. Anschließend wird dem `tax_query`-Array ein weiterer Wert hinzugefügt, der für alle Seiten-Templates gilt: Die Einschränkung auf Galerie- und Bild-Beiträge.

Als Nächstes wird überprüft, ob es sich bei dem Seiten-Template nicht um das Archiv handelt und ob ein Archiv eingesetzt wird – ist das der Fall, wird `tax_query` ein weiteres Unter-Array hinzugefügt, das die Beiträge aus der Archiv-Kategorie ausschließt. Wenn es sich bei dem Template hingegen um das Archiv handelt und sowohl eine Archiv-Kategorie gesetzt ist als auch ein Archiv erzeugt werden soll, werden die Beiträge mit einer weiteren Bedingung für `tax_query` auf die Archiv-Kategorie eingeschränkt.

Handelt es sich jedoch um eine Kategorie-Seite, müssen die angezeigten Beiträge auch der Kategorie, die im Customizer für die aktuelle Seite gewählt ist, zugewiesen sein. Wenn das Portfolio anhand einer Kategorie gebildet wird und eine Kategorie gewählt wurde, dann muss unabhängig vom Seiten-Template noch die Bedingung eingefügt werden, dass die Beiträge in der Portfolio-Kategorie sind.

Abschließend wird in `$args` das Argument-Array aus `posts_per_page`, `paged` und dem zusammengesetzten Array für `tax_query` gebildet und zurückgegeben.

Der Aufruf der Funktion aus den einzelnen Dateien ist in Listing 150, Listing 151 und Listing 152 dargestellt – die jeweils bereits vorhandenen Teile um den Aufruf sind ausgegraut.

```
</header>
<?php $args = hannover_page_template_query_args( $post, 'portfolio' );
$portfolio_query = new WP_Query( $args );
$temp_query = $wp_query;
```

Listing 149: Aufruf von hannover_page_template_query_args() in der portfolio-page.php

```
</header>
<?php $args = hannover_page_template_query_args( $post,
   'portfolio-category-page' );
$portfolio_query = new WP_Query( $args );
$temp_query = $wp_query;
```

Listing 150: Aufruf von hannover_page_template_query_args() in der portfolio-category-page.php

```
</header>
<?php $args = hannover_page_template_query_args( $post,
   'portfolio-archive' );
$archive_query = new WP_Query( $args );
$temp_query = $wp_query;
```

Listing 151: Aufruf von hannover_page_template_query_args() in der portfolio-archive-page.php

Weitere kleine Anpassungen

Die beiden Dateien `customizer.php` und `class-hannover-social-menu-walker.php` werden vom Theme ursprünglich mit der `require`-Kontrollstruktur eingebunden. Auch wenn es unwahrscheinlich ist, dass diese Dateien zweimal eingebunden werden, kann das mit dem Einsatz von `require_once` im Vorhinein verhindert werden – die Anpassung ist in Listing 152 abgebildet.[275]

```
require_once get_template_directory() . '/inc/customizer.php';

require_once get_template_directory()
   . '/inc/class-hannover-social-menu-walker.php';
```

Listing 152: Nutzung von require_once statt require in der functions.php

Zudem wurden bei den Kommentarblöcken über den Funktionen teilweise falsche Werte für `@return` angegeben. Wenn eine Funktion kein `return`-Statement enthält, sondern direkt etwas ausgibt, handelt es sich um eine Void-Funktion. In diesem Fall wird `@return void` angegeben.

Darüber hinaus wurde in die JavaScript-Datei des SVG-Fallbacks die Lizenz eingefügt und drei Einheiten im CSS-Code gelöscht, bei der der Wert `0` war.

[275] vgl. PHP Group: Kontrollstrukturen › require_once

7 Fazit

Ergebnis der Bachelorarbeit ist ein WordPress-Theme, das allen WordPress-Nutzern kostenlos über das Theme-Verzeichnis zur Verfügung steht.[276] Die eingangs aufgeworfene Frage nach wünschenswerten Funktionen für Fotografen konnte durch die Analyse bestehender Websites beantwortet werden: Wichtig ist die Möglichkeit, eine Trennung der Portfolio-Elemente von der Blog-Funktion vorzunehmen. Die Portfolio-Elemente sollen alle auf einer Seite angezeigt werden, aber mit dem Theme sollte auch eine Aufteilung nach Kategorien möglich sein. Darüber hinaus sollte der Nutzer Portfolio-Elemente für ein Archiv markieren können, sodass sie nicht in der Portfolio-Übersicht auftauchen. Zudem werden Bilder oft als Slider angezeigt, statt einzeln untereinander. Für die Startseite sollte es die Möglichkeiten geben, einen Slider oder ein zufälliges Bild anzuzeigen.

Auch die Frage nach den Voraussetzungen, die für das Theme-Verzeichnis erfüllt werden müssen, konnte mithilfe des Theme-Review-Handbuchs beantwortet werden. So darf der Code keinerlei Fehler auswerfen und muss – wenn vorhanden – statt eigener Funktionen die Core-Funktionen von WordPress nutzen. Änderungen am Inhalt dürfen nicht fest im Code verankert sein, sondern müssen über Hooks oder Funktionsparameter umgesetzt werden. Außerdem muss sich ein Theme auf die Präsentation beschränken und darf keine Funktionen wie Custom-Post-Types oder Shortcodes integrieren, durch die der Nutzer Inhalte erstellen kann. Zudem muss das Theme dokumentiert werden und übersetzungsfähig sein. Der Theme-Code und alle verwendeten Ressourcen müssen unter der GPL oder einer kompatiblen Lizenz stehen, und das Theme darf weder den Begriff *WordPress* noch *Theme* im Namen tragen. Ein Theme darf weder Plugin-Code mitliefern noch Plugins für die korrekte Funktionalität voraussetzen. Weitere erforderliche Voraussetzungen sind ein Screenshot sowie die Nutzung von Funktionen für die sichere Ausgabe und Speicherung von Informationen. Bei einem eingebauten Link zum Entwickler dürfen auf seiner Seite nur Themes verkauft werden, die unter der GPL oder einer kompatiblen Lizenz stehen. Stylesheets und Skripte müssen, genau wie Template-Dateien, mit den vorgesehenen Funktionen eingebunden werden. Daneben gibt es noch einige empfohlene Richtlinien, deren Verletzung aber nicht zu einem Ausschluss aus dem Verzeichnis führt.

Alle Funktionen, die aus der Analyse der Fotografen-Websites hervorgegangen sind, konnten mit Bordmitteln umgesetzt werden, ohne die Richtlinien des Theme-Verzeichnisses zu

[276] vgl. Brinkmann: Theme-Verzeichnis › Hannover

verletzten. Der Code des fertigen Themes kann in einem GitHub-Repository eingesehen werden, wobei das Tag *v1.0.8* den Stand des Themes nach dieser Arbeit ohne zukünftige Updates festhält.[277, 278]

Durch die kleinschrittige Bearbeitung der Theme-Entwicklung ist die Arbeit hilfreich für alle, die sich näher mit der Theme-Programmierung auseinandersetzen möchten oder die sich in das Theme einarbeiten wollen, um Anpassungen vorzunehmen.

Hilfreich für den Verfasser der Arbeit war der Prozess der Theme-Entwicklung, weil er sich intensiv mit der Materie beschäftigen musste, die für seinen späteren Berufsweg wichtig ist. Besonders herausfordernd war die Umsetzung des Social-Media-Menüs mit den SVGs. Zudem hat der Verfasser sich erstmals detailliert mit der Nutzung der Überschriftenebenen nach Aspekten der Barrierefreiheit auseinandergesetzt und einen Weg der getrennten Zählung von Kommentaren und Trackbacks gefunden, der weniger Datenbankabfragen benötigt als die bis dahin genutzte Methode. Durch das Feedback des Reviewers wurde der Verfasser auf wichtige Punkte wie die Auslagerung von Logik aufmerksam gemacht, die an mehreren Stellen fast identisch genutzt wird, um weniger redundanten Code zu schreiben. Durch die Admin-Review wurde er außerdem auf eine praktische Lösung aufmerksam gemacht, um die Main-Query zu verändern, ohne die Funktion von Plugins zu gefährden.

[277] vgl. Brinkmann: hannover
[278] vgl. Brinkmann: hannover „v1.0.8“

Literaturverzeichnis

automattic: Genericons. http://genericons.com/ (Zugriff vom 22.12.2015).

automattic; Fields, Michael; Willett, Lance: „Monster Widget“-Plugin. https://wordpress.org/plugins/monster-widget/ (Zugriff vom 25.11.2015).

automattic; Jangda, Mohammad; Mills, Alex; Bachiyski, Nikolay; Ott, Thorsten; Bachhuber, Daniel; Betz, Josh; Nadarajah, Prasath; Daugherty, Dick: „Developer“-Plugin. https://wordpress.org/plugins/developer/ (Zugriff vom 25.11.2015).

Bennett, Chip: Using Sane Defaults in Themes. https://make.wordpress.org/themes/2014/07/09/using-sane-defaults-in-themes/ (Zugriff vom 26.11.2015).

Bennett, Chip: WPTRT/code-examples/customizer/sanitization-callbacks.php. https://github.com/WPTRT/code-examples/blob/master/customizer/sanitization-callbacks.php (Zugriff vom 20.12.2015).

