HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

Sommersemester 2020

Projektarbeit:

Performanceverbesserung mittels des statischen Seitengenerators GatsbyJS anhand eines Fallbeispiels

Vorgelegt von:	Oliver Hagel und Simon Schwegler
Matrikelnummer:	30490 und 30306
Telefon:	015222002926
Betreuer:	Prof. Dr. rer. nat. Marius Hofmeisterx

Inhalt

Abbildungsverzeichnis

1. Einführung

1.1 Einleitung

Stand 2019 gibt es über 1.5 Milliarden Webseiten.[1] Bei einer so hohen Zahl scheint es nahezu unmöglich, dass nach einer Google Suchanfrage eines potenziellen Besuchers unserer Webseite ihm eben diese als eine der ersten in der Google Ergebnisliste vorgeschlagen wird. Umso bedauernswerter ist es, wenn wir Besucher durch Szenerien wie folgendes verlieren: Ein User möchte unsere Webseite besuchen, doch nach einem Klick auf den Link blickt er auf einen leeren Bildschirm – die Seite lädt. Nach kurzem Warten klickt er den Zurück Button und verlässt genervt die Seite.

Für Seitenbetreiber, denen das beschriebene Szenario keinesfalls fremd ist und die aufgrund schlechter Performance immer wieder Besucher verlieren oder die eigene Webseite gar nicht erst bei Google Suchanfragen gelistet wird, könnte das auf React[2] basierende open source Framework GatsbyJS Abhilfe schaffen. Nach eigener Aussage der Gatsby Entwickler hilft das Framework „blitzschnelle Websites und Apps“ zu erstellen.[3]

Das Ziel dieser Arbeit ist es, anhand eines Fallbeispiels, mit GatsbyJS, statische Seiten zu generieren und auf deren Grundlage ein Performance Vergleich vorzunehmen wobei das Versprechen der Gatsby Betreiber auf den Prüfstand gestellt wird. Im Rahmen der Arbeit sollen auch die Einsatzmöglichkeiten und Grenzen des Frameworks aufgezeigt werden.

[1] Vgl. Fowler, S. Daniel (2014), How many websites are there in the world?, https://tekeye.uk/computing/how-many-websites-are-there, Stand 13.05.2020
[2] React ist eine JavaScript-Softwarebibliothek, die ein Grundgerüst für die Ausgabe von User-Interface-Komponenten von Webseiten zur Verfügung stellt
[3] Vgl. https://www.gatsbyjs.org, Stand 13.05.2020

1.2 Statische und dynamisch Webseiten

Ein grundlegender Unterschied der Eigenschaften und Einsatzmöglichkeiten einer Website hängt davon ab, ob diese statisch oder dynamisch ist. In diesem Unterkapitel werden die zwei Arten erklärt und deren Vorteile kurz aufgezeigt.

Statische Webseiten

Charakteristisch für eine statische Webseite ist, dass die Webseite und jede ihrer Unterseiten als separate Dokumente entwickelt werden und anschließend auf einem sogenannten Webserver abgelegt werden. Ein Webserver ist eine Software, die von Webbrowsern angeforderte Webseiten bereitstellt.

Alle Inhalte, wie Texte, Bilder oder Videos, sind "fest" in den HTML Code der Seite eingebunden und werden so wie sie erstellt wurden aufgerufen. Beim Besuch auf der Webseite durch einen Benutzer wird dann auf das angeforderte Dokument und den Inhalt eins-zu-eins zugegriffen. Der Inhalt ändert sich also nicht mehr wenn diese angefordert wird, wie es sonst bei dynamischen Seiten der Fall ist.

Vereinfacht dargestellt läuft es bei einem Zugriff auf eine statische Webseite durch einen Besucher dann wie folgt ab: Der Webbrowser fordert eine statische Seite über eine http-Request an. Der Webserver sucht die Seite heraus und sendet diese dann anschließend an den Browser. *(siehe Abbildung 1)*[4]

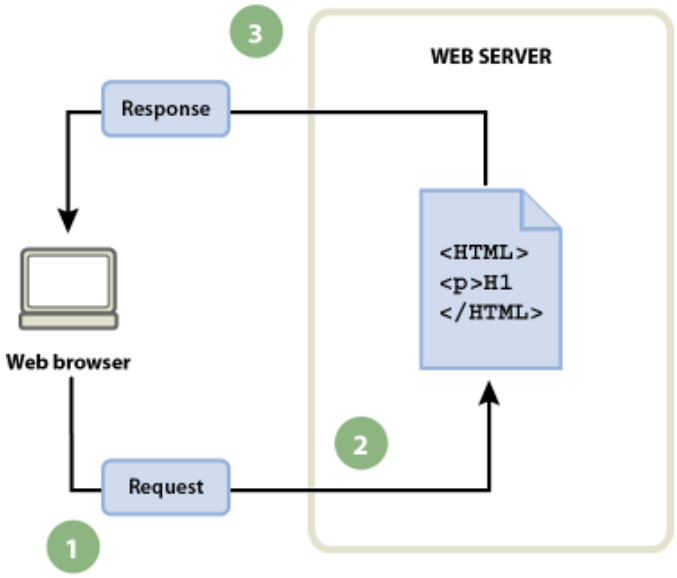

Abbildung 1: Verarbeitung einer statischen Webseite[5]

[4] Vgl. Adobe Dreamwaver Benutzerhandbuch (2017), https://helpx.adobe.com/de/dreamweaver/using/web-applications.html, Stand 16.05.2020
[5] Vgl. Adobe Dreamwaver Benutzerhandbuch (2017), https://helpx.adobe.com/de/dreamweaver/using/web-applications.html, Stand 16.05.2020

Statische Webseiten bieten Vorteile wie geringe laufende Kosten, da die Server keine speziellen technischen Anforderungen benötigen und es wird keine Datenbank benötigt. Doch allem voran steht der Vorteil, dass durch den direkten Zugriff auf die bereits „fertig“ auf dem Server liegende Webseite eine hohe Zugriffsgeschwindigkeit erreicht wird. [6]

Dynamische Webseiten

Dynamische Webseiten basieren darauf, wie der Name bereits vermuten lässt, dass der zu anzeigende Inhalt erst im Augenblick des Seitenaufrufes entsteht. Wird eine dynamische Seite angefordert, wird die Seite an einen Anwendungsserver weitergeleitet, der sich um die Fertigstellung der Webseite kümmert. Diese Software liest den Code auf der Seite ein, stellt gemäß diesen Anweisungen eine Seite zusammen und entfernt dann den Code aus der Seite. Die daraus entstandene statische Seite wird vom Anwendungsserver an den Webserver übergeben und gelangt so zum Browser.

Zusammengefasst kann man sich die Verarbeitung einer dynamischen Webseite also wie folgt vorstellen:

Der Webbrowser fordert eine dynamische Seite über einen Request an. Der Webserver sucht die Seite und übergibt sie an den Anwendungsserver, der die Anweisungen analysiert und die Seite vervollständigt. Der Anwendungsserver gibt die fertiggestellte Seite an den Webserver zurück und sendet die fertiggestellte Seite an den Browser. *(siehe Abbildung 2)*[7]

[6] Vgl. Graack Christoph (2011), https://blog.kompaktdesign.com/webdesign/statisch-vs-dynamisch, Stand 10.05.2020
[7] Vgl. Adobe Dreamwaver Benutzerhandbuch (2017), https://helpx.adobe.com/de/dreamweaver/using/web-applications.html, Stand 16.05.2020

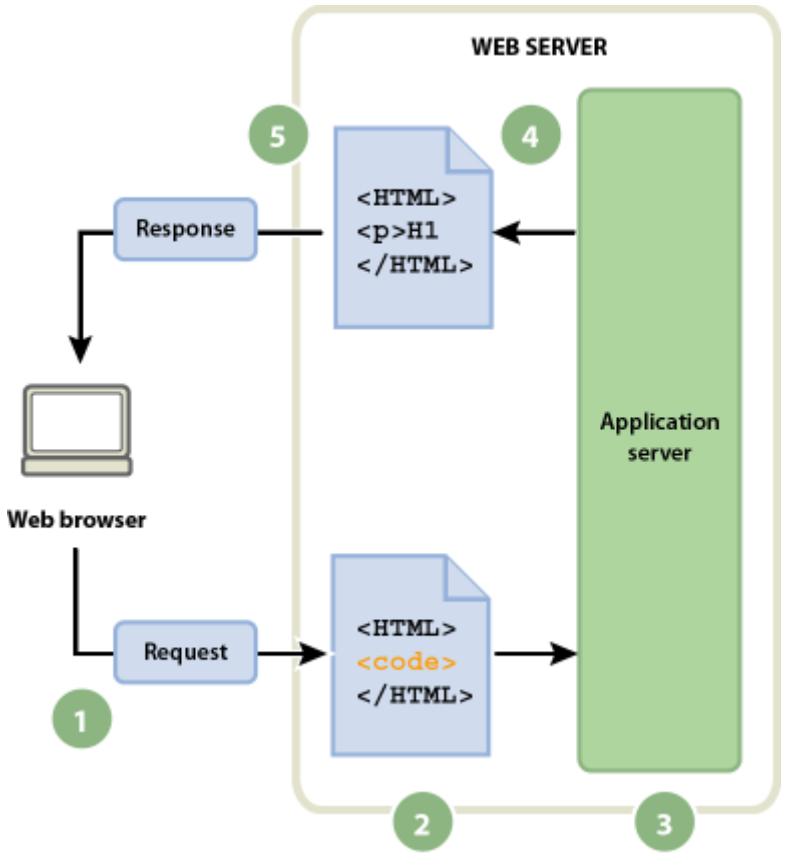

Abbildung 2: Verarbeitung einer dynamischen Webseite[8]

In der Praxis wird der Anwendungsserver mit serverseitigen Ressourcen, wie z.B. Datenbanken kombiniert. So wird der Anwendungsserver beispielsweise angewiesen, Daten aus einer Datenbank zu extrahieren und in den HTML-Code der Seite einzufügen. In der Datenbank werden die Inhalte, die auf der Webseite angezeigt werden sollen, eingespeichert. Somit müssen nicht mehr einzelne HTML-Dateien für jede Seite geschrieben werden, sondern es wird eine Vorlage erstellt, die dann für alle Informationsarten die präsentiert werden sollen genutzt wird. Die Befehle, Daten aus einer Datenbank zu extrahieren (Datenbankabfrage), bestehen aus Anweisungen die in einer Datenbanksprache wie SQL oder GraphQL ausgedrückt werden. Die Query wird in die serverseitigen Skriptsprachen wie PHP, Perl oder Python in die Tags der Seite geschrieben. Die aus der Datenbank bezogenen Daten werden anschließend in einer Datensatzgruppe zusammengefasst, die dann später vom Anwendungsserver in die Seite eingefügt werden kann. Da Anwendungsserver nicht direkt mit Datenbanken kommunizieren können, ist die Kommunikation nur über entsprechende Datenbanktreiber möglich. Ein Datenbanktreiber ist eine Software mit der Funktion eines

[8] Vgl. Adobe Dreamwaver Benutzerhandbuch (2017), https://helpx.adobe.com/de/dreamweaver/using/web-applications.html, Stand 16.05.2020

Vermittlers zwischen dem Anwendungsserver und der Datenbank. Sofern auf dem Server die entsprechenden Datenbanktreiber installiert sind, kann so gut wie jede Datenbank verwendet werden.