Bennett, Chip (2013): How to fix pagination for custom loops? http://wordpress.stackexchange.com/a/120408 (Zugriff vom 23.12.2015).

Brinkmann, Florian: hannover. https://github.com/FlorianBrinkmann/hannover/ (Zugriff vom 05.01.2016).

Brinkmann, Florian: hannover „v1.0.8“. https://github.com/FlorianBrinkmann/hannover/tree/v1.0.8 (Zugriff vom 14.01.2016).

Brinkmann, Florian: Tag „v1.0.8“ – hannover/inc/class-hannover-social-walker.php. https://github.com/FlorianBrinkmann/hannover/blob/v1.0.8/inc/class-hannover-social-menu-walker.php (Zugriff vom 14.01.2016).

Brinkmann, Florian: Tag „v1.0.8“ – hannover/js/svg4everybody.js. https://github.com/FlorianBrinkmann/hannover/blob/v1.0.8/js/svg4everybody.js (Zugriff vom 14.01.2016).

Brinkmann, Florian: Tag „v1.0.8“ – hannover/svg/social-media-icons.svg. https://github.com/FlorianBrinkmann/hannover/blob/v1.0.8/svg/social-media-icons.svg (Zugriff vom 14.01.2016).

Brinkmann, Florian: Theme-Verzeichnis › Hannover. https://de.wordpress.org/themes/hannover/ (Zugriff vom 14.01.2016).

Can I use: SVG (basic support). http://caniuse.com/#feat=svg (Zugriff vom 31.12.2015).

Carlson, Ty; trenzterra; Irie, Takashi; Maharzan, Chandra; Vorotnov, Ihor; Klein, Fränk; Moore, Philip A.: twentysixteen/js/functions.js. https://github.com/WordPress/twentysixteen/blob/4c56411d81155e696ee57723c6d5a911ba4cb45f/js/functions.js (Zugriff vom 16.12.2015).

Castaneda, Jose (2015): Theme Review Tools. https://make.wordpress.org/themes/2015/07/23/theme-review-tools/ (Zugriff vom 25.11.2015).

do.media: Help Document – Invictus › III. Adding Photos to your gallery. http://help.doitmax.de/invictus/#galleryIII (Zugriff vom 25.11.2015).

Friedman, Vitaly (Hrsg.), 2015: *The smashing book #5. Real-life responsive web design.* Freiburg: Smashing Media GmbH.

Gandy, Dave: Font Awesome. http://fontawesome.io/ (Zugriff vom 22.12.2015).

inTheme: Alerts – Fluxus. http://inthe.me/demo/fluxus/features/alerts/ (Zugriff vom 25.11.2015).

jQuery: .append. http://api.jquery.com/append/ (Zugriff vom 15.12.2015).

jQuery: .click. http://api.jquery.com/click/ (Zugriff vom 15.12.2015).

jQuery: .ready. http://api.jquery.com/ready/ (Zugriff vom 15.12.2015).

jQuery: .wrap. http://api.jquery.com/wrap/ (Zugriff vom 14.12.2015).

Koster, James (2015): Explore WordPress' free theme options. In: net, H. 274, S. 108-112.

Landsiedel, Torsten (2015): Torsten Landsiedel: Das Child Theme Dilemma. http://wordpress.tv/2015/07/15/torsten-landsiedel-das-child-theme-dilemma/ (Zugriff vom 27.12.2015).

Landsiedel, Torsten; Wiechers, Ralf; Staude, Frank; Hübinger, Caspar; Altenburg, Hinnerk: Child Theme Check. https://wordpress.org/plugins/child-theme-check/ (Zugriff vom 27.12.2015).

Mills, Alex: „Regenerate Thumbnails"-Plugin. https://wordpress.org/plugins/regenerate-thumbnails/ (Zugriff vom 25.11.2015).

Mozilla: Webtechnologien für Entwickler | MDN. https://developer.mozilla.org/de/docs/Web (Zugriff vom 04.01.2016).

Nacin, Andrew: „Log Deprecated Notices"-Plugin. https://wordpress.org/plugins/log-deprecated-notices/ (Zugriff vom 25.11.2015).

Neal, Jonathan: svg4everybody. https://github.com/jonathantneal/svg4everybody (Zugriff vom 22.12.2015).

Novotny, Michael: WP Test. http://wptest.io/ (Zugriff vom 26.11.2015).

Obenland, Konstantin: Core-Trac › Ticket #30589 – Comments navigation template tags. https://core.trac.wordpress.org/ticket/30589 (Zugriff vom 06.12.2015).

Obenland, Konstantin (2015): Title Tags in 4.1. https://make.wordpress.org/core/2014/10/29/title-tags-in-4-1/ (Zugriff vom 29.11.2015).

Perrier, Jean-Yves; Zartner, Sebastian; Scarfone, Karen; Stone-Thompson, Peter; Shepherd, Eric; medicdude; ethertank; Sonne, Christian; dhodder; Scholz, Florian; Wilsson, Jonathan; Hobson, Trevor; Swisher, Janet: MDN › HTML › <nav>. https://developer.mozilla.org/de/docs/Web/HTML/Element/nav (Zugriff vom 28.11.2015).

PHP Group: array_shift. http://php.net/manual/de/function.array-shift.php (Zugriff vom 14.12.2015).

PHP Group: count. http://php.net/manual/de/function.count.php (Zugriff vom 30.12.2015).

PHP Group: date. http://php.net/manual/de/function.date.php (Zugriff vom 30.11.2015).

PHP Group: explode. http://php.net/manual/de/function.explode.php (Zugriff vom 06.12.2015).

PHP Group: in_array. http://php.net/manual/de/function.in-array.php (Zugriff vom 26.12.2015).

PHP Group: Kontrollstrukturen › require. http://php.net/manual/de/function.require.php (Zugriff vom 05.12.2015).

PHP Group: Kontrollstrukturen › require_once. http://php.net/manual/de/function.require-once.php (Zugriff vom 03.01.2016).

PHP Group: PHP-Handbuch. http://php.net/manual/de/ (Zugriff vom 04.01.2016).

PHP Group: preg_match. http://php.net/manual/de/function.preg-match.php (Zugriff vom 08.12.2015).

PHP Group: preg_replace. http://php.net/manual/de/function.preg-replace.php (Zugriff vom 16.12.2015).

PHP Group: printf. http://php.net/manual/de/function.printf.php (Zugriff vom 30.11.2015).

PHP Group: shuffle. http://php.net/manual/de/function.shuffle.php (Zugriff vom 15.12.2015).

PHP Group: sprintf. http://php.net/manual/de/function.sprintf.php (Zugriff vom 30.11.2015).

Rietveld, Rian (2014): HTML5 Headings in WordPress: A11y versus SEO? http://blog.rrwd.nl/2014/11/21/html5-headings-in-wordpress-lets-fight/ (Zugriff vom 27.11.2015).

Sari, Dennis; Zartner, Sebastian; Perrier, Jean-Yves; Kraut, Tim: MDN › HTML › <time>. https://developer.mozilla.org/de/docs/Web/HTML/Element/time (Zugriff vom 08.12.2015).

Schilling, Dominik (2014): Language chooser in 4.0. https://make.wordpress.org/core/2014/09/05/language-chooser-in-4-0/ (Zugriff vom 17.01.2016).

Schulp, Michelle: Template Hierarchy › Visual Overview. https://developer.wordpress.org/themes/basics/template-hierarchy/#visual-overview (Zugriff vom 29.11.2015).

Soueidan, Sara (2015): Mastering SVG for Responsive Web Design, in *The smashing book #5. Real-life responsive web design*, hrsg. v. Friedman, Vitaly, S. 127-208. Freiburg: Smashing Media GmbH.

Tadlock, Justin (2015): Details on the new theme settings (customizer) guideline. https://make.wordpress.org/themes/2015/04/22/details-on-the-new-theme-settings-customizer-guideline/ (Zugriff vom 19.12.2015).

Tadlock, Justin: Make WordPress Themes › Theme: Hannover › Comment 29. https://themes.trac.wordpress.org/ticket/29486#comment:29 (Zugriff vom 14.01.2016).

Tadlock, Justin (2015): The theme review team's content creation discussion. http://justintadlock.com/archives/2015/05/28/the-theme-review-teams-content-creation-discussion (Zugriff vom 27.11.2015).

Tadlock, Justin (2015): Meeting Minutes 06/11/15 › Kommentar. https://make.wordpress.org/themes/2015/06/13/meeting-minutes-061115/#comment-42032 (Zugriff vom 25.11.2015).

Tadlock, Justin (2015): Themes should be 100% GPL. https://make.wordpress.org/themes/2015/08/15/themes-should-be-100-gpl/ (Zugriff vom 25.11.2015).

ThemeForest: ANAN – For Photography Creative Portfolio. http://themeforest.net/item/anan-for-photography-creative-portfolio/174507 (Zugriff vom 09.11.2015).

ThemeForest: Meistverkaufte WordPress-Themes mit Tags „photography“ und „portfolio“. http://themeforest.net/search?utf8=%E2%9C%93&term=&view=list&sort=sales&date=&category=wordpress&tags=photography&tags=portfolio&price_min=&price_max=&sales=&rating_min= (Zugriff vom 09.11.2015).

ThemeForest: Premium WordPress Themes & WordPress Templates. http://themeforest.net/category/wordpress (Zugriff vom 09.11.2015).

Valutis, Osvaldas: Image Lightbox › Demo. http://osvaldas.info/examples/image-lightbox-responsive-touch-friendly/ (Zugriff vom 16.12.2015).

Valutis, Osvaldas: Image Lightbox, Responsive and Touch-friendly. http://osvaldas.info/image-lightbox-responsive-touch-friendly (Zugriff vom 15.12.2015).

vlastuin: Code Reference › Functions › bloginfo › Show Character Set. https://developer.wordpress.org/reference/functions/bloginfo/#comment-515 (Zugriff vom 26.11.2015).

Westwood, Peter: „WordPress Beta Tester“-Plugin. https://wordpress.org/plugins/wordpress-beta-tester/ (Zugriff vom 25.11.2015).

Williams, Brad; Damstra, David; Stern, Hal (2015): *Professional WordPress. Design and development.* Third edition. Indianapolis, Indiana: Wrox.

Wojciechowski, Bartosz: Owl Carousel. http://www.owlcarousel.owlgraphic.com (Zugriff vom 14.12.2015).

Wojciechowski, Bartosz: Owl Carousel › Events. http://www.owlcarousel.owlgraphic.com/docs/api-events.html (Zugriff vom 15.12.2015).

Wojciechowski, Bartosz: Owl Carousel › Installation. http://www.owlcarousel.owlgraphic.com/docs/started-installation.html (Zugriff vom 14.12.2015).

Wojciechowski, Bartosz: Owl Carousel › Options. http://www.owlcarousel.owlgraphic.com/docs/api-options.html (Zugriff vom 14.12.2015).

Wood, Samuel: Passing parameters from PHP to Javascripts in plugins. http://ottopress.com/2010/passing-parameters-from-php-to-javascripts-in-plugins/ (Zugriff vom 14.12.2015).

Wood, Samuel; Obenland, Konstantin (2015): theme-check/checks/style_needed.php. https://github.com/Otto42/theme-check/blob/542e2b520c94f5455cbf86f8d59d3bc6555aac1e/checks/style_needed.php#L11-L16 (Zugriff vom 26.11.2015).

Wood, Samuel; Prosser, Simon: „Theme Check“-Plugin. https://wordpress.org/plugins/theme-check/ (Zugriff vom 25.11.2015).

WordPress: array_unique. http://php.net/manual/de/function.array-unique.php (Zugriff vom 06.12.2015).

WordPress: Code Reference. https://developer.wordpress.org/reference/ (Zugriff vom 04.01.2016).

WordPress: Code Reference › Classes › Walker_Nav_Menu. https://developer.wordpress.org/reference/classes/walker_nav_menu/ (Zugriff vom 22.12.2015).

WordPress: Code Reference › Classes › Walker_Nav_Menu › Walker_Nav_Menu::start_el. https://developer.wordpress.org/reference/classes/walker_nav_menu/start_el/ (Zugriff vom 22.12.2015).

WordPress: Code Reference › Classes › WP_Customize_Manager. https://developer.wordpress.org/reference/classes/wp_customize_manager/ (Zugriff vom 17.12.2015).

WordPress: Code Reference › Classes › WP_Customize_Manager › WP_Customize_Manager::add_control. https://developer.wordpress.org/reference/classes/wp_customize_manager/add_control/ (Zugriff vom 15.12.2015).

WordPress: Code Reference › Classes › WP_Customize_Manager › WP_Customize_Manager::add_panel. https://developer.wordpress.org/reference/classes/wp_customize_manager/add_panel/ (Zugriff vom 16.12.2015).

WordPress: Code Reference › Classes › WP_Customize_Manager › WP_Customize_Manager::add_section. https://developer.wordpress.org/reference/classes/wp_customize_manager/add_section/ (Zugriff vom 16.12.2015).

WordPress: Code Reference › Classes › WP_Customize_Manager › WP_Customize_Manager::add_setting. https://developer.wordpress.org/reference/classes/wp_customize_manager/add_setting/ (Zugriff vom 16.12.2015).

WordPress: Code Reference › Classes › wpdb. https://developer.wordpress.org/reference/classes/wpdb/ (Zugriff vom 14.12.2015).

WordPress: Code Reference › Classes › wpdb › wpdb::get_var. https://developer.wordpress.org/reference/classes/wpdb/get_var/ (Zugriff vom 15.12.2015).

WordPress: Code Reference › Functions › __. https://developer.wordpress.org/reference/functions/__/ (Zugriff vom 30.12.2015).

WordPress: Code Reference › Functions › _e. https://developer.wordpress.org/reference/functions/_e/ (Zugriff vom 30.12.2015).

WordPress: Code Reference › Functions › _n. https://developer.wordpress.org/reference/functions/_n/ (Zugriff vom 17.01.2016).

WordPress: Code Reference › Functions › absint. https://developer.wordpress.org/reference/functions/absint/ (Zugriff vom 18.12.2015).

WordPress: Code Reference › Functions › add_action. https://developer.wordpress.org/reference/functions/add_action/ (Zugriff vom 17.12.2015).

WordPress: Code Reference › Functions › add_filter. https://developer.wordpress.org/reference/functions/add_filter/ (Zugriff vom 17.12.2015).

WordPress: Code Reference › Functions › add_theme_support. https://developer.wordpress.org/reference/functions/add_theme_support/ (Zugriff vom 29.11.2015).

WordPress: Code Reference › Functions › array_merge. http://php.net/manual/de/function.array-merge.php (Zugriff vom 06.12.2015).

WordPress: Code Reference › Functions › attachment_url_to_postid. https://developer.wordpress.org/reference/functions/attachment_url_to_postid/ (Zugriff vom 04.12.2015).

WordPress: Code Reference › Functions › bloginfo › Source. https://developer.wordpress.org/reference/functions/bloginfo/#source-code (Zugriff vom 25.11.2015).

WordPress: Code Reference › Functions › body_class. https://developer.wordpress.org/reference/functions/body_class/ (Zugriff vom 25.11.2015).

WordPress: Code Reference › Functions › comment_author_link. https://developer.wordpress.org/reference/functions/comment_author_link/ (Zugriff vom 20.12.2015).

WordPress: Code Reference › Functions › comment_class. https://developer.wordpress.org/reference/functions/comment_class/ (Zugriff vom 18.12.2015).

WordPress: Code Reference › Functions › comment_form. https://developer.wordpress.org/reference/functions/comment_form/ (Zugriff vom 18.12.2015).

WordPress: Code Reference › Functions › comment_ID. https://developer.wordpress.org/reference/functions/comment_id/ (Zugriff vom 18.12.2015).

WordPress: Code Reference › Functions › comment_reply_link. https://developer.wordpress.org/reference/functions/comment_reply_link/ (Zugriff vom 19.12.2015).

WordPress: Code Reference › Functions › comment_text. https://developer.wordpress.org/reference/functions/comment_text/ (Zugriff vom 18.12.2015).

WordPress: Code Reference › Functions › comments_open. https://developer.wordpress.org/reference/functions/comments_open/ (Zugriff vom 19.12.2015).

WordPress: Code Reference › Functions › comments_template. https://developer.wordpress.org/reference/functions/comments_template/ (Zugriff vom 20.12.2015).

WordPress: Code Reference › Functions › edit_comment_link. https://developer.wordpress.org/reference/functions/edit_comment_link/ (Zugriff vom 20.12.2015).

WordPress: Code Reference › Functions › esc_url. https://developer.wordpress.org/reference/functions/esc_url/ (Zugriff vom 26.11.2015).

WordPress: Code Reference › Functions › get_avatar. https://developer.wordpress.org/reference/functions/get_avatar/ (Zugriff vom 19.12.2015).

WordPress: Code Reference › Functions › get_bloginfo. https://developer.wordpress.org/reference/functions/get_bloginfo/ (Zugriff vom 26.11.2015).

WordPress: Code Reference › Functions › get_categories.
https://developer.wordpress.org/reference/functions/get_categories/
(Zugriff vom 31.12.2015).

WordPress: Code Reference › Functions › get_comment_date.
https://developer.wordpress.org/reference/functions/get_comment_date/
(Zugriff vom 18.12.2015).

WordPress: Code Reference › Functions › get_comment_link.
https://developer.wordpress.org/reference/functions/get_comment_link/
(Zugriff vom 18.12.2015).

WordPress: Code Reference › Functions › get_comment_time.
https://developer.wordpress.org/reference/functions/get_comment_time/
(Zugriff vom 18.12.2015).

WordPress: Code Reference › Functions › get_comments.
https://developer.wordpress.org/reference/functions/get_comments/
(Zugriff vom 15.12.2015).

WordPress: Code Reference › Functions › get_comments_number.
https://developer.wordpress.org/reference/functions/get_comments_number/
(Zugriff vom 12.12.2015).

WordPress: Code Reference › Functions › get_footer.
https://developer.wordpress.org/reference/functions/get_footer/
(Zugriff vom 10.12.2015).

WordPress: Code Reference › Functions › get_header.
https://developer.wordpress.org/reference/functions/get_header/
(Zugriff vom 30.11.2015).

WordPress: Code Reference › Functions › get_header_image.
https://developer.wordpress.org/reference/functions/get_header_image/
(Zugriff vom 26.11.2015).