Man kann sich die Generierung einer dynamischen Webseite samt Zugriff auf die Datenbank vereinfacht dargestellt folgendermaßen vorstellen:

Der Webbrowser fordert eine dynamische Seite über ein Request an. Der Webserver sucht die Seite und übergibt sie an den Anwendungsserver, welcher dann die Anweisungen auf der Seite analysiert. Im Anschluss sendet er die Abfrage an den Datenbanktreiber, der wiederrum die eigentliche Datenbankabfrage ausführt. Die Datensatzgruppe wird an den Treiber zurückgegeben und der Treiber übergibt die Datensatzgruppe an den Anwendungsserver. Hier werden die Daten in die Seite eingefügt und die Seite im Anschluss an den Webserver übergeben. Schließlich sendet der Webserver die fertiggestellte Seite an den Browser. (*siehe Abbildung 3)*[9]

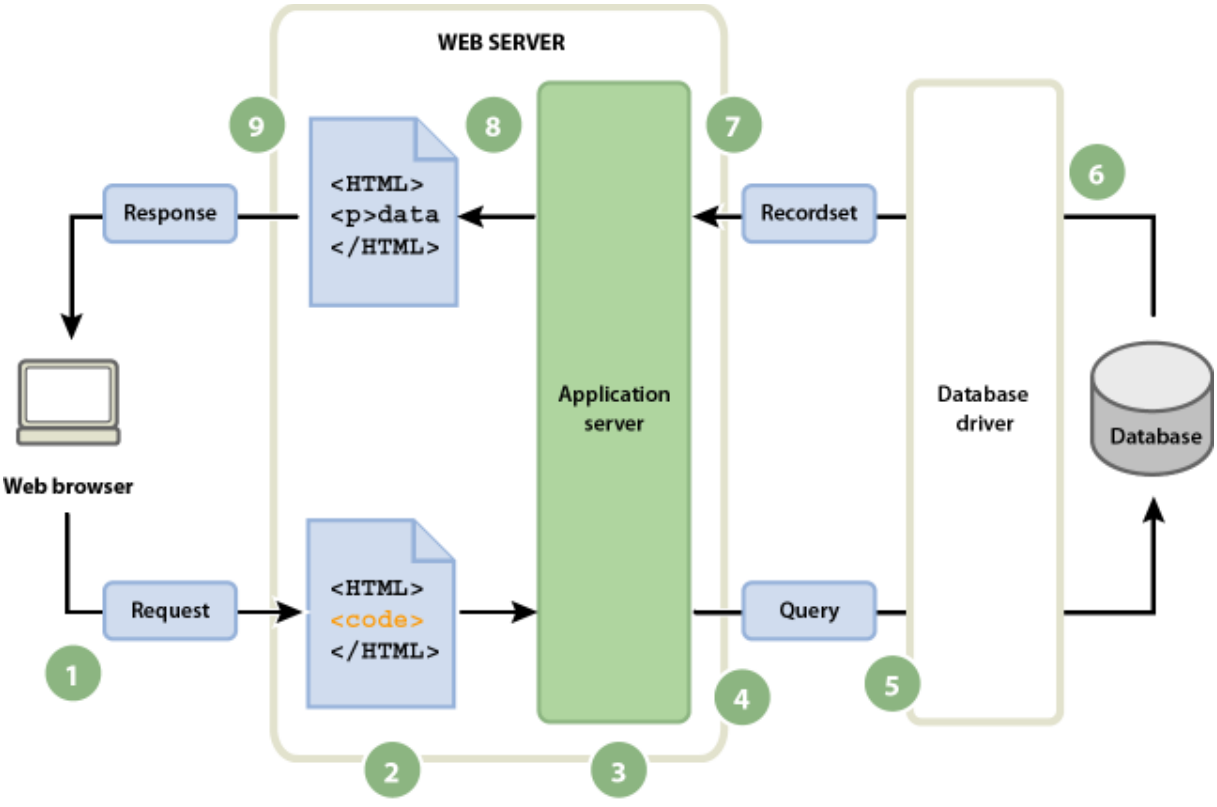

Abbildung 3: Auf Datenbanken zugreifen[10]

[9] Vgl. Adobe Dreamwaver Benutzerhandbuch (2017), https://helpx.adobe.com/de/dreamweaver/using/web-applications.html, Stand 16.05.2020
[10] Vgl. Adobe Dreamwaver Benutzerhandbuch (2017), https://helpx.adobe.com/de/dreamweaver/using/web-applications.html, Stand 16.05.2020

Durch die Eigenschaften der dynamischen Webseiten ergeben sich einige Vorteile:

Der Aufwand zur Pflege dynamischer Webseiten ist im Vergleich zu statischen Webseiten deutlich geringer und einfacher, da Design und Inhalt voneinander getrennt sind. Während die bereitzustellenden Informationen meist in strukturierter Form in einer XML-Datei oder in einer Datenbank vorhanden sind, werden für die Darstellung diese Informationen mit einer HTML-Vorlage gemischt und entsprechende Referenzen gesetzt. Soll jetzt beispielsweise bei einer Webseite das Design verändert werden, müssen im Optimalfall nur die HTML-Vorlagen oder die allgemeinen Design-Vorschriften geändert werden. Alle Seiten, die auf dieser Vorlage basieren spiegeln diese Änderungen wieder.

Durch die Erstellung der Website zur Laufzeit ist gewährleistet das immer der aktuellste Stand der Site dargestellt wird und die just-in-time Erzeugung ermöglicht Interaktivität der Website, d.h. die Website reagiert je nach Eingabe des Benutzers anders. Ein Beispiel hierfür wäre eine Login Mechanismus. Gerade bei Web-Anwendungen wie Online-Shops ist die Aktualität besonders relevant. Sobald es eine Änderung am Rohbestand der Daten gibt, wie Zum Beispiel durch das Tätigen einer Bestellung, wirkt sich diese auf die Seite aus, sodass Besucher immer mit den neusten Informationen versorgt werden können.[11]

1.3 Content Management Systeme

Ein Content Management System, kurz CMS, ist ein System das auf die Bereitstellung und Verwaltung von digitalen Inhalten spezialisiert ist. Meistens kommt es bei Webseiten zur Verwendung, aber auch bei anderen Medienformen wäre eine Verwendung möglich. Im Mittelpunkt steht immer der Inhalt und nicht die Struktur oder die technische Basis. Dabei kann der Inhalt klar von der Struktur, dem Aussehen und der technischen Basis getrennt werden, was grundsätzlich von dem Konzept der konventionellen Webseiten abweicht, bei denen HTML Elemente mit Text oder dem Design oft gemischt sind. Ein CMS ist vom Aufbau mit einem MVC-Pattern zu vergleichen (Model-View-Controller), einem Konzept das eine schnelle Aktualisierung von Inhalten sowie eine flexible Änderung und Anpassung des Designs und Layouts, bis hin zum gesamten Aufbau einer Seite erlaubt. So müssen keine Anpassungen am Inhalt und oft auch nicht an der Struktur vorgenommen werden, da die Ebenen strikt getrennt sind.

In einem klassischen Content Management System wird der erstellte Content über eine Oberfläche in das Backend eingespeist und hier in einer relationalen Datenbank wie MySQL oder Maria DB gespeichert. Bei einer Seitenanfrage wird dann von dort aus der Content

[11] Vgl. Hauser Benjamin (2007): Erstellen von dynamischen Websites: Einsatz von Webservern & Datenbanken unter Windows CE, Seite 5 ff.

geladen. Im Anschluss verknüpft das System den Inhalt mit Themes (dem Design) und stellt die Webseite über die View im Frontend dar.

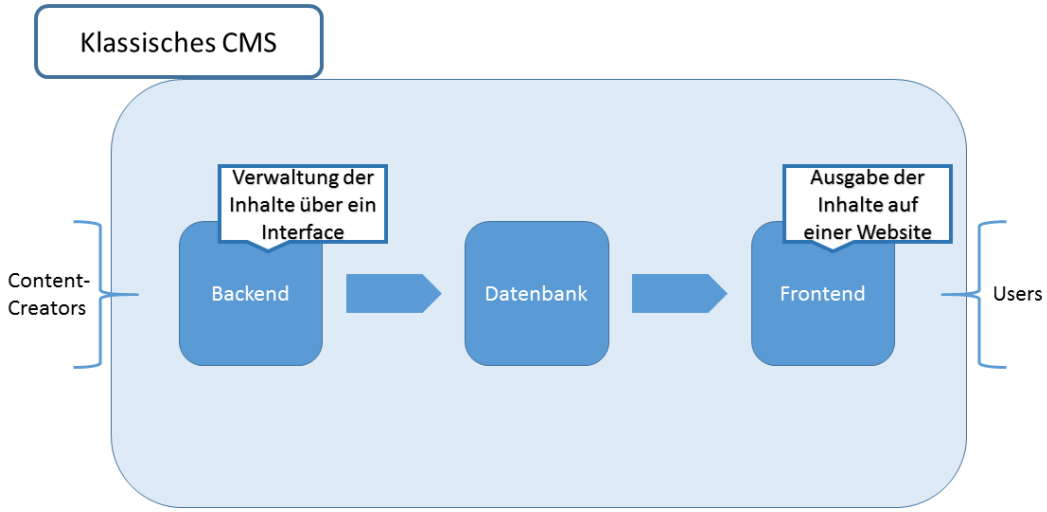

Abbildung 4: Klassisches CMS[12]

Zum Managen des Contents bieten CMS eine Benutzeroberfläche an über die alle grundlegenden Funktionen genutzt werden. Dazu gehört die Nutzerverwaltung, damit sich die Nutzer überhaupt anmelden können, eine Rollenverwaltung, da nicht alle Nutzer die gleiche Aufgabe übernehmen und die Seitenverwaltung, um die einzelnen Seiten der Website anzulegen, zu sortieren und zu verwalten. Zum Erstellen von Inhalt wird ein Editor bereitgestellt, der je nach Anbieter verschiedene Zusatzfunktionen anbietet und nahezu grenzenlose Ausbaustufen hat. Medien wie Bilder, PDFs oder Videos werden in einem Content Management System immer separat gespeichert. Um das CMS nach eigenen Wünschen zu konfigurieren, wird ein Konfigurations-Management integriert, das selbst bei den einfachsten Content Management Systemen ein paar Möglichkeiten zu Grund-Konfigurationen anbietet.[13]

[12] Vgl. https://www.ionos.de/digitalguide/hosting/cms/headless-cms-was-sind-die-vorteile, Stand 23.05.2020
[13] Vgl. Sebastian Schürmanns (2020) https://cmsstash.de/empfehlungen/einfuhrung-cms, Stand 23.05.2020

1.4 Search Engine Optimization

Der Begriff der Search Engine Optimization (dt. Suchmaschinenoptimierung), kurz SEO, umfasst alle Maßnahmen, die genutzt werden, damit die Suchmaschinen des Webs die Site besser finden können. Das oberste Ziel ist immer das gleiche – mit der eigenen Webseite ganz oben zu stehen. Neben der Position (Ranking) ist es auch entscheidend wie hoch die Übereinstimmung zwischen, dem vom Nutzer gewählten Suchbegriff, der Suchergebnisdarstellung (Snippet) und der Relevanz der Landeseite, ist. SEO gilt dann als nachhaltig, wenn der Nutzer seiner Aufgabe effizient auf der Webseite lösen kann. Wichtige Bereiche des SEOs sind zum Beispiel:

Keywording: Hierbei geht es darum die wichtigsten Schlüsselbegriffe (Keywords) zu ermitteln, auf die die Webseite dann optimiert werden soll.