WordPress: Code Reference › Functions › get_page_template_slug.
https://developer.wordpress.org/reference/functions/get_page_template_slug/
(Zugriff vom 15.12.2015).

WordPress: Code Reference › Functions › get_post.
https://developer.wordpress.org/reference/functions/get_post/
(Zugriff vom 06.12.2015).

WordPress: Code Reference › Functions › get_post_format.
https://developer.wordpress.org/reference/functions/get_post_format/
(Zugriff vom 02.12.2015).

WordPress: Code Reference › Functions › get_post_galleries.
https://developer.wordpress.org/reference/functions/get_post_galleries/
(Zugriff vom 06.12.2015).

WordPress: Code Reference › Functions › get_posts.
https://developer.wordpress.org/reference/functions/get_posts/
(Zugriff vom 06.12.2015).

WordPress: Code Reference › Functions › get_search_form.
https://developer.wordpress.org/reference/functions/get_search_form/
(Zugriff vom 27.12.2015).

WordPress: Code Reference › Functions › get_search_query.
https://developer.wordpress.org/reference/functions/get_search_query/
(Zugriff vom 27.12.2015).

WordPress: Code Reference › Functions › get_sidebar.
https://developer.wordpress.org/reference/functions/get_sidebar/
(Zugriff vom 30.11.2015).

WordPress: Code Reference › Functions › get_stylesheet_directory_uri.
https://developer.wordpress.org/reference/functions/get_stylesheet_directory_uri/
(Zugriff vom 14.12.2015).

WordPress: Code Reference › Functions › get_template_directory.
https://developer.wordpress.org/reference/functions/get_template_directory/
(Zugriff vom 10.12.2015).

WordPress: Code Reference › Functions › get_template_part.
https://developer.wordpress.org/reference/functions/get_template_part/
(Zugriff vom 01.12.2015).

WordPress: Code Reference › Functions › get_template_part_directory.
https://developer.wordpress.org/reference/functions/get_template_directory_uri/
(Zugriff vom 14.12.2015).

WordPress: Code Reference › Functions › get_the_author.
https://developer.wordpress.org/reference/functions/get_the_author/
(Zugriff vom 03.12.2015).

WordPress: Code Reference › Functions › get_the_category.
https://developer.wordpress.org/reference/functions/get_the_category/
(Zugriff vom 03.12.2015).

WordPress: Code Reference › Functions › get_the_category_list.
https://developer.wordpress.org/reference/functions/get_the_category_list/
(Zugriff vom 03.12.2015).

WordPress: Code Reference › Functions › get_the_date.
https://developer.wordpress.org/reference/functions/get_the_date/
(Zugriff vom 29.11.2015).

WordPress: Code Reference › Functions › get_the_tag_list.
https://developer.wordpress.org/reference/functions/get_the_tag_list/
(Zugriff vom 02.12.2015).

WordPress: Code Reference › Functions › get_the_tags.
https://developer.wordpress.org/reference/functions/get_the_tags/
(Zugriff vom 02.12.2015).

WordPress: Code Reference › Functions › get_the_time.
https://developer.wordpress.org/reference/functions/get_the_time/
(Zugriff vom 02.12.2015).

WordPress: Code Reference › Functions › get_the_title.
https://developer.wordpress.org/reference/functions/get_the_title/
(Zugriff vom 30.12.2015).

WordPress: Code Reference › Functions › get_theme_mod.
https://developer.wordpress.org/reference/functions/get_theme_mod/
(Zugriff vom 21.12.2015).

WordPress: Code Reference › Functions › has_category.
https://developer.wordpress.org/reference/functions/has_category/
(Zugriff vom 23.12.2015).

WordPress: Code Reference › Functions › has_excerpt.
https://developer.wordpress.org/reference/functions/has_excerpt/
(Zugriff vom 29.12.2015).

WordPress: Code Reference › Functions › has_nav_menu.
https://developer.wordpress.org/reference/functions/has_nav_menu/
(Zugriff vom 16.12.2015).

WordPress: Code Reference › Functions › has_post_thumbnail.
https://developer.wordpress.org/reference/functions/has_post_thumbnail/
(Zugriff vom 04.12.2015).

WordPress: Code Reference › Functions › have_comments.
https://developer.wordpress.org/reference/functions/have_comments/
(Zugriff vom 18.12.2015).

WordPress: Code Reference › Functions › have_posts.
https://developer.wordpress.org/reference/functions/have_posts/
(Zugriff vom 29.11.2015).

WordPress: Code Reference › Functions › header_image.
https://developer.wordpress.org/reference/functions/header_image/
(Zugriff vom 25.11.2015).

WordPress: Code Reference › Functions › home_url.
https://developer.wordpress.org/reference/functions/home_url/
(Zugriff vom 25.11.2015).

WordPress: Code Reference › Functions › is_front_page.
https://developer.wordpress.org/reference/functions/is_front_page/
(Zugriff vom 25.11.2015).

WordPress: Code Reference › Functions › is_home.
https://developer.wordpress.org/reference/functions/is_home/
(Zugriff vom 25.11.2015).

WordPress: Code Reference › Functions › is_main_query.
https://developer.wordpress.org/reference/functions/is_main_query/
(Zugriff vom 14.01.2016).

WordPress: Code Reference › Functions › is_singular.
https://developer.wordpress.org/reference/functions/is_singular/
(Zugriff vom 25.11.2015).

WordPress: Code Reference › Functions › language_attributes.
https://developer.wordpress.org/reference/functions/language_attributes/
(Zugriff vom 25.11.2015).

WordPress: Code Reference › Functions › load_theme_textdomain.
https://developer.wordpress.org/reference/functions/load_theme_textdomain/
(Zugriff vom 26.12.2015).

WordPress: Code Reference › Functions › number_format_i18n.
https://developer.wordpress.org/reference/functions/number_format_i18n/
(Zugriff vom 01.12.2015).

WordPress: Code Reference › Functions › pings_open.
https://developer.wordpress.org/reference/functions/pings_open/
(Zugriff vom 25.11.2015).

WordPress: Code Reference › Functions › post_class.
https://developer.wordpress.org/reference/functions/post_class/
(Zugriff vom 30.11.2015).

WordPress: Code Reference › Functions › post_password_required.
https://developer.wordpress.org/reference/functions/post_password_required/
(Zugriff vom 30.12.2015).

WordPress: Code Reference › Functions › register_nav_menu.
https://developer.wordpress.org/reference/functions/register_nav_menu/
(Zugriff vom 26.11.2015).

WordPress: Code Reference › Functions › register_sidebar.
https://developer.wordpress.org/reference/functions/register_sidebar/
(Zugriff vom 03.12.2015).

WordPress: Code Reference › Functions › register-nav_menus.
https://developer.wordpress.org/reference/functions/register_nav_menus/
(Zugriff vom 01.12.2015).

WordPress: Code Reference › Functions › sanitize_key.
https://developer.wordpress.org/reference/functions/sanitize_key/
(Zugriff vom 17.12.2015).

WordPress: Code Reference › Functions › separate_comments.
https://developer.wordpress.org/reference/functions/separate_comments/
(Zugriff vom 16.12.2015).

WordPress: Code Reference › Functions › single_post_title.
https://developer.wordpress.org/reference/functions/single_post_title/
(Zugriff vom 30.12.2015).

WordPress: Code Reference › Functions › the_archive_description.
https://developer.wordpress.org/reference/functions/the_archive_description/
(Zugriff vom 04.12.2015).

WordPress: Code Reference › Functions › the_archive_title.
https://developer.wordpress.org/reference/functions/the_archive_title/
(Zugriff vom 16.12.2015).

WordPress: Code Reference › Functions › the_content.
https://developer.wordpress.org/reference/functions/the_content/
(Zugriff vom 15.12.2015).

WordPress: Code Reference › Functions › the_excerpt.
https://developer.wordpress.org/reference/functions/the_excerpt/
(Zugriff vom 11.12.2015).

WordPress: Code Reference › Functions › the_post.
https://developer.wordpress.org/reference/functions/the_post/
(Zugriff vom 02.12.2015).

WordPress: Code Reference › Functions › the_post_navigation.
https://developer.wordpress.org/reference/functions/the_post_navigation/
(Zugriff vom 03.12.2015).

WordPress: Code Reference › Functions › the_post_thumbnail. https://developer.wordpress.org/reference/functions/the_post_thumbnail/ (Zugriff vom 29.11.2015).

WordPress: Code Reference › Functions › the_posts_pagination. https://developer.wordpress.org/reference/functions/the_posts_pagination/ (Zugriff vom 07.12.2015).

WordPress: Code Reference › Functions › the_title. https://developer.wordpress.org/reference/functions/the_title/ (Zugriff vom 07.12.2015).

WordPress: Code Reference › Functions › update_option. https://developer.wordpress.org/reference/functions/update_option/ (Zugriff vom 26.12.2015).

WordPress: Code Reference › Functions › wp_enqueue_script. https://developer.wordpress.org/reference/functions/wp_enqueue_script/ (Zugriff vom 30.11.2015).

WordPress: Code Reference › Functions › wp_enqueue_style. https://developer.wordpress.org/reference/functions/wp_enqueue_style/ (Zugriff vom 30.11.2015).

WordPress: Code Reference › Functions › wp_footer. https://developer.wordpress.org/reference/functions/wp_footer/ (Zugriff vom 30.11.2015).

WordPress: Code Reference › Functions › wp_get_attachment_image. https://developer.wordpress.org/reference/functions/wp_get_attachment_image/ (Zugriff vom 04.12.2015).

WordPress: Code Reference › Functions › wp_head. https://developer.wordpress.org/reference/functions/wp_head/ (Zugriff vom 27.11.2015).

WordPress: Code Reference › Functions › wp_link_pages. https://developer.wordpress.org/reference/functions/wp_link_pages/ (Zugriff vom 15.12.2015).

WordPress: Code Reference › Functions › wp_list_comments. https://developer.wordpress.org/reference/functions/wp_list_comments/ (Zugriff vom 26.11.2015).

WordPress: Code Reference › Functions › wp_localize_script. https://developer.wordpress.org/reference/functions/wp_localize_script/ (Zugriff vom 14.12.2015).

WordPress: Code Reference › Functions › wp_nav_menu. https://developer.wordpress.org/reference/functions/wp_nav_menu/ (Zugriff vom 27.11.2015).

WordPress: Code Reference › Functions › wp_reset_postdata. https://developer.wordpress.org/reference/functions/wp_reset_postdata/ (Zugriff vom 04.12.2015).

WordPress: Code Reference › Hooks › comment_post. https://developer.wordpress.org/reference/hooks/comment_post/ (Zugriff vom 03.12.2015).

WordPress: Code Reference › Hooks › customize_register.
https://developer.wordpress.org/reference/hooks/customize_register/
(Zugriff vom 03.12.2015).

WordPress: Code Reference › Hooks › init.
https://developer.wordpress.org/reference/hooks/init/ (Zugriff vom 27.11.2015).

WordPress: Code Reference › Hooks › pre_get_posts.
https://developer.wordpress.org/reference/hooks/pre_get_posts/
(Zugriff vom 14.01.2016).

WordPress: Code Reference › Hooks › the_content.
https://developer.wordpress.org/reference/hooks/the_content/
(Zugriff vom 04.12.2015).

WordPress: Code Reference › Hooks › the_content_more_link.
https://developer.wordpress.org/reference/hooks/the_content_more_link/
(Zugriff vom 16.12.2015).

WordPress: Code Reference › Hooks › widget_categories_args.
https://developer.wordpress.org/reference/hooks/widget_categories_args/
(Zugriff vom 23.12.2015).

WordPress: Code Reference › Hooks › wp_head.
https://developer.wordpress.org/reference/hooks/wp_head/ (Zugriff vom 26.11.2015).

WordPress: Codex › add_theme_support › Custom Header.
https://codex.wordpress.org/Function_Reference/add_theme_support#Custom_Header
(Zugriff vom 29.11.2015).

WordPress: Codex › add_theme_support › Feed Links.
https://codex.wordpress.org/Function_Reference/add_theme_support#Feed_Links
(Zugriff vom 29.11.2015).

WordPress: Codex › add_theme_support › HTML5.
https://codex.wordpress.org/Function_Reference/add_theme_support#HTML5
(Zugriff vom 29.11.2015).

WordPress: Codex › add_theme_support › Post Thumbnails.
https://codex.wordpress.org/Function_Reference/add_theme_support#Post_Thumbnails
(Zugriff vom 30.11.2015).

WordPress: Codex › Class Reference › WP_Query › Taxonomy Parameters.
https://codex.wordpress.org/Class_Reference/WP_Query#Taxonomy_Parameters
(Zugriff vom 03.12.2015).

WordPress: Codex › Content Width. https://codex.wordpress.org/Content_Width
(Zugriff vom 26.12.2015).

WordPress: Codex › Function Reference › bloginfo › Parameters.
https://codex.wordpress.org/Function_Reference/bloginfo#Parameters
(Zugriff vom 26.11.2015).

WordPress: Codex › Function Reference › get_bloginfo › Parameters.
https://codex.wordpress.org/Function_Reference/get_bloginfo#Parameters
(Zugriff vom 26.11.2015).

WordPress: Codex › Introducion to Blogging › Pingbacks.
http://codex.wordpress.org/Introduction_to_Blogging#Pingbacks
(Zugriff vom 26.11.2015).

WordPress: Codex › L10n. https://codex.wordpress.org/L10n (Zugriff vom 26.11.2015).

WordPress: Codex › Plugin API › Action Reference › pre_get_posts › Exclude categories on your main page. https://codex.wordpress.org/Plugin_API/Action_Reference/pre_get_posts#Exclude_categories_on_your_main_page (Zugriff vom 14.01.2016).

WordPress: Codex › Post Formats › Supported Formats. https://codex.wordpress.org/Post_Formats#Supported_Formats (Zugriff vom 28.11.2015).

WordPress: Codex › Theme Development › Theme Stylesheet. https://codex.wordpress.org/Theme_Development#Theme_Stylesheet (Zugriff vom 26.11.2015).

WordPress: Codex › Theme Unit Test. http://codex.wordpress.org/Theme_Unit_Test (Zugriff vom 25.11.2015).

WordPress: Core-Trac › tags/4.3.1/src/wp-includes/post-template.php › Zeile 544. https://core.trac.wordpress.org/browser/tags/4.3.1/src/wp-includes/post-template.php#L544 (Zugriff vom 26.11.2015).

WordPress: Debugging in WordPress. https://codex.wordpress.org/Debugging_in_WordPress (Zugriff vom 25.11.2015).

WordPress: Function Reference › Functions › is_active_sidebar. https://developer.wordpress.org/reference/functions/is_active_sidebar/ (Zugriff vom 27.11.2015).

WordPress: Make WordPress Themes › Theme: Hannover. https://themes.trac.wordpress.org/ticket/29486 (Zugriff vom 03.01.2016).

WordPress: Plugin Handbook › Internationalization › How to Internationalize Your Plugin › Descriptions. https://developer.wordpress.org/plugins/internationalization/how-to-internationalize-your-plugin/#descriptions (Zugriff vom 17.01.2016).

WordPress: Plugin Handbook › Internationalization › How to Internationalize Your Plugin › Disambiguation by context. https://developer.wordpress.org/plugins/internationalization/how-to-internationalize-your-plugin/#disambiguation-by-context (Zugriff vom 17.01.2016).

WordPress: Theme Directory › Getting Started. https://wordpress.org/themes/getting-started/ (Zugriff vom 03.01.2016).

WordPress: Theme Directory › photoblogging-Tag. https://wordpress.org/themes/tags/photoblogging/ (Zugriff vom 02.11.2015).

WordPress: Theme Handbook › Advanced Theme Topics › Theme Options – The Customizer API › Controls. https://developer.wordpress.org/themes/advanced-topics/customizer-api/#controls (Zugriff vom 17.12.2015).

WordPress: Theme Handbook › Advanced Theme Topics › Theme Options – The Customizer API › Settings. https://developer.wordpress.org/themes/advanced-topics/customizer-api/#settings (Zugriff vom 17.12.2015).

WordPress: Theme Handbook › Internationalization › Text Domains. https://developer.wordpress.org/themes/functionality/internationalization/#text-domains (Zugriff vom 27.11.2015).

WordPress: Theme Handbook › Template Files Section › Page Template Files › Page Templates › Creating Custom Page Templates for Global Use.

https://developer.wordpress.org/themes/template-files-section/page-template-files/page-templates/#creating-custom-page-templates-for-global-use (Zugriff vom 12.12.2015).

WordPress: Theme Handbook › Template Files Section › Page Template Files › Page Templates › File Organization of Page Templates. https://developer.wordpress.org/themes/template-files-section/page-template-files/page-templates/#file-organization-of-page-templates (Zugriff vom 12.12.2015).

WordPress: Theme Handbook › Template Files Section › Post Template Files › index.php. https://developer.wordpress.org/themes/template-files-section/post-template-files/#index-php (Zugriff vom 30.11.2015).

WordPress: Theme Handbook › Theme Basics › Template Files › Common WordPress template files. https://developer.wordpress.org/themes/basics/template-files/#common-wordpress-template-files (Zugriff vom 29.11.2015).

WordPress: Theme Handbook › Theme Basics › Template Files › Template Partials. https://developer.wordpress.org/themes/basics/template-files/#template-partials (Zugriff vom 22.12.2015).

WordPress: Theme Review Handbook. https://make.wordpress.org/themes/handbook/ (Zugriff vom 04.01.2016).

WordPress: Theme Review Handbook › Accessibility › Required › Link Text. https://make.wordpress.org/themes/handbook/review/accessibility/required/#link-text (Zugriff vom 03.12.2015).

WordPress: Theme Review Handbook › Core Functionality and Features. https://make.wordpress.org/themes/handbook/review/recommended/core-functionality-and-features/ (Zugriff vom 25.11.2015).

WordPress: Theme Review Handbook › Design. https://make.wordpress.org/themes/handbook/review/recommended/design/ (Zugriff vom 25.11.2015).

WordPress: Theme Review Handbook › Documentation. https://make.wordpress.org/themes/handbook/review/recommended/documentation/ (Zugriff vom 25.11.2015).

WordPress: Theme Review Handbook › Recommended. https://make.wordpress.org/themes/handbook/review/recommended/ (Zugriff vom 25.11.2015).