Technisches SEO: Hierunter versteht man alle Maßnahmen, die dafür sorgen, dass der Google-Algorithmus die Inhalte der Seite lesen und verstehen kann.

Wichtige Kennzahlen im Bereich der SEO sind Klickrate (CTR), Absprungraten, Verweildauer auf der Seite, Performance und einige mehr. Mittlerweile gibt es viele Optimierungsmöglichkeiten und Kennzahlen im Bereich der Search Engine Optimization, auf die aber im Rahmen dieser Arbeit nicht alle eingegangen werden soll.[14]

2. Relevanz der Seiten Performance

Um die Relevanz der Performance einer Webseite herauszustellen und darzulegen warum kurze Ladezeiten für jeden Seitenbetreiber erstrebenswert sein sollten, wird im folgenden Kapitel ein kurzer Überblick zu zwei relevanten Zusammenhängen im Bezug zur Seitenladezeit gegeben.

2.1 Zusammenhang von Performance und User Experience

Die Seitengeschwindigkeit ist ein Maß dafür wie schnell der Inhalt einer Webseite geladen wird. Untersuchungen dazu zeigten, dass Seiten mit einer längeren Ladezeit tendenziell höhere Absprungraten vorweisen und Besucher durchschnittlich weniger Zeit auf der Seite verbringen. Jakob Nielsen, ein dänischer Schriftsteller, Redner und Berater im Bereich Software- und Webdesign-Gebrauchstauglichkeit, definierte schon 1933 die drei sogenannten „response time limits“:

[14] Vgl. Erlhofer Sebastian (2011), Suchmaschinen Optimierung das umfassende Handbuch, 9. Auflage, Seite 29 ff.

0,1 Sekunde ist das Limit, bei dem der Besucher das Gefühl hat, dass das System direkt antwortet. Somit ist kein besonderes Feedback nötig, außer das Anzeigen des angefragten Inhalts.

Bei ungefähr 1 Sekunde liegt das Limit, bei dem der Gedankenfluss des Benutzers noch nicht unterbrochen wurde. Bemerken wird er aber die Verzögerung. Normalerweise ist bei Verzögerungen von mehr als 0,1, aber weniger als 1,0 Sekunden keine spezielle Rückmeldung erforderlich, aber der Benutzer verliert das Gefühl unmittelbar mit den Daten arbeiten zu können.

Bei Wartezeiten länger als 10 Sekunden verliert der Benutzer die Aufmerksamkeit und beschäftigt sich mit anderen Dingen bis der PC fertig geladen hat. Daher sollte der Benutzer in Kenntnis gesetzt werden wann mit einem Ergebnis zu rechnen ist. Besonders wichtig ist es bei derartig langen Wartezeiten den Benutzer mit einem Feedback auf dem Laufenden zu halten. [15]

Auch wenn die 10 Sekunden Obergrenze für die Aufmerksamkeitsspanne immer noch gilt ist die Zeit, die der Benutzer bereit ist zu warten über die letzten Jahre deutlich gesunken. In vielen Fällen verlassen die Benutzer die Webseite einfach, wenn sie nicht in wenigen Sekunden geladen hat. Einen ungefähren Zusammenhang der Ladezeiten und der Wahrscheinlichkeit, dass ein User abspringt zeigt folgende Statistik

[15] Vgl. Jakob Nielsen (1993), https://www.nngroup.com/articles/response-times-3-important-limits/, Stand 16.05.2020

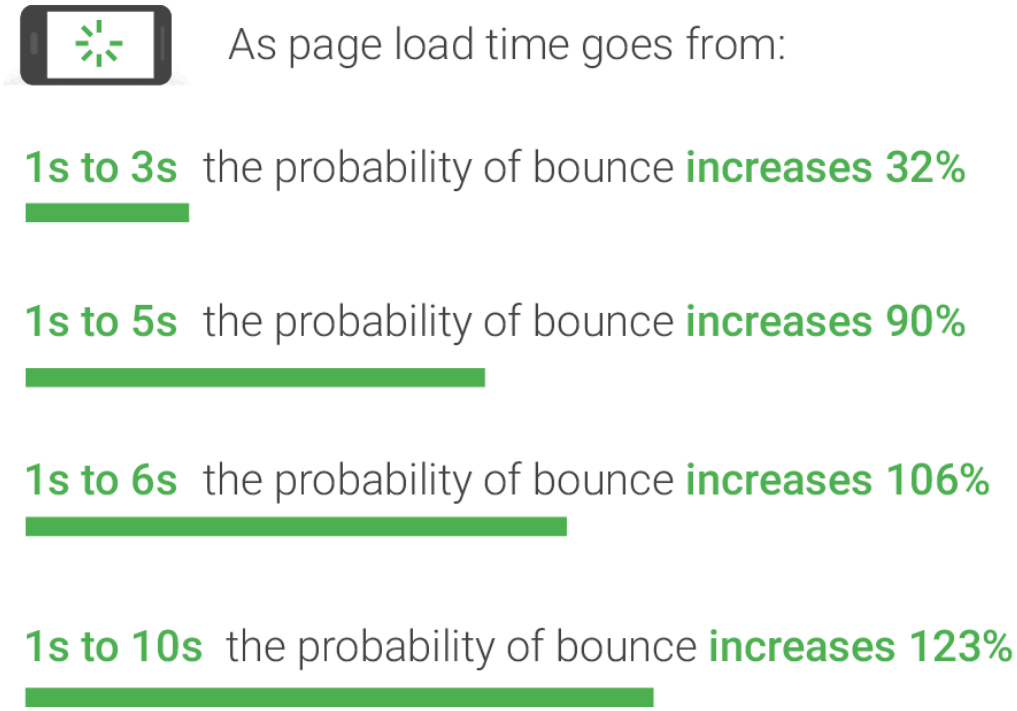

Abbildung 5: Page loading time and the probability of bounce[16]

Wie in Abbildung 5 zu sehen ist, erhöht sich die Wahrscheinlichkeit, dass in den ersten drei Sekunden der User abspringt, wenn die Seite nicht innerhalb dieses Zeitrahmens geladen wird, um 32 Prozent.

Sobald der Benutzer das Gefühl bekommt, er müsse auf den Seiteninhalt zu lange warten wird es seine User Experience maßgeblich beeinflussen. Er wird sich mehr auf das Warten konzentrieren als auf die Erfüllung der Aufgabe, für die er die Seite überhaupt besucht. Infolgedessen ist es weniger wahrscheinlich, dass er eine schwierige Aufgabe erfolgreich erfüllen wird. Vielmehr kommt durch das unerwartete Warten beim Benutzer das Gefühl auf, keine Systemsteuerung zu haben. Dieses Gefühl sollte dem Besucher einer Seite unbedingt erspart bleiben, da es sich maßgeblich negativ auf seine User Experience auswirkt.

Abschließend lässt sich also zusammenfassen, dass sich Menschen mehr mit einer Webseite beschäftigen werden, wenn sie sich frei auf dieser bewegen können und sich dabei auf den Inhalt statt der Wartezeit konzentrieren können.[17]

[16] Vgl. https://www.thinkwithgoogle.com/marketing-resources/data-measurement/mobile-page-speed-new-industry-benchmarks, Stand 01.07.20
[17] Vgl. uxplanet.org (2019), https://uxplanet.org/how-page-speed-affects-web-user-experience-83b6d6b1d7d7, Stand: 16.05.2020

2.2 Zusammenhang von Performance und dem Page Rank

Der PageRank ist ein Algorithmus Verfahren, welches Google benutzt um die Linkpopularität[18] einer Webseite zu beurteilen. Die Linkpopularität wird von Suchmaschinen zur Bewertung von Webseiten beim Suchmaschinenranking verwendet und spielt deshalb bei der Suchmaschinenoptimierung eine wichtige Rolle. Ohne den Aufbau dieses Algorithmus im Rahmen dieser Arbeit genauer zu betrachten, soll trotzdem festgehalten werden, dass ein hoher PageRank erstrebenswert ist um beim Suchmaschinenranking[19] eine gute Position der eigenen Webseite zu erlangen.

Im Jahr 2010 kündigte Google an, dass die Seitengeschwindigkeit als einer der Ranking-Faktoren für den Suchindex aufgenommen wird. [20] Im Jahr 2018 kündigte Google an, die Seitengeschwindigkeit noch stärker zu berücksichtigen, indem die Geschwindigkeit mobiler Websites berücksichtigt werden.[21] Die Relevanz der Performance für den Rang lässt sich auf zwei Ursache zurückführen. Zum einen gibt es, wie bereits im vorherigen Kapitel dargelegt, einen Zusammenhang zwischen User Experience und Performance. Langsame Seiten wirken sich negativ auf die UX aus und Google möchte Seiten mit schlechten Nutzererfahrungen nicht gut ranken.

Darüber hinaus bedeutet eine langsame Seitengeschwindigkeit, dass Suchmaschinen mit ihrem zugewiesenen Crawling-Budget weniger Seiten crawlen können. Dies kann sich negativ auf Ihre Indexierung auswirken.

Google selbst spricht von mehr als 200 Faktoren, die das Ranking beeinflussen wie und in welchem Maß die Faktoren gewichtet werden, bleibt nach wie vor ein Firmengeheimnis, doch die Ladezeiten einer Webseite möglichst gering zu halten, ist aus Sicht der SEO für ein gutes Ranking durchaus relevant. [22]

[18] Maßstab für die Anzahl und Qualität von Hyperlinks, die auf eine Webseite weisen
[19] Reihenfolge, in der die bei der Benutzung der Suchmaschine ermittelten Ergebnisse aufgeführt werden
[20] Vgl. Amit Singhal (2010), https://webmasters.googleblog.com/2010/04/using-site-speed-in-web-search-ranking.html, Stand 17.05.2020
[21] Vgl. Zhiheng Wang and Doantam Phan (2018), https://webmasters.googleblog.com/2018/01/using-page-speed-in-mobile-search.html, Stand 17.05.2020
[22] Vgl. https://moz.com/learn/seo/page-speed, Stand 17.05.2020

3. Gatsby JS

Im folgenden Kapitel wird das Fallbeispiel sowie dessen Zielsetzung vorgestellt und vorab eine kurze Einführung zu den benutzten Technologien, GatsbyJS und dem Content Management System WordPress gegeben. Im Zuge dessen werden auch die Einsatzmöglichkeiten und etwaigen Grenzen des Tools herausgestellt.

3.1 Grundagen GatsbyJS

GatsbyJS ist ein auf React und GraphQL basierender, statischer Seitengenerator. Statisch bedeutet dabei nichts anderes, als dass Gatsby für den Anwender Dateien bereitstellt, die anschließend auf einen Server geladen werden können. Bei einer Serveranfrage werden, diese Dateien an den Client zurückgegeben, ohne dass zuvor die zurückgebende Seite gerendert werden muss. Die bisherige Funktionsweise vieler Webseiten beschränkt sich darauf, dass bei deren Besuch die Seite serverseitig dynamisch generiert wird. Eine klassische Seite mit WordPress funktioniert nach diesem Schema. Serverseitig werden über eine Datenbankabfrage alle notwendigen Daten beschafft und im Anschluss über vordefinierte Methoden zu einer Webseite, unter Einbeziehung layouttechnischer Konfigurationen, erstellt.

Gatsby stellt nun einen weiteren Ansatz zur Verfügung. Hierbei erfolgt die Erstellung diverser Seiten zu bestimmten Zeitpunkten (manuell oder automatisiert). Hierbei werden über einen Build-Prozess alle notwendigen Dateien generiert, die im Anschluss einem Server übergeben werden können. Im Anschluss hat der Server nur noch die Aufgabe zu einer Abfrage die entsprechenden Seiten bzw. Dateien an den Client zurückzugeben. Durch diese Vorgehensweise verspricht GatsbyJS Performancevorteile gegenüber dynamischen Seiten. Durch die Rückgabe statischer Seiten verbessert sich zudem das SEO einer Seite, da so ein Webcrawler nun alle vorhandenen Seiten mit samt ihren statischen Inhalten durchlaufen kann.

Bleibt zu klären was mit Generator gemeint ist. Im Falle von GatsbyJs ist damit ein Framework, mit dem infolgedessen Web-Projekte erstellt werden können, gemeint. Ein Framework ist eine Architektur aus Klassenhierarchien, die eine allgemeine Lösung für ähnliche Probleme in einem bestimmten Kontext zur Verfügung stellt. Konkret gesagt heißt das, dass Gatsby eine allgemeine Lösung zur Generierung von statischen Webseiten im Kontext der Webentwicklung bietet. Als Teil seines „Tooling Systems“ benötigt Gatsby NodeJs [23] um alle Dateien bzw. Seiten generieren zu können. Für das Endergebnis muss NodeJS jedoch nicht auf der Serverseite ausgeführt werden.[24]

[23] Node.js ist eine serverseitige, Event basierte JavaScript-Laufzeitumgebung
[24] Vgl. Zac Gordon (2019), https://javascriptforwp.com/what-is-gatsby-js-and-why-use-it, Stand 18.05.2020

Die Funktionsweise und die eingesetzten Technologien lassen sich mit folgender Abbildung gut zusammenfassen.

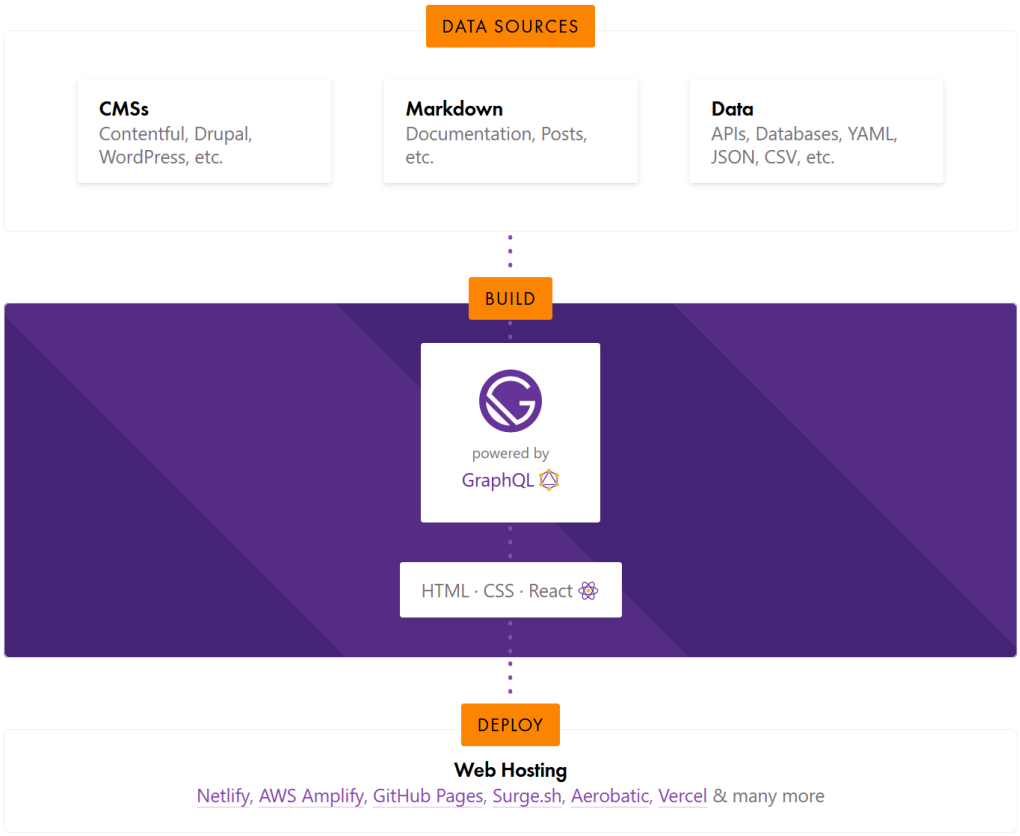

Abbildung 6: Wie GatsbyJS funktioniert[25]

3.2 Einsatzmöglichkeiten und Grenzen von GatsbyJS

Nachdem im vorigen Kapitel das grundlegende Wissen über Gatsby vermittelt wurde, stellt sich nun die Frage wann man GatsbyJs einsetzen soll und wann es sich eher weniger eignet.

Der wohl häufigste Einsatzzweck ist die Verbesserung der Seitenperformance, die sich aus der Nutzung der statisch generierten Seiten ergibt.

Da es bei der Verwendung von Gatsby keine Live-Datenbank gibt, bietet es auch weniger Angriffsfläche für Hacker. Es gibt keine Benutzerdaten, die auf dem Server mit der Gatsby-Site gespeichert werden. Selbst wenn jemand in der Lage wäre, den Server selbst zu hacken, erhält er nur Zugriff auf HTML-Dateien und kann weitaus weniger Schaden anrichten, als wenn er beispielsweise Zugriff auf eine WordPress-Site erhalten würde oder Zugriff auf User Daten

[25] Vgl. https://www.gatsbyjs.org, Stand 18.05.2020

hätte. Webseitenbetreiber, die mit Gatsby arbeiten, erzielen also auch enorme Sicherheitsgewinne.

Für Seitenbetreiber, die ihre laufenden Serverkosten senken möchten, könnte Gatsby auch in Frage komme. Da das Hosting einer dynamischen Webseite erfordert, dass der Server mit dem genutzten Technologiestack kompatibel ist, ist der Webseitenbetreiber oft stark eingeschränkt in der Auswahl der Anbieter. Eine statische Webseite hingegen kann auf fast jedem Server gehostet werden, was auch die mit dem Hosting verbundenen Kosten senkt.

Wofür sich Gatsby weniger eignet:

Für Personen mit wenig technischem Verständnis bzw. Wissen ist Gatsby nicht geeignet. Da der statische Seitengenerator auf React und GraphQL basiert, bedarf es einem Grundverständnis in JavaScript und GraphQL um mit einer Webseite arbeiten zu können und sie in ein Gatsby-Projekt zu importieren. Wer auf seiner Webseite viel dynamischen Inhalt darstellen möchte, muss bei der Nutzung von Gatsby neu überdenken wie dieser kontrolliert und ausgeliefert werden soll. Jede Änderung auf der Seite kann erst dann dargestellt werden, wenn die Seiten neu generiert und auf den Server geladen wurden, müssen. Dazu muss eine Mischung aus statischen und dynamischen Elementen erstellt werden. Für Funktionen wie beispielsweise die Kommentarsektion unter einem Blog müssen dann über neue Möglichkeiten der Umsetzung nachgedacht werden. Soll mehrmals am Tag neuer Inhalt auf die Seite eingepflegt werden oder eine häufig genutzte Kommentarsektion bereitgestellt werden, so ist ein statischer Seitengenerator aufgrund hoher Build-Zeiten ungeeignet. [26]

3.3 Einführung Wordpress

WordPress ist das am weitesten verbreitete Content Management System zum Betrieb von Webseiten mit ca. 50% Anteil an allen CMS bzw. 32% Anteil aller Websites[27]. Word Press bietet Kunden die Möglichkeit Webseiten in Eigenregie zu verwalten und die Möglichkeit ihre Website-Inhalte ohne Kenntnisse von Technik oder Programmierung schnell und einfach zu verwalten. Das CMS basiert auf der Skriptsprache PHP und benötigt eine MySQL- oder MariaDB-Datenbank, in der Daten abgespeichert werden. Die Benutzeroberfläche ist wie gewohnt über das Web (www.<NAME_DER_SEITE>/wp-admin) erreichbar, wobei auch Inhalte über E-Mail oder die WordPress App veröffentlicht werden können.

[26] Shaumik Daityari (2020), https://kinsta.com/de/blog/gatsby-wordpress, Stand 20.05.2020
[27] Vgl. https://trends.builtwith.com/cms, Stand 15.04.2020

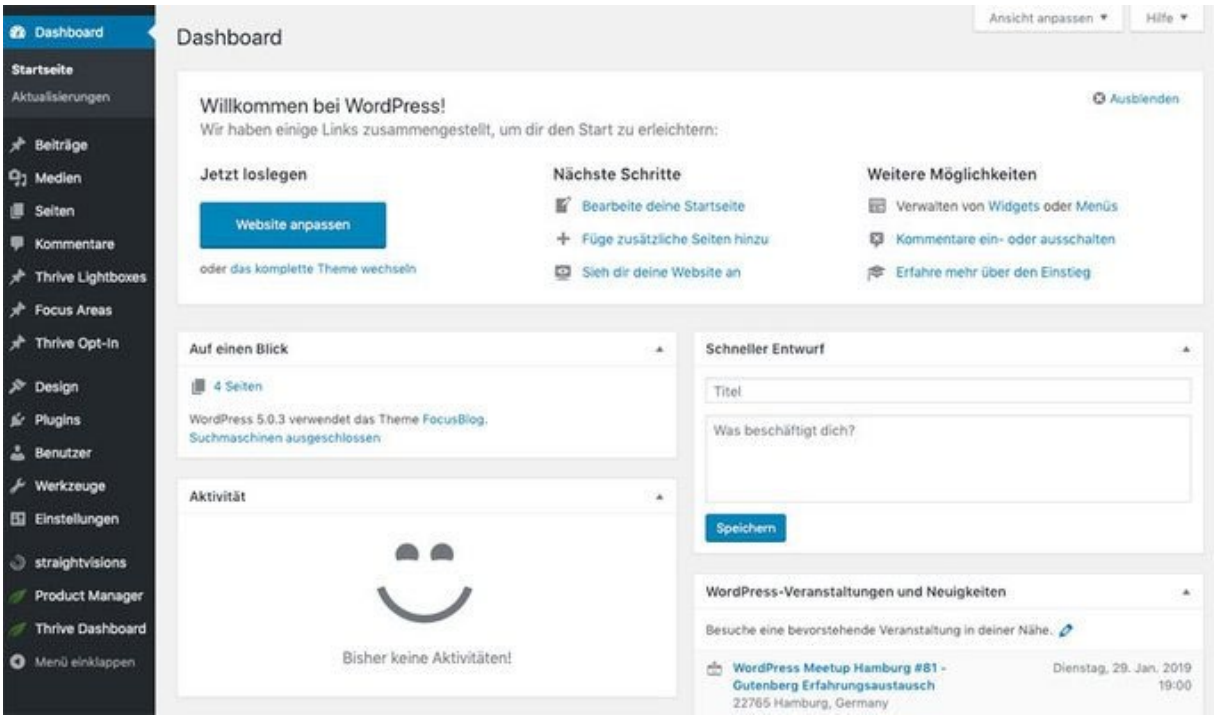

Abbildung 7: Admin-Oberfläche von Wordpress

Als Ausgangspunkt für die grundlegenden Funktionen dient das Admin Dashboard (Abbildung 6) mit der seitlichen Navigationsbar. Innerhalb dieses Dashboards gibt es etliche Möglichkeiten für den Nutzer, die Seite zu verwalten und zu konfigurieren. Im Folgenden werden kurz die wichtigsten Funktionen erläutert. Als CMS-System bietet Wordpress eine hervorragende Möglichkeit Inhalte einzupflegen oder zu ändern. Es wird grundsätzlich zwischen Seiten und Blog-Beiträgen unterschieden. Der Inhalt lässt sich über den von Wordpress bereitgestellten Gutenberg Editor verwalten.