WordPress: Theme Review Handbook › Required. https://make.wordpress.org/themes/handbook/review/required/ (Zugriff vom 25.11.2015).

WordPress: Theme Review Handbook › Required › Code. https://make.wordpress.org/themes/handbook/review/required/#code (Zugriff vom 25.11.2015).

WordPress: Theme Review Handbook › Required › Core Functionality and Features. https://make.wordpress.org/themes/handbook/review/required/#core-functionality-and-features (Zugriff vom 25.11.2015).

WordPress: Theme Review Handbook › Required › Documentation. https://make.wordpress.org/themes/handbook/review/required/#documentation (Zugriff vom 25. November).

WordPress: Theme Review Handbook › Required › Language. https://make.wordpress.org/themes/handbook/review/required/#language (Zugriff vom 25.11.2015).

WordPress: Theme Review Handbook › Required › Licensing. https://make.wordpress.org/themes/handbook/review/required/#licensing (Zugriff vom 25.11.2015).

WordPress: Theme Review Handbook › Required › Naming. https://make.wordpress.org/themes/handbook/review/required/#naming (Zugriff vom 25.11.2015).

WordPress: Theme Review Handbook › Required › Options and Settings. https://make.wordpress.org/themes/handbook/review/required/#options-and-settings (Zugriff vom 25.11.2015).

WordPress: Theme Review Handbook › Required › Plugins. https://make.wordpress.org/themes/handbook/review/required/#plugins (Zugriff vom 25.11.2015).

WordPress: Theme Review Handbook › Required › Presentation vs Functionality. https://make.wordpress.org/themes/handbook/review/required/#presentation-vs-functionality (Zugriff vom 25.11.2015).

WordPress: Theme Review Handbook › Required › Screenshot. https://make.wordpress.org/themes/handbook/review/required/#screenshot (Zugriff vom 25.11.2015).

WordPress: Theme Review Handbook › Required › Security and Privacy. https://make.wordpress.org/themes/handbook/review/required/#security-and-privacy (Zugriff vom 26.11.2015).

WordPress: Theme Review Handbook › Required › Selling, credits and links. https://make.wordpress.org/themes/handbook/review/required/#selling-credits-and-links (Zugriff vom 26.11.2015).

WordPress: Theme Review Handbook › Required › Templates. https://make.wordpress.org/themes/handbook/review/required/#templates (Zugriff vom 26.11.2015).

WordPress: Theme Review Handbook › Selling, Themes and Links. https://make.wordpress.org/themes/handbook/review/recommended/selling-themes-and-links/ (Zugriff vom 26.11.2015).

WordPress: Theme Review Handbook › Templates. https://make.wordpress.org/themes/handbook/review/recommended/templates/ (Zugriff vom 26.11.2015).

WordPress: Theme-Verzeichnis. https://de.wordpress.org/themes/ (Zugriff vom 04.01.2016).

WordPress; Boren, Ryan; Westwood, Peter; Koopersmith, Daryl; Cave, Jon; Yoshitaka Erlewine, Michael: „Debug Bar“-Plugin. https://wordpress.org/plugins/debug-bar/ (Zugriff vom 26.11.2015).

World Press Photo Foundation: 2015 Photo Contest. http://www.worldpressphoto.org/collection/photo/2015 (Zugriff vom 30.10.2015).

xrds; Shepherd, Eric; thehaverchuck; hasAngel; Dursun, Resul; Hobson, Stephanie; Rebert, Chris; Taylor, Mike; wesj; zigomir; martind1; ddsgdfgidshvbeivbosdlvbdfvdsv; Dutton, Sam; Brubeck, Matt; Vukicevic, Vlad: Using the viewport meta tag to control

layout on mobile browsers. https://developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport_meta_tag (Zugriff vom 27.11.2015).

Anhang A:
Preisträger-Websites des Photo Contest von der World Press Photo Foundation

Datum der Untersuchung war der 28. Oktober 2015

Website	Besondere Funktionen/Merkmale
jeromesessiniukraine.com	- Übersicht von Referenzen auf Startseite - Übersicht der drei Reportagen auf Startseite - Bilder auf Detailseite als Thumbnails lassen sich in Lightbox öffnen
kierandoherty.com	- Slideshow auf Startseite und Detailseiten - Gibt die Möglichkeit, auf Thumbnails umzuschalten
sofiavaliente.com	- Bilder einer Reportage nebeneinander in einem scrollbaren Bereich
raphaelarosella.com	- Bilder auf Startseite und Detail-Ansicht in Slideshow - Wieder die Möglichkeit, auf Thumbs umzuschalten
varmaphoto.com	- Vollbild-Slider auf Startseite - Bilder einer Galerie einfach in groß untereinander. Beschreibung in Lightbox nach Klick
darcypadilla.com	- Fast alle Infos direkt auf der Startseite - Verlinkt mehr auf Seiten, als direkt ihre Arbeiten auf der eigenen Seite zu zeigen
petemullerphotography.com	- Slideshow auf der Startseite - In Detailansicht große, bildschirmfüllende Bilder in einem scrollbaren Bereich - Andere Variante Detail: große Bilder mit kleinen Thumbnails darunter – Slider
www.ilnitsky-photography.com	- Großes Titelbild auf der Startseite - Wird zufällig angezeigt - Übersicht aller Arbeiten in einer Portfolio-Ansicht - Archiv-Ansicht - Auf Einzelseite eines Portfolio-Elements ein Slider
michelepalazziphotographer.com	- Slider auf der Startseite - Portfolio-Übersicht o Im Menü lassen sich die einzelnen Kategorien auswählen - In Einzelansicht ein Slider - WordPress-Theme *Evolution* in Nutzung: http://www.elegantthemes.com/gallery/evolution/
madsnissen.com	- Viewportfüllender Slider auf Startseite - Auf Einzelansicht Text und Bild-Grid – Lightbox zeigt große Versionen - Gibt die Möglichkeit, mehrere Galerien auf einer Seite anzuzeigen (nach Kategorie?) http://www.madsnissen.com/category/stories/ - WordPress-Theme *Widescreen*: https://graphpaperpress.com/themes/widescreen/
giuliodisturco.com	- Slideshow auf Startseite - Auf Detailseite entweder große Bilder im scrollbaren Bereich oder in einem Slider
massimosestini.it	- Einfach alle Bilder einer Kategorie auf einer Seite, vergrö-

	ßerbar in einer Lightbox
gianfrancotripodo.com	- Auf Startseite eine Übersicht der Kategorien. Jeweils ein Bild für eine Kategorie - Auf Einzelansicht der Kategorien ein Bild pro Bilderstrecke - Auf Einzelansicht einer Bilderstrecke eine Slideshow
sergeyponomarev.com	- Slideshow mit Infos zum jeweiligen Bild auf der Startseite - Auf Einzelansicht ein Slider - Text zu der jeweiligen Serie wird mit in der Sidebar angezeigt
glennagordon.com	- Großes Bild auf der Startseite - Auf der Einzelansicht große Bilder in scrollbarem Bereich
kacperkowalski.pl	- Slider auf Startseite - Auf Galerie-Seite sieht man jeweils in einer Art Slider die Vorschau auf eine Galerie. Da kann man draufklicken und dann die Galerie anschauen.
sandrahoyn.de	- Slider auf der Startseite - Übersicht der Galerien auf einer Seite: o Vorschaubild, Titel, Beschreibung o Zeigt immer nur Galerien einer Kategorie an
christianziegler.photography	- Slider auf der Startseite und auf den Detailseiten
paolomarchetti.org	- Großes Bild auf der Startseite - Übersicht der *Stories* mit Titel und Datum auf einer Seite - Auf der Einzelansicht Text mit Thumb-Grid.
lisakrantz.com	- Video auf der Startseite - Große Bilder im Slider auf Einzelansicht
paoloverzone.com	- Startseite zeigt gleich die erste Übersicht einer Kategorie - Auf Kategorie-Seite werden die Bilder in klein dargestellt, können durch Klick vergrößert werden
markmetcalfe.com.au	- Bilder werden auf jeder Seite groß dargestellt, in einem scrollbaren Bereich - Als System wird *Format* genutzt: http://format.com/l/your_new_portfolio#horizon-left
albello.com	- Slideshow auf der Startseite - Ebenso auf allen anderen Seiten
asasjostrom.com	- Gleich auf der Startseite Grid der Arbeiten - Einzelansicht zeigt Slider mit Text, darunter immer weiter das Grid von der Startseite
malinfezehai.net	- Vollflächiges Hintergrundbild auf Startseite - Slider auf der eigentlichen Home-Seite - Die übrigen Einzelansichten öffnen sich mit einem großen Bild. Man kann aber auf die Thumbnails umschalten
sarkerprotick.com	- Einzelnes Bild auf der Startseite - Auf den Einzelseiten jeweils große Bilder im scrollbaren Bereich - COLr-WordPress-Theme von http://minimalistwp.com/
turicalafato.com	- Slideshow auf der Startseite - Portfolio-Ansicht zeigt die verschiedenen Galerien in einer Übersicht - In Einzelansicht eine Slideshow mit Auswahl-Thumbs
tomasvh.com	- Slider auf der Startseite - Unter *Works* die Vorschau zu drei Serien, und ein Link zum *Archiv* - Auf Einzelseite Slider