Weiterhin bietet WordPress die Möglichkeit Kommentareinträge und Links zu verwalten. Neben der Erstellung und Verwaltung von Inhalten bietet Wordpress unter anderem die Möglichkeit Plugins für eine große Menge an Zusatzfunktionalitäten und Probleme in das Wordpress-Projekt zu integrieren.

Das Aussehen einer Webseite wird in WordPress durch Themes festgelegt. So werden Design und Programmkern von WordPress getrennt, wodurch individuelle Designs entwickelt werden können, ohne mit der Programmierung der Software an sich vertraut zu sein. Hierbei bietet Wordpress eine große Menge an kostenloser wie auch kostenpflichtiger Themes an. Unter einem Theme versteht man in Wordpress eine Ansammlung von php Methoden, mit denen nach einem bestimmten Muster Seiten zu Inhalten generiert werden. Zudem werden einige Schnittstellen über php zu Verfügung gestellt. Ebenso sind alle benötigten JS und CSS-Dateien in dem Editor vorhanden. Über diese PHP-Methoden lässt sich ein Theme anpassen bzw. erweitern.

3.3.1 Wordpress als Headless CMS

Ein Headless Content Management System kennzeichnet sich dadurch, dass das Frontend und Backend nicht mehr als Ganzes miteinander verknüpft sind. Der Begriff "kopflos" stammt aus dem Konzept, den "Kopf" (das Front-End, d.h. Die Website) vom "Körper" (dem Back-End, d.h. Dem Content-Repository) abzuhacken. Als Kommunikationsschnittstelle um den Inhalt, der im Backend eingepflegt wurde zu erreichen, dient dann eine REST-API (Representational State Transfer-Application Programming Interface). Mit den HTTP-Anfragemethoden wie GET, POST, PUT oder DELETE kann ein Client auf die Informationen des Servers zugreifen, diese abrufen oder gar verändern.

Wird also Wordpress als Headless CMS verwendet, kann das Front-End einer Webanwendung mit einer beliebigen Webtechnologie erstellt werden und deren Inhalt wird dann mit Wordpress verwaltet. Über das WP-REST-API Plugin kann dann diese Kommunikation ermöglicht werden.[28]

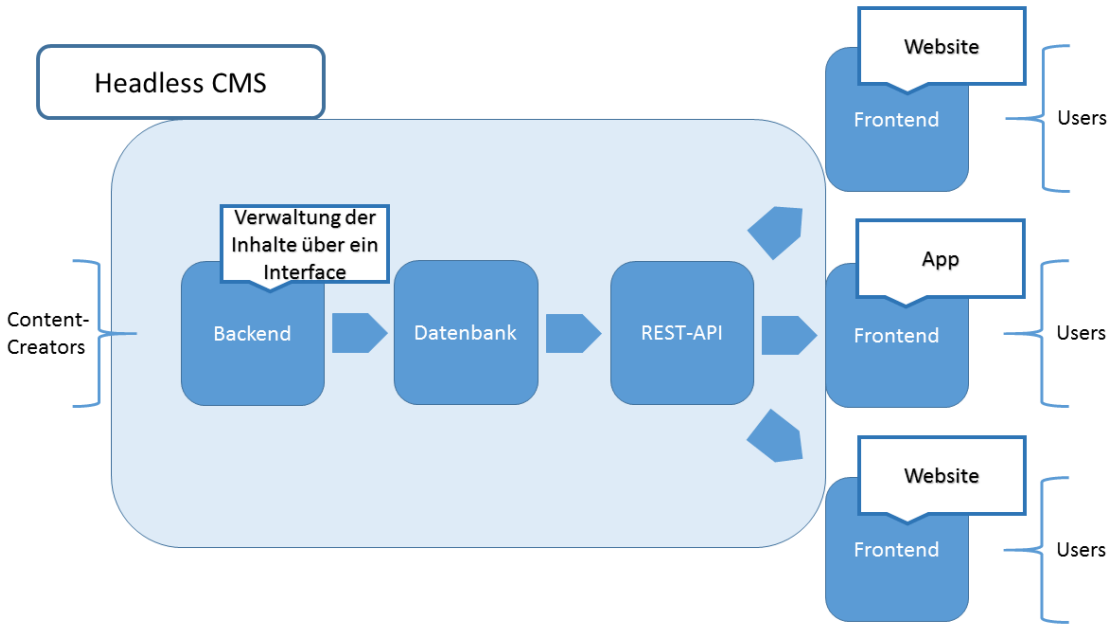

Abbildung 8: Architektur bei Verwendung eines Headless CMS[29]

[28] Ibad Ur Rehman (2020), https://www.cloudways.com/blog/use-react-with-wordpress-to-create-headless-cms/#benefitsofheadlesscms, Stand 22.05.2020
[29] Vgl. https://www.ionos.de/digitalguide/hosting/cms/headless-cms-was-sind-die-vorteile, Stand 18.05.2020

3.4 Fallbeispiel

Im Rahmen der Projektarbeit soll nun Gatsby auf Grundlage des Blogs (https://web-forward.de/) eingesetzt werden. Da sich hinter der Domain eine auf Wordpress-basierende Seite verbirgt, eignet sich die Seite hervorragend als CMS-Quelle für ein Gatsby-Projekt. Ziel hinter diesem Vorhaben ist es die Funktionalität des „statischen Seitengenerators“ kennen zu lernen. Zudem sollen Anwendungsszenarien bestimmt werden, in denen die Nutzung Sinn ergibt. Im Anschluss daran erflogt eine kurze Validierung der Ergebnisse über bestimmte Tools, mit denen eine Webseite bewertet werden kann. Im Folgenden wird das Projekt in der Reihenfolge eines fiktiven Entwicklungsablaufs vorgestellt.

Hinweise zur Verwendung des Projektes

Um die im folgenden beschriebenen Schritte besser nachvollziehen zu können, eignet sich es sich gegebenenfalls das Gatsby-Projekt zu klonen, um einen genaueren Blick auf die Funktionalität und die konkrete Realisierung zu erhalten:

➔ **Git clone https://github.com/simonx44/projektarbeit.git**

➔Nach dem Clonen: **npm install**

➔Entwickeln: **gatsby develop**

4. Die Umsetzung mit GatsbyJS

Wie bereits oben erwähnt, wird in dem Fallbeispiel Wordpress als Datenquelle verwendet. Üblicherweise dient Wordpress in diesem Kontext mit Gatsby als Headless CMS. Damit ist die Gestaltung und der Inhalt strikt getrennt. Die Funktionalität eines Backend übernimmt Wordpress, währenddessen über Gatsby das FrontEnd erstellt wird.

Ein typisches Anwendungsszenario wäre daher FrontEnd und Backend strikt zu trennen. Konkret bedeutet dies, dass Wordpress nur als Datenquelle dient und so als Backend keine Gestaltungsmöglichkeit auf Blogbeiträge erlaubt. Dies ist vor allem für Seiten wichtig, die eine einheitliche Darstellung ihrer Inhalte vorsehen (Corporate Design).

Wie schon in Wordpress gibt es auch für Gatsby Themes, auf deren Grundlage sich Webseiten aufbauen lassen. In dem oben genannten Kontext bedeutet dies jedoch auch, dass das FrontEnd rein über GatsbyJS erstellt und verändert werden kann und soll.

Will man nun dennoch einem Autor bzw. Redakteur gewisse Freiheiten in der Gestaltung ermöglichen, entsteht ein zusätzlicher Mehraufwand in der Implementierung. Die klare Trennung von Front- und Backend muss leicht aufgebrochen werden. Das Problem ist, dass

der Gutenberg-Editor in Wordpress etliche Gestaltungsoptionen standardmäßig über CSS-Klassen umsetzt. GatsbyJS beschafft im Anschluss den Inhalt eines Blogs in Form eines Html-Textes. Dieser enthält logischerweise auch CSS-Klassen, die nicht aufgelöst werden können. Da nun FrontEnd und Backend getrennt sind, gibt es auf FrontEnd Ebene keine Möglichkeit diese Klassen aufzulösen, außer man nimmt gewisse Anpassungen vor:

a) Von Oben nach Unten (Vom Frontend zum Backend)

Nach der Fertigstellung des Frontend innerhalb des Projektes mit GatsbyJS muss das in Wordpress hinterlegte Theme angepasst werden. Dieses besteht stets aus etlichen PHP-Methoden, die die Inhalte der Seite in das korrekte Format (Blog-Seite / Normale-Seite) auflösen, und entsprechende CSS-Klassen in die HTML Struktur aufnehmen. Hierbei ist es wichtig, dass das hinterlegte Theme denselben Aufbau (sowohl HTML wie auch CSS) wie innerhalb des Projektes hat. Im Anschluss lassen sich Layoutänderungen innerhalb des Gutenberg Editor ohne Probleme im Frontend anzeigen.

b) Von Unten nach Oben (Vom Backend zum Frontend)

Hat man bereits eine auf Wordpress basierende Seite und versucht nun die Seite zu optimieren, in dem Wordpress eingesetzt wird, kann auch ein bereits vorhandenes Wordpress Theme in ein Gatsby Theme umgewandelt werden. Konkret bedeutet das, dass innerhalb des Frontend das bestehende Wordpress Theme nachgebaut und alle CSS-Dateien lokal in das Gatsby-Projekt integriert werden. Sämtliche Theme-Funktionalitäten, insbesondere die PHP-Methoden, die die Wordpressseite rendern, müssen in GatsbyJS über React nachgebildet werden. Die im Editor hinterlegten CSS-Dateien können einfach in das Projekt kopiert werden. So kann im Anschluss sichergestellt werden, dass durch GatsbyJS ein genaues Abbild der Originalseite entsteht. Im Zuge des Projektes haben wir uns für die zuletzt genannte eher unübliche Möglichkeit entschieden, ein bestehende Wordpress Seite mit GatsbyJS nachzubauen.

4.1 Projektinitialisierung

Um Gatsby nutzen zu können muss Node.js installiert werden. Node.js ist eine Entwicklungsumgebung, die es erlaubt JavaScript außerhalb eines Webbrowsers auszuführen. Zudem ermöglicht die Installation die Nutzung von NPM um Abhängigkeiten zu nützlichen Bibliotheken zu regeln. So lassen sich Erweiterungen und Plugins für das Projekt installieren. Im Anschluss lässt sich Gatsby über **npm install -g gatsby-cli** installieren. Sind diese Schritte erledigt, kann Gatsby genutzt werden. Ein neues Gatsby Projekt lässt sich beispielsweise über „gatsby new hello-world https://github.com/gatsbyjs/gatsby-starter-hello-world“ erstellen. (Format: gatsby new <NAME> <Repositorium>

Wird der Zusatz „Repositorium“ weggelassen, wird automatisch ein Default Repositorium geladen. Durch den oben gezeigten Befehl wird ein neues lokales Projekt erstellt, indem ein vorhandenes Repositorium geklont wird. Mag man so beispielsweise ein Gatsby-Theme nutzen, wird einfach das zugrundeliegende Repositorium mit angegeben.

4.2 Projektstrukur

Jedes Gatsby Projekt weist eine vorgegebene Struktur auf, die bei der Initialisierung erzeugt bzw. geklont wird. Im Folgenden werden die wichtigsten Komponenten vorgestellt. In der späteren Betrachtung des Projektes wird ein genauerer Blick auf das Projekt geworfen.

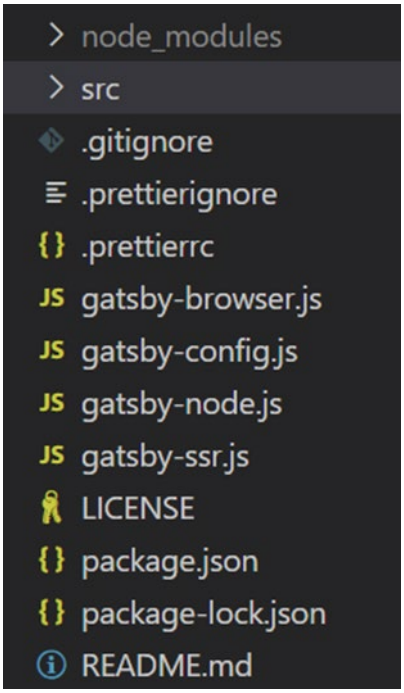

Abbildung 9: Projektstruktur

1. **/node_modules:** Alle Erweiterungen, die sich über NPM installieren lassen, befinden sich als Code in dem Ordner
2. **/src:** Enthält alle notwendigen Dateien, die für den Bau einer Seite benötigt werden. Hierzu zählen Templates, React-Komponenten, Bilder, CSS-Dateine ….
   Allgemein liegt hier der Code für das Front-End des Projektes
3. **/.gitignore:** Wichtig wenn Git genutzt wird, enthaltene Dateien werden ignoriert
   Standardmäßig ist hier zu Beginn der Ordner **/node_modules** enthalten, der eine große Menge an Daten besitzt und nach jedem Clonen neu über NPM generiert werden kann
4. **.prettierrc:** Prettier lässt sich konfigurieren, um den Code einheitlich zu formatieren
5. **Gatsby-browser.js:** Das Verhalten von Gatsby im Browser lässt sich konfigurieren.
   Zudem lassen sich Dateien einbinden, die so für das Projekt

global sind

6. **Gatsby-config.js:** Hauptkonfigurationsseite für Gatsby. Hier lassen sich Plugins konfigurieren
7. **Gatsby-node.js:** Hier wird die Gatsby Node API verwendet, um so alle benötigten Seiten zu generieren
8. **Gatsby-ssr.js:** Nutzung der Server-Side rendering API, erlaubt Konfiguration des Standard-Verhaltens (wird im Projekt nicht genutzt)
9. **Package-json:** Abhängigkeiten zu Bibliotheken werden hier festgelegt
10. **README.md:** Textdatei, die nützliche Informationen zum Projekt enthalten kann

Wichtige Gatsby-Anweisungen

Gatsby develop - Startet den Entwicklungsserver, Änderungen am Projekt sind direkt sichtbar -> Erreichbar über http://localhost:8000/

Gatsby build – Projekt wird gebaut, generierte Dateien werden in Ordner **PUBLIC** gespeichert

Gatsby serve – Fertig gebildetes Projekt über Browser ausführen, Serverzugriff wird simuliert

Gatsby clean – Löscht Cache und den durch den Build Prozess erstellten **PUBLIC** Ordner

4.3 Datenbeschaffung über Plugins

Für eine Webseite mit Gatsby können Daten in unterschiedlichen Varianten vorliegen und beschafft werden. Neben der im Fallbespiel genutzten Variante die Daten von einem CMS zu beschaffen, gibt es für kleinere Projekte ebenso die Möglichkeit Daten aus einem Markdown (.md) zu beschaffen. Die Gatsby Dokumentation liefert für dieses Vorhaben eine detaillierte Erklärung. Zudem ist ein direkter Zugriff auf APIs, Datenbank etc. möglich.

In Zusammenhang mit einem CMS geschieht die Datenbeschaffung über ein Plugin, das sich über NPM einfach in das Projekt integrieren lässt. Alternativ lassen sich diese Abhängigkeiten auch manuell über die package.json regeln. Dieses Plugin beschafft automatisch über eine Schnittstelle alle Daten zu einem CMS.

Npm-Anweisung: `npm install --save gatsby-source-wordpress`

Im Anschluss muss noch innerhalb von gatsby-config.js das Plugin auf die Datenquelle konfiguriert werden. Für jedes Plugin gibt es in der Regel eine Menge an Konfigurationsmöglichkeiten. Diese befinden sich alle in einem JS-Objekt, das von der eben genannten JS-Datei exportiert wird. Innerhalb des Projektes wurden mehrere Plugins installiert. In der Projektarbeit wird im Anschluss eine Anpassung eines Plugins beispielhaft erklärt. Auf weiteren Plugins wird im Anschluss nicht mehr im Detail eingegangen.

Abbildung 10: Datenbeschaffung

Um nach der Installation das Plugin nutzen zu können, muss das jeweilige Plugin über resolve: *PluginName* in das Objekt aufgenommen werden. Zudem kann ein Plugin über die Property *option* konfiguriert werden. Die möglichen Optionen sind der Dokumentation zu entnehmen. In dem Beispiel der Abbildung muss die Wordpress-Url sowie das Protokoll angegeben werden. Für Wordpress muss der Präfix wp-json mitangegeben werden, sodass auf die Wordpress-API zugegriffen werden kann.

Das Plugin sorgt nun beim Builden, aber auch während der Entwicklung dafür, dass alle Daten, für die eine Autorisierung vorliegt, von der API beschafft werden. Sollte eine Autorisierung benötigt werden, kann diese gemäß Dokumentation über Felder innerhalb des *Options* Objekt hinzugefügt werden.

Wie in dem Beispiel zu sehen ist, lassen sich Plugins relativ einfach und ohne viel Aufwand in das Projekt integrieren. Stößt man nun einen Prozess mit **gatsby develop oder gatsby build** an, werden die Daten beschafft. Im Technischen heißt das, dass Gatsby sich alle Daten der Wordpress-API besorgt und im Anschluss in ein Graphql-nutzbares Format umwandelt. So kann im Anschluss innerhalb des Projektes über Graphql-Queries auf die Daten zugegriffen werden. Zudem stellt Gatsby während der Entwicklung den Graphiql –

Explorer[30] zu Verfügung. Dieser ist über die Url http://localhost:8000/___graphql, während der Entwicklung erreichbar. So können relativ einfach Queries erstellt und im Anschluss direkt getestet werden, um diese danach hard-codiert in das Projekt aufzunehmen. Wie in der nachfolgenden Abbildung zu sehen ist, bietet dieser Explorer neben einem Editor und dem Ergebnisbereich zu einer Abfrage auch eine detaillierte Übersicht der Struktur aller vorhandenen Daten. So lassen sich Queries aufgrund der veranschaulichten Struktur der hinterlegten Daten einfach erstellen.

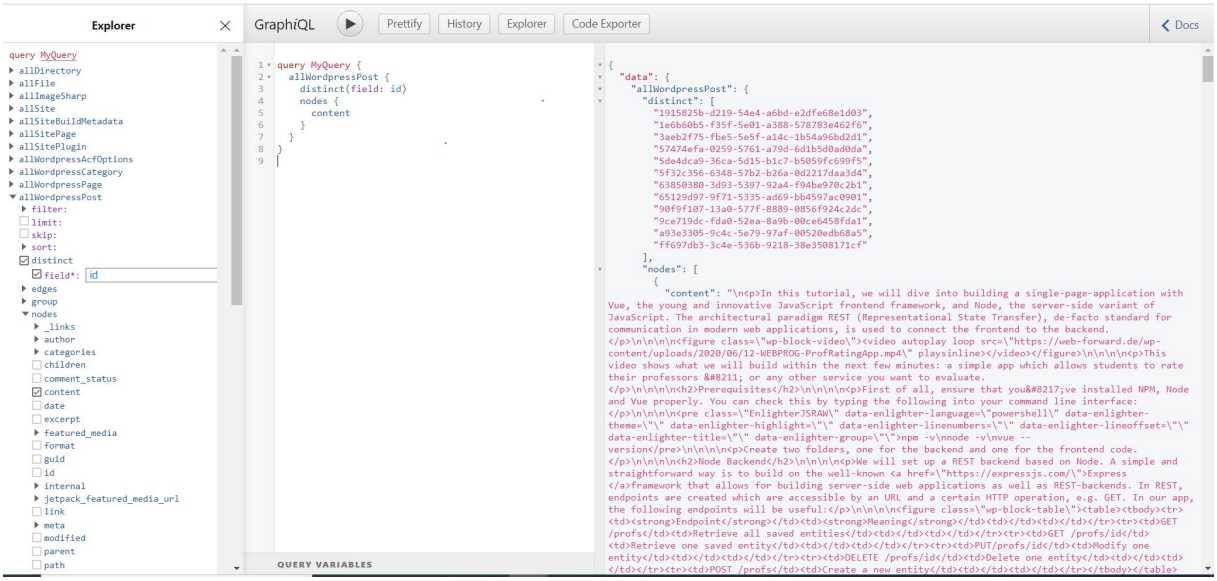

Abbildung 11: GraphiQL

Das Ergebnis einer Query wird in einer JSON-Datei gespeichert.

4.4 Seitengenerierung über Gatsby

Wie bereits oben kurz aufgeführt wird die Datei „gatsby-node.js“ benötigt, um Seiten eines Gatsby Projektes zu erzeugen. Gatsby bietet eine Menge an sogenannten Gatsby APIs an, die je nach Anwendungsfall für das Projekt genutzt werden können. Um Seiten zu generieren, wird die API „createPages“ genutzt, die aufgerufen wird, sobald alle benötigten Daten geladen und das Graphql-Schema erstellt sind. Konkret gesagt, wird darauf gewartet, dass die initiale Datenbeschaffung über das zuvor genannte Plugin abgeschlossen ist.

[30] GraphiQL ist die integrierte GraphQL-Entwicklungsumgebung (IDE)

```
exports.createPages = async ({ graphql, actions }) => {
  const { createPage } = actions



  return new Promise((resolve, reject) => {

    // Posts
    graphql(postQuery)
      .then(result => {
        if (result.errors) {
          console.log(result.errors)
          reject(result.errors)
        }

        const postTemplate = path.resolve(`./src/templates/post.js`)
```

Abbildung 12: Gatsby API createPages

In der Abbildung wird ein Ausschnitt des Codes zur Nutzung der API gezeigt. Zur Generierung aller benötigten Seiten wird der Funktion über Destrucuring ein Objekt übergeben, das wiederum eine Funktion und ein weiteres Objekt (actions) enthält. Innerhalb dieses Objektes sind mehrere nützliche Methoden, die sich ebenso über ES6 (object destructuring) extrahieren lassen. Für das Anwendungsbeispiel wird nur die „Action“ bzw. Funktion „createPage“ benötigt.