Anhang B:
Untersuchung der kostenlosen WordPress-Themes

Untersuchung der 126 kostenlosen WordPress-Themes mit dem Tag *Fotoblog* von WordPress.org. Untersuchungszeitraum ist der 3. und 4. November 2015

Die Funktionen, auf die untersucht wird:

- Trennung von Portfolio-Elementen und Blogbeiträgen
- Galerie als Slider oder einzelne Bilder anzeigen
- Slider oder zufälliges Bild auf der Startseite
- Übersicht aller Arbeiten auf einer Seite und Archiv
- Kategorie-Seiten für Arbeiten
- Verschiedene Darstellungsmöglichkeiten für Portfolio und Kategorie-Übersicht

Da die Funktion, Bilder einer Galerie als einzelne Bilder anzuzeigen, eine Standard-Funktion von WordPress ist, wird diese Funktion hier nicht weiter untersucht

de.wordpress.org/themes/omega/	- Keine der besonderen Funktionen - Mehr ein Theme-Framework, das als Ausgangslage für ein eigenes Theme gedacht ist
de.wordpress.org/themes/x2/	- Bietet die Möglichkeit, einen Slider einzubinden
de.wordpress.org/themes/make/	- Bietet die Möglichkeit, eine Galerie als Slider darzustellen
de.wordpress.org/themes/sans-serif/	- Keine der besonderen Funktionen
de.wordpress.org/themes/veal/	- Slider-Möglichkeit für Startseite
de.wordpress.org/themes/cubic/	- Keine der besonderen Funktionen - Vorschau von Galerien auf der Homepage sieht gut aus
de.wordpress.org/themes/tari/	- Keine der besonderen Funktionen
de.wordpress.org/themes/treeson/	- Keine der besonderen Funktionen
de.wordpress.org/themes/precious-lite/	- Gibt Feature, dass man Galerie mit Shortcode einbinden und dann die Bilder filtern kann - Keine der Funktionen, die aus der Fotografen-Analyse hervorgingen
de.wordpress.org/themes/pinboard/	- Möglichkeit, Portfolio-Elemente vom Blog auszuschließen - Möglichkeit, alle Beiträge aus dieser Kategorie auf einer Seite anzuzeigen
de.wordpress.org/themes/origami-evergreen/	- Bestimmte Elemente können auf einer Seite angezeigt werden
de.wordpress.org/themes/nova-lite/	- Keine der Funktionen
de.wordpress.org/themes/realm/	- Keine der Funktionen
de.wordpress.org/themes/white/	- Möglichkeit, Slider einzufügen

de.wordpress.org/themes/foliogine-lite/	- Möglichkeit, Slider einzubauen
de.wordpress.org/themes/skt-white/	- Slider-Möglichkeit
de.wordpress.org/themes/yuuta/	- Keine der Funktionen, aber Design gefällt
de.wordpress.org/themes/alum/	- Slider auf Startseite
de.wordpress.org/themes/snapshot/	- Slider
de.wordpress.org/themes/virtue/	- Können Portfolio-Elemente angelegt werden - Slider auf den Einzel-Seiten - Nutzt für die Portfolio-Funktion aber ein Plugin
de.wordpress.org/themes/family/	- Keine der Funktionen
de.wordpress.org/themes/diarjo-lite/	- In der Free-Version keine der Funktionen
de.wordpress.org/themes/skt-parallaxme/	- Slider
de.wordpress.org/themes/paraxe/	- Keine der Funktionen
de.wordpress.org/themes/boardwalk/	- Keine der Funktionen
de.wordpress.org/themes/rara-clean/	- Beitrags-Slider
de.wordpress.org/themes/lens/	- Keine der Funktionen
de.wordpress.org/themes/semper-fi-lite/	- Keine der Funktionen
de.wordpress.org/themes/marla/	- Keine der Funktionen
de.wordpress.org/themes/kwikload/	- Keine der Funktionen
de.wordpress.org/themes/editor/	- Keine der Funktionen
de.wordpress.org/themes/verbo/	- Keine der Funktionen
de.wordpress.org/themes/inovate/	- Slider-Möglichkeit
de.wordpress.org/themes/skt-black/	- Slider-Möglichkeit
de.wordpress.org/themes/coherent/	- Keine der Funktionen
de.wordpress.org/themes/sylvia/	- Keine der Funktionen
de.wordpress.org/themes/willingness/	- Keine der Funktionen
de.wordpress.org/themes/bushwick/	- Keine der Funktionen
de.wordpress.org/themes/photostory/	- Keine der Funktionen
de.wordpress.org/themes/verge/	- Keine der Funktionen
de.wordpress.org/themes/nerocity/	- Keine der Funktionen
de.wordpress.org/themes/seiryuu/	- Keine der Funktionen
de.wordpress.org/themes/sg-window/	- Keine der Möglichkeiten
de.wordpress.org/themes/snaps/	- Keine der Funktionen
de.wordpress.org/themes/dazzling/	- Beitrags-Slider
de.wordpress.org/themes/adventurous/	- Featured-Content-Slider
de.wordpress.org/themes/seller/	- Featured-Content-Slider
de.wordpress.org/themes/writ/	- Keine der Funktionen
de.wordpress.org/themes/jehanne/	- Content-Slider
de.wordpress.org/themes/quest/	- Slider
de.wordpress.org/themes/tography-lite/	- Portfolio-Elemente lassen sich auf einer Seite anzeigen - Können auch gefiltert werden
de.wordpress.org/themes/quickpic/	- Keine der Funktionen

de.wordpress.org/themes/sidekick/	- Keine der Funktionen
de.wordpress.org/themes/dms/	- Portfolio-Übersicht möglich - Kann auch gefiltert werden
de.wordpress.org/themes/kelly/	- Keine der Funktionen
de.wordpress.org/themes/fukasawa/	- Keine der Funktionen
de.wordpress.org/themes/ariwoo/	- Keine der Funktionen
de.wordpress.org/themes/panaroma/	- Slider
de.wordpress.org/themes/patio/	- Keine der Funktionen
de.wordpress.org/themes/aladdin/	- Keine der Funktionen
de.wordpress.org/themes/birdsite/	- Keine der Funktionen
de.wordpress.org/themes/aspen/	- Keine der Funktionen
de.wordpress.org/themes/landline/	- Keine der Funktionen
de.wordpress.org/themes/invisible-assassin/	- Slider
de.wordpress.org/themes/bose/	- Content-Slider
de.wordpress.org/themes/pinnacle/	- Portfolio-Übersicht in zwei verschiedenen Varianten - Auf einzelnem Objekt ist Slider möglich - Portfolio-Funktion kann nur mit einem Plugin erreicht werden
de.wordpress.org/themes/protopress/	- Keine der Funktionen
de.wordpress.org/themes/sg-grid/	- Portfolio-Übersicht - Lässt sich filtern
de.wordpress.org/themes/the-j-a-mortram/	- Keine der Funktionen
de.wordpress.org/themes/pagelines/	- Keine der Funktionen
de.wordpress.org/themes/aron/	- Slider
de.wordpress.org/themes/alhena-lite/	- Content-Slider - Bilder können als Projekte angelegt und gefiltert werden
de.wordpress.org/themes/sparkling/	- Content-Slider
de.wordpress.org/themes/professional/	- Slider
de.wordpress.org/themes/i-max/	- Content-Slider
de.wordpress.org/themes/gridsby/	- Fotos lassen sich auf einer Seite anzeigen, interessante Ansicht beim Vergrößern (mit Beschreibung)
de.wordpress.org/themes/skt-photo-session/	- Slider
de.wordpress.org/themes/sg-double/	- Keine der Funktionen
de.wordpress.org/themes/horas/	- Slider
de.wordpress.org/themes/weaver-ii/	- Keine der Funktionen
de.wordpress.org/themes/xclusive/	- Keine der Funktionen
de.wordpress.org/themes/aldehyde/	- Content-Slider
de.wordpress.org/themes/follet/	- Keine der Funktionen
de.wordpress.org/themes/swell-lite/	- Lassen sich Portfolio-Elemente festlegen
de.wordpress.org/themes/skt-photo-world/	- Slider
de.wordpress.org/themes/museum/	- Keine der Funktionen
de.wordpress.org/themes/freak/	- Slider