Um eine sukzessive Abarbeitung des Codes zu ermöglichen, werden Promises genutzt. So wird sichergestellt, dass die Seitengenerierung erst angestoßen wird, sobald alle Daten vorhanden sind. Diese Daten werden über die Funktion *graphql,* zur Ausführung einer Graphql-Abfrage, beschafft. Wie in der Abbildung zu sehen ist, wird diese Methode innerhalb des Objektes im Parameterbereich mitgegeben. Diese benötigt als Parameter eine Graphql-Query. Nachdem die Daten der Query vorhanden sind, kann zu jedem zurückgegeben Eintrag eine Seite generiert werden. In der folgenden Abbildung wird dies exemplarisch für alle Posts der Wordpress-Seite gezeigt. Für jeden gefunden Post wird so ein Pfad erstellt. Zudem wird ein Template angegeben, welches den Aufbau der zu erstellenden Seite vorgibt. Im nachfolgenden Kapitel wird zum besseren Verständnis auf dieses Template genauer eingegangen. Ebenso wird das Objekt *context* benötigt, in dem festgelegt wird, welche Daten zur Erzeugung der Seite mitgegeben werden. Die eben aufgeführten Daten, werden innerhalb eines Objektes an die Action *createPage* mitgegeben (siehe Abbildung). Der Ablauf für die Erstellung alle zugehörigen Seiten einer Wordpress-Seite erflogt synchron zu diesem Beispiel. Zusammenfassend wird innerhalb der Datei „gatsby-node.js“ eine Funktion/API createPages exportiert, innerhalb dieser zwei Methoden der Gatsby API (graphql, createPage) genutzt werden. Nachdem die Daten vorhanden sind, wird die

Methode createPage aufgerufen, die innerhalb eines Objektes ein Pfad, eine Komponente für die Auflösung über ein Template (React-Komponente) und ein Objekt *context* mit Inhalt zu dem Blogbeitrag mitgegeben wird. Unter dem Pfad wird im Anschluss der Blogbeitrag im Browser erreichbar sein. Die Komponente dient zur Auflösung der Seite. Das bedeutet, dass createPages, die mitgegebenen Daten an das Template schickt, in dem die eigentliche Generierung und das Rendern einer Seite erfolgt. Innerhalb des Context befinden sich Daten, die im Anschluss auch innerhalb der Templates zur Verfügung stehen. In dem Beispiel wird nur die Id des Blogbeitrags mitgegeben. Nachfolgend erfolgt für jeden gefundenen Knoten ein weiterer Datenzugriff über die mitgegebene Id. Alternativ wäre auch möglich, hier alle Daten direkt zu beschaffen und diese innerhalb von Context an das Template zu schicken, um eine weitere Abfrage zu umgehen.

```
  const postTemplate = path.resolve(`./src/templates/post.js`)
  result.data.allWordpressPost.edges.forEach(edge => {
    console.log(edge.node.title);
    createPage({
      // will be the url for the page
      path: edge.node.slug,  // /post/${edge.node.slug}/`,
      // specify the component template of your choice
      component: slash(postTemplate),
      // In the ^template's GraphQL query, 'id' will be available
      // as a GraphQL variable to query for this posts's data.
      context: {
        id: edge.node.id,
      },
    })
  })
})

.then(() => {
```

Abbildung 13: Seitengenerierung über gatsby-node.js

4.5 Templates

Templates werden wie bereits im vorangegangenen Abschnitt erwähnt, dafür eingesetzt, um alle Seiten von Wordpress über Gatsby zu generieren. Hierfür wird das Framework React genutzt, um so die Struktur der Seiten dynamisch in Abhängigkeit bestimmter Status zu rendern. Da Gatsby über den Wordpress Zugriff sowohl alle Seiten wie auch alle Blogbeiträge beschafft, werden in dem Projekt zwei Templates für jede Art von Seite angeboten. In der folgenden Abbildung sehen wir das Template für alle Wordpress-Seiten, dass aufgrund der gegebenen Kompaktheit besser zur Veranschaulichung dient als das Template für die Blogbeiträge, in dem viel mehr Logik steckt. React ermöglicht in Abhängigkeit der Daten und vordefinierter Status ein flexibles Rendern der Seiten.

Grundsätzlich wird innerhalb von Gatsby der moderne funktionale React Ansatz und nicht der klassische klassenbasierte Ansatz genutzt. Konkret bedeutet das, dass eine Methode die gesamte Aufgabe übernimmt, die zuvor über eine klassenbasierte Variante durchgeführt wurde. Dies ist aber nicht, wie später zu sehen ist, verpflichtend.

```
import React from "react"
import Layout from "../layout/layoutPage"
import SEO from "../components/seo"

export default ({ data }) => {
  const currentPage = data.wordpressPage

  return (
    <Layout>
      <SEO title={ currentPage.title} description={currentPage.excerpt} />
      <div className="post">
        <div className="post-inner">
          <div className="post-header">
            <h1 className="post-title">{currentPage.title}</h1>
          </div>

          <div
            className="post-content"
            dangerouslySetInnerHTML={{ __html: currentPage.content }}
          />
          <div className="clear"></div>
        </div>
      </div>
    </Layout>
  )
}

export const pageQuery = graphql`
  query pageQuery($id: String!) {
    wordpressPage(id: { eq: $id }) {
      title
      content
      id
      excerpt
      slug
    }
  }
```

Abbildung 14: Template für WP-Seiten

Wie in der Abbildung zu sehen ist, sollte ein Template aus mindestens zwei Komponenten bestehen.

Zum einen muss eine React-Komponente vorhanden sein, damit Inhalt generiert werden kann, und zum anderen sollte es eine graphql-Query geben, über die der Inhalt für die React-Komponente beschafft werden kann.

Wird nun innerhalb der Datei „gatsby-node.js“ die zuvor vorgestellte Action createPages({…} aufgerufen, springt GatsbyJS automatisch in das hinterlegte Template und führt dieses aus. Der Action wird innerhalb eins Objekts eine Property *Context* übergeben. Das Template kann nun auf den Inhalt dieses Objektes zugreifen. In dem Fallbeispiel wird nur die ID der gefundenen Seite an das Template mitgegeben (siehe Abbildung createPages). Diese ID dient nun im Anschluss als Restriktion für die aufzurufende Query. Hierbei können Daten

speziell für die ID beschafft werden. Nachdem die Daten beschafft sind, können diese von der React-Komponente genutzt werden, um die Seite zu generieren. Diese Daten werden als Parameter der Funktion übergeben. So kann im Anschluss der spezifisch zu der Id gehörige Inhalt über die Komponente gerendert werden. Unabhängig von den Daten besitzt das Template immer die selbe Struktur. Wie im Beispiel zu sehen ist, wird sowohl die Query wie auch die React-Funktion exportiert. Diese werden im Hintergrund von GatsbyJS zur Generierung von Seiten während des Builds automatisch aufgerufen.

4.6 Layout und Styling

Das Projekt ist aus mehreren unterschiedlichen Layouts aufgebaut, die jeweils in einer eigenen JS-Datei als React Komponente vorliegen. Diese JS-Komponenten liefern so ein spezifisches Ergebnis, das sich im Anschluss in weitere Komponenten integrieren lassen kann. So besteht jede Seite beispielsweise aus einem einheitlichen Header wie auch Footer. Diese beiden Komponenten lassen sich dann im Anschluss einfach in eine weitere Komponente packen, die für einen speziellen Seitentyp benötigt wird. In der folgenden Abbildung wird beispielhaft das Layout für alle Blog-Seiten dargestellt. Ein Template (Blog/Page) nutzt im Anschluss wiederrum dieses Layout, um eine Seite zu rendern. In der Abbildung wird die dafür zusätzlich erzeugte HTML-Struktur innerhalb von *{children}* geladen.

```
  return (
    <div className="post-template-default single single-pos single-format-standard has-featured-image">
      <Header></Header>
      <div className="wrapper">
        <div className="wrapper-inner section-inner thin">
          <div className="content">
            {children}
          </div>
        </div>
      </div>
<Footer/>
    </div>
  )
}

export default Layout
```

Abbildung 15: Layout und Styling

Der Aufruf erfolgt hierbei von unten nach oben. Ganz unten befindet sich das Template mit dem zugehörigen Inhalt, das wiederrum in ein Layout gebettet wird, welches weitere Komponenten nutzt. Für dieses Projekt ergeben sich hierfür drei unterschiedliche Layouts: Ein Layout für Blog-Seiten, ein Layout für Normale-Seiten und ein weiteres Layout für die Index-Seite.

Es gibt nun unterschiedliche Möglichkeiten, CSS in diese Komponenten aufzunehmen.

Die erste beschriebene Möglichkeit nutzt Standard CSS-Dateien um die Komponenten zu stylen. Hierfür müssen alle CSS-Dateien, die global genutzt werden wollen, innerhalb der *gatsby-browser.js* importiert werden.

Eine weitere Möglichkeit ist, sogenannte CSS Module zu nutzen, um speziell einzelne React-Komponenten zu stylen:

```
import containerStyles from "./container.module.css"
export default function Container({ children }) {
  return <div className={containerStyles.container}>{children}</div>
}
```

Wie der Code-Ausschnitt zeigt, wird eine CSS-Datei importiert, die im Anschluss nur innerhalb der Komponente sichtbar ist. Auf die einzelnen Klassen kann nun im Anschluss direkt über den Namen der Import + Name der Klasse zugegriffen werden.

Die zweite Methode ist der ersten in der Regel vorzuziehen, da sie mehr Sicherheit mit sich bringt.

Index-Seite

Neben allen dynamischen Seiten, die über die Action *createPage(..)* erstellt werden, gibt es noch eine weitere Möglichkeit über Gatsby Seiten zu erstellen. Hierfür werden keine Templates und Gatsby APIs benötigt. Jedes Projekt besitzt, wie der Projektstruktur zu entnehmen ist, zudem noch einen SRC Ordner, in dem die zuvor vorgestellten Layout und Template enthalten sind. Zudem gibt es innerhalb dieses Ordners ein Unterordner *pages*. Innerhalb dieses Ordners können JS-Script Dateien abgelegt werden, in denen weitere individuellen Seiten geladen werden. Alle Dateien innerhalb dieses Ordners liefern eine zusätzliche Seite, die im Anschluss unter der URL: seitenUrl/dateiname erreichbar sind. Die Bedeutung einer Index-Datei hat innerhalb von Gatsby keine besondere Zusatzfunktion. Während des Build-Prozesses wird die index.js in eine Index.html umgewandelt und im Anschluss in den Zielordner Public gepackt. Eine Datei innerhalb des Ordners *src* wird, wie in GatsybJS üblich, einfach über eine React-Komponente erstellt.

```
import React, { Component } from "react"
import Layout from "../layout/layoutIndex"
import IndexMain from "../components/indexMain"

class Homepage extends Component {

  render() {
    return (
  <Layout>

<IndexMain data={this.props.data}/>
  </Layout>
    )
  }
}

export default Homepage


export const pageQuery = graphql`
  query {
    allWordpressPost {
      edges {
        node {
          date(formatString: "DD.MMMM YYYY")
          title
          slug
          excerpt
          wordpress_id
          featured_media {
            localFile {
              childImageSharp {
                fluid {
                  ...GatsbyImageSharpFluid
                }
              }
            }
            alt_text
          }
        }
      }
    }
    allWordpressWpComments {
      nodes {
        post
      }
    }
  }
`
```

Abbildung 16: Homepage - Index.js

Die Abbildung zeigt beispielshaft den Code für die Indexseite, der über die klassenbasierte Variante von React umgesetzt wird. In der Regel sollte die neuere funktionsbasierte Variante von React genutzt werden, jedoch beeinträchtigt dies das Endresultat nicht. Die Komponente setzt sich abermals aus unterschiedlichen Komponenten zusammen, in denen Teile der Gesamtseite gerendert werden. Der eigentliche Inhalt der Seite wird innerhalb der React-Komponente <IndexMain> gerendert. Innerhalb dieser können Queries eingesetzt werden, die im bei Aufruf der Seite die benötigten Daten beschaffen. Um die Query innerhalb der Komponente auszuführen, kann die GatsbyJS-Funktion useStaticQuery genutzt werden.

Die Datenbeschaffung kann nun aber auch synchron wie bei den Templates funktionieren. Hierfür wird innerhalb der React-Komponente im SRC-Ordner die Query ausgeführt und exportiert. Im Anschluss können die Daten an weitere Komponenten über props weitergegeben werden (siehe Abbildung X2)

In beiden Fällen gibt es nun das Problem, dass die Query nur statisch sein kann. Weder über Variablen noch über Template Strings kann die Query verändert werden.

```
import React, { useEffect } from "react"
import { Link, graphql, useStaticQuery } from "gatsby"
import Img from "gatsby-image"
import SEO from "./seo"

export function Index() {
  const data = useStaticQuery(pageQuery)
  const posts = data.allWordpressPost.edges
  const comments = data.allWordpressWpComments.nodes
```

Abbildung 17:Queries über useStaticQuery()

```
class Homepage extends Component {

  render() {
    return (
    <Layout>

<IndexMain data={this.props.data}/>
    </Layout>
    )
  }
}

export default Homepage


export const pageQuery = graphql`
  query {
    allWordpressPost {
      edges {
```

Abbildung 18: Seitengenerierung über SRC-Ordner

4.7 Besonderheiten für den Build

Während des Build Prozesses werden alle für die Webseite benötigten Dateien erstellt und in den Ordner *Public* abgelegt, der im Anschluss direkt auf einen Server geuploadet werden kann. Hierfür werden alle Dateien in optimierte HTML, CSS und JavaScript Dateien umgewandelt. Während dieses Builds kann es jedoch zu Problemen kommen, wenn innerhalb des Codes auf die Browservariablen *Window* oder *Document* zugegriffen wird. Hierbei wird der Prozess aufgrund eines Fehlers abgebrochen und der Public Ordner kann im Anschluss nicht an den Server ausgeliefert werden.

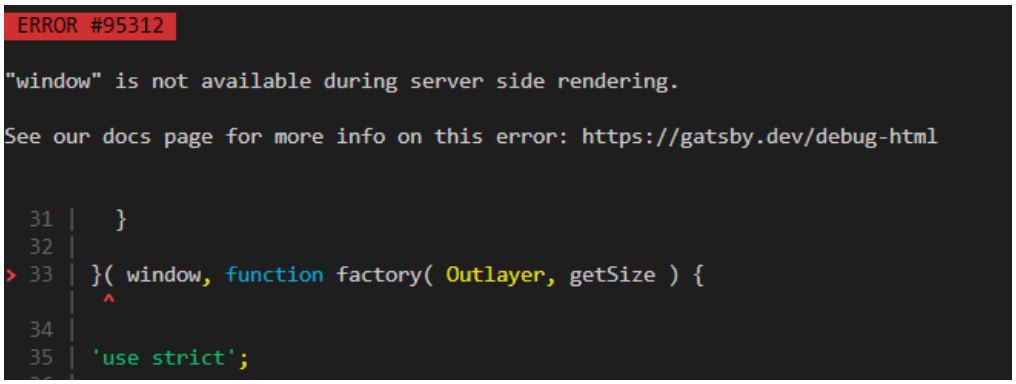

Abbildung 19: Fehlermeldung

Dieses Problem lässt sich lösen, indem während des Build geprüft wird, ob die Variable überhaupt vorhanden ist. Ist dies nicht der Fall, werden alle Operationen im Zusammenhang mit diesen Variablen nicht ausgeführt. In der Abbildung wird eine Lösungsoption im Zusammenhang mit der Index-Seite gezeigt. Für die spezielle Anordnung der Blog-Beitragsinformation wird die Bibliothek *masonry* genutzt. Diese wird nach dem Rendern aufgerufen und fügt angegeben Klassen CSS-Eigenschaften hinzu, die im Anschluss dieses Grid-typischen Design erzeugt. In der Abbildung wird dies nochmal verdeutlich.

```
useEffect(() => {
  const Masonry =
    typeof window !== `undefined` ? require("masonry-layout") : null
  if (!Masonry) {
    return;
  }

  var container = document.querySelector(".posts")

  var msnry = new Masonry(container, {
    // options...
    columnWidth: ".post-container",
    itemSelector: ".post-container",
  })
}, [])
```

Abbildung 20: Einsatz Bibliothek Masonry

Nach jedem Rendern() wird über die React-Hook *useEffekt()* geprüft, ob window zur Verfügung steht. Steht window zur Verfügung kann das Layout importiert und im Anschluss verwendet werden. Ist dem nicht so, wird der nachfolgende Code nicht ausgeführt.

Da der Build-Prozess nur zur Optimierung dient, ist dies nicht weiter schlimm. Wird später die besagte JS-Datei vom Server ausgeliefert, kann nach dem React-Rendern der Code ausgeführt werden, da window definiert ist.

4.8 Dynamischer Inhalt beschaffen und ändern

Gatsby als statischer Seitengenerator optimiert die Performance vieler Seiten, indem nicht aktiv serverseitig gerendert wird. Es wird in gewissen Zeiten der Build-Prozess entweder manuell oder automatisiert angestoßen, um die Daten einer Seite auf einen Webserver zu laden. Hierbei ergibt sich das Problem, dass die Daten einer Seite so maximal aktuell sind, wie der letzte Build zurückliegt. Möchte man nun trotzdem dynamische Daten innerhalb eines Projektes verwenden, muss wie bei anderen Seiten auch ein clientseitiger Zugriff auf die Datenquelle erfolgen. Im Fallbeispiel wird die JavaScript API fetch() hierfür eingesetzt. Hierbei wird auf ein Wordpress-API Endpunkt zugegriffen und alle Kommentare zu diesem Endpunkt in Form einer JSON beschafft.

```
// React HOOK -> wird nach jedem Render aufgerufen
useEffect(() => {

  //Kommentare der Api beschaffen
  const loadComment = id => {
    // document.addEventListener('DOMContentLoaded', function() {

    //Daten von API beschaffen
    return fetch(ACTION_URL + "?post=" + id)
      .then(response => response.json())
      .then(data => {
        setComments(data)
        return data
      })
  }
  loadComment(wpId)
}, [])
```

Abbildung 21: Dynamische Daten beschaffen

Wie in der Abbildung zu sehen ist, gibt es bei diesem Zugriff keinen Unterschied zu konventionellen Seiten, die nicht mit Gatsby arbeiten. Ähnlich verhält es sich mit dem Hinzufügen oder Ändern bestehender Inhalte. Wie dem Bild unten zu entnehmen ist, wird über ein POST-Request ein Objekt übergeben, um so im Anschluss einen Kommentar auf der Wordpress-Seite hinzuzufügen.

```
  fetch(ACTION_URL, {
  method: 'post',
  headers: {
    'Content-Type': 'application/json',
  },
  body: newComment,
})
  .then((response) => {

    if (response.ok === true) {
      // Submitted successfully!

    }
    return response.json();
  })
  .then((object) => {
    // Comment submission failed.
    // Output `object.message` to see the error message.
  })
  .catch(error => console.error('Error:', error));

  // Bestehende Eingabe löschen
  $("#commentform")[0].reset();
  return createComment(siteComment)
}
```

Abbildung 22: Kommentare Posten

5. Evaluation der Seiten Performance

Die von uns zu Testzwecken mit Gatsby gebaute Seite wurde unter der Subdomain http://gatsby.web-forward.de/ gehostet und wird mit der nur auf WordPress basierenden Seite https://web-forward.de/ verglichen. Dabei sollte noch erwähnt werden, dass CMS-Umgebungen noch optimiert werden könnten beispielsweise durch höhere Prozessorgeschwindigkeit, mehr Speicher, mehr Datenbank-Speicher oder bessere Internetanbindung. Außerdem ist die Infrastruktur von Strato (hier werden beide Seiten gehostet) nicht einsehbar. Optimierungen der CMS-Umgebung werden im Rahmen unserer Tests also außer Acht gelassen, da dies den Umfang der Arbeit sprengen würde.

Um sich einen Überblick zu verschaffen, in welchem Maße Verbesserungen durch den Einsatz von GatsbyJs erzielbar sind, werden im folgenden Kapitel mithilfe zweier Online Tools die Seiten Performance der beiden Seiten verglichen.

5.1 Vorstellung der Werkzeuge

Um das Ergebnis der Evaluation möglichst repräsentativ zu gestalten kamen zwei verschiedene Tools zum Einsatz. Kriterien für die Auswahl der Tools waren zum einen, dass eine kostenfreie Nutzung möglich ist, zum anderen welche Tools am häufigsten zum Einsatz kommen. Damit fiel die Wahl auf Google Lighthouse und GTMetrix.

5.1.1 Google Lighthouse

Lighthouse ist ein Open-Source-Google-Tool, das die Leistung von Webseiten analysiert. Der Schwerpunkt liegt dabei auf Webanwendungen und mobile Webseiten. Die Anwendung führt eine Reihe von Tests aus, die als Audit bezeichnet werden. Dadurch werden Informationen auf einer Webanwendung/-seite analysiert und anschließend ein Testbericht darüber erstellt, wie gut die App oder Seite funktioniert hat. Bei der Prüfung wird die Seite über eine simulierte, schwache 3G-Verbindung geladen, während sie auf einem langsamen Gerät angezeigt wird. Es simuliert auch Paket- oder Datenverlust, Netzwerk- und CPU-Drosselung. Durch diese Testumgebung würde selbst die größte, schnellste und optimierteste Seite der Welt leiden. Kann die Performance unter diesen belastenden Bedingungen verbessert werden, wird sich die Leistung auf einem schnellen Gerät, das mit einem schnellen Netzwerk verbunden ist, wirklich verbessern. Gute Ergebnisse sind dabei eine Punktzahl von 90-100. Der Durchschnitt liegt bei 50-89, während Punktzahlen zwischen 0-49 schlecht sind. Laut Google gehören Sie mit einer Punktzahl über 90 zu den Top 5% der gut funktionierenden Websites.

Bei Lighthouse wird nach folgenden Kriterien getestet:

Performance:

Da die Wahrnehmung von Geschwindigkeit ebenso wichtig wie die tatsächliche Geschwindigkeit ist, konzentriert sich der Bericht von Lighthouse auf eine Reihe von Dingen, die damit zusammenhängen, Inhalte so schnell wie möglich auf dem Bildschirm anzuzeigen. Der Abschnitt „Performance“ enthält damit folgende Aspekte:

- First Contentful Paint: Ein Maß für die Dauer bis Inhalte jeglicher Art auf dem Bildschirm angezeigt werden.
- First Meaningful Paint: Gibt an wie lange es dauert, bis der erste aussagekräftige Inhalt auf dem Bildschirm angezeigt wird. Je niedriger die Punktzahl, desto schneller wird diese Seite angezeigt.
- Speed Index: Ein Seiten-Geschwindigkeitstest, der zeigt, wie schnell der Inhalt einer Seite angezeigt werden kann. Der Index basiert auf einer Ladezeit unter 1,25 Sekunden.
- First CPU Idle: Die Zeit, in der das Gerät nicht mehr zum Rendern der Seite arbeitet.
- Time to Interactive