de.wordpress.org/themes/dream-way/	- Portfolio-Übersicht - Lässt sich auch filtern
de.wordpress.org/themes/aperture/	- Slider
de.wordpress.org/themes/colorsnap/	- Keine der Funktionen
de.wordpress.org/themes/sliding-door/	- Keine der Funktionen
de.wordpress.org/themes/market/	- Slider
de.wordpress.org/themes/fifteen/	- Keine der Funktionen
de.wordpress.org/themes/harmonic/	- Keine der Funktionen
de.wordpress.org/themes/biancaa/	- Content-Slider
de.wordpress.org/themes/divina/	- Keine der Funktionen
de.wordpress.org/themes/emphaino/	- Keine der Funktionen
de.wordpress.org/themes/jolene/	- Keine der Funktionen
de.wordpress.org/themes/hoffman/	- Keine der Funktionen
de.wordpress.org/themes/phogra/	- Slider - Portfolio-Übersicht - Slider auf Einzelseite
de.wordpress.org/themes/biker/	- Keine der Funktionen
de.wordpress.org/themes/stained-glass/	- Keine der Funktionen
de.wordpress.org/themes/jax-lite/	- Portfolio-Übersicht - Kann gefiltert werden
de.wordpress.org/themes/undiscovered/	- Keine der Funktionen
de.wordpress.org/themes/satu/	- Keine der Funktionen
de.wordpress.org/themes/water-lily/	- Keine der Funktionen
de.wordpress.org/themes/times/	- Keine der Funktionen
de.wordpress.org/themes/weaver-xtreme/	- Slider
de.wordpress.org/themes/great/	- Content-Slider
de.wordpress.org/themes/times-square/	- Keine der Funktionen
de.wordpress.org/themes/eryn/	- Keine der Funktionen
de.wordpress.org/themes/visual/	- Keine der Funktionen
de.wordpress.org/themes/suevafree/	- Galerie-Bilder als Slider
de.wordpress.org/themes/catch-kathmandu/	- Content-Slider
de.wordpress.org/themes/adaption/	- Keine der Funktionen
de.wordpress.org/themes/skt-full-width/	- Slider - Portfolio nur in Pro-Version
de.wordpress.org/themes/careta/	- Keine der Funktionen
de.wordpress.org/themes/enlightenment/	- Content-Slider - Portfolio-Übersicht (mit Plugin)
de.wordpress.org/themes/photolab/	- Keine der Funktionen
de.wordpress.org/themes/avnii/	- Keine der Funktionen
de.wordpress.org/themes/blaskan/	- Keine der Funktionen
de.wordpress.org/themes/azeria/	- Content-Slider - Portfolio-Übersicht, aber mit Plugin
de.wordpress.org/themes/sg-simple/	- Keine der Funktionen
de.wordpress.org/themes/fluxipress/	- Keine der Funktionen

de.wordpress.org/themes/bouquet/	- Keine der Funktionen

Anhang C:
Untersuchung der kostenpflichtigen Themes

Untersuchung von 10 kostenpflichtigen Premium-Themes von ThemeForest. Untersuchungszeitraum ist der 9. November 2015

Die Funktionen, auf die untersucht wird:

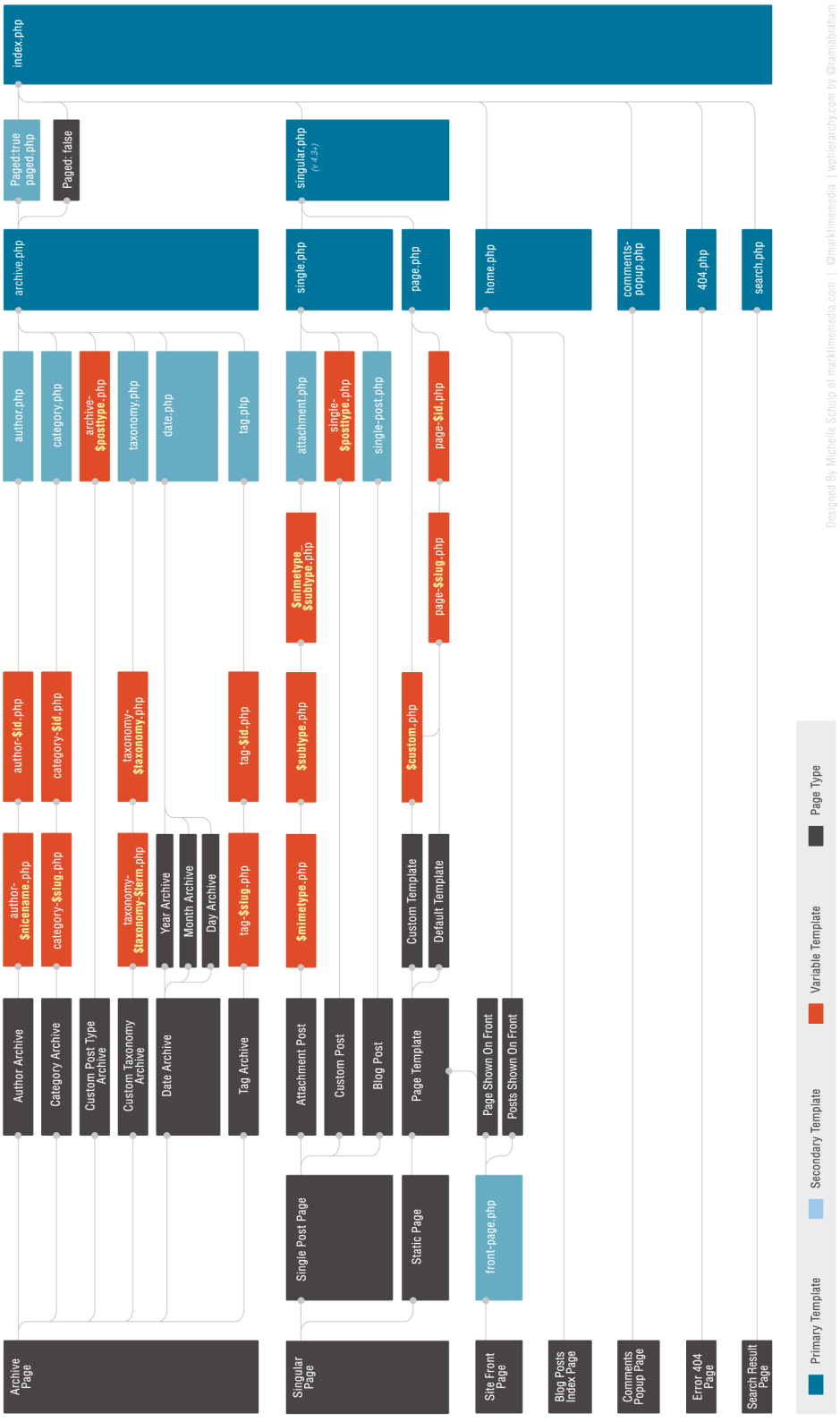
- Trennung von Portfolio-Elementen und Blogbeiträgen
- Galerie als Slider oder einzelne Bilder anzeigen
- Slider oder zufälliges Bild auf der Startseite
- Übersicht aller Arbeiten auf einer Seite und Archiv
- Kategorie-Seiten für Arbeiten
- Verschiedene Darstellungsmöglichkeiten für Portfolio und Kategorie-Übersicht

Da die Funktion, Bilder einer Galerie als einzelne Bilder anzuzeigen, eine Standard-Funktion von WordPress ist, wird diese Funktion hier nicht weiter untersucht

themeforest.net/item/core-minimalist-photography-portfolio/240185	- Bietet verschiedene Möglichkeiten, eine Galerie darzustellen (mit Shortcodes) - Kann eine Art Slider auf der Startseite darstellen - Arbeitet viel mit Shortcodes (Buttons, Lightbox-Funktion, Slider …)
themeforest.net/item/invictus-a-fullscreen-photography-wordpress-theme/180096	- Möglichkeit, einen Vollbild-Slider für die Startseite zu nutzen – optional mit Video-Hintergrund - Verschiedene Templates für die Darstellung einer Portfolio-Übersicht - Portfolio-Elemente scheinen mit Custom-Post-Types angelegt zu werden - Einige Shortcodes
themeforest.net/item/photolux-photography-portfolio-wordpress-theme/894193	- Portfolio-Elemente lassen sich auf einer Seite darstellen - Können gefiltert werden - Kann Slider eingebaut werden - Portfolio-Elemente sind keine Beiträge
themeforest.net/item/fluxus-portfolio-theme-for-photographers/3854385	- Homepage-Slider - verschiedene Portfolio-Ansichten, die sich auch filtern lassen - Portfolio-Elemente keine Beiträge - Setzt auf Shortcodes für Accordion, Meldungen, Service-Ansicht und mehr
themeforest.net/item/expression-photography-responsive-wordpress-theme/2855595	- Slider - Filterbare Portfolio-Ansicht, nur eine Variante

	- Shortcodes für Tabs
themeforest.net/item/dk-for-photography-creative-portfolio/631383	- Slider auf Homepage - Auch einzelnes Bild möglich - Keine Portfolio-Ansicht
themeforest.net/item/skylab-portfolio-photography-wordpress-theme/4740718	- Slider auf Homepage - Verschiedene Portfolio-Layouts - Portfolio-Elemente sind keine Beiträge - Möglichkeit, Einzelansicht eines Portfolio-Elements als einzelne Bilder oder in einem Slider anzuzeigen
themeforest.net/item/border-a-delightful-photography-wordpress-theme/6920241	- Homepage-Slider - Portfolio-Ansicht in verschiedenen Varianten - Setzt auf Shortcodes
themeforest.net/item/epix-fullscreen-photography-wordpress-theme/5783556	- Slider auf Homepage - Portfolio-Ansicht, lässt sich auch filtern - Portfolio-Items keine Beiträge - Verschiedene Möglichkeiten, Bilder anzuzeigen
themeforest.net/item/titan-responsive-portfolio-photography-theme/5072056	- Homepage-Slider - Verschiedene Möglichkeiten, Galerien darzustellen - Portfolio-Funktion mit verschiedenen Varianten - Arbeitet auch mit Shortcodes

Anhang D:
Template-Hierarchie in WordPress-Themes[279]



[279] vgl. Schulp

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die eingereichte Bachelorarbeit selbständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Hannover, den 22. Januar 2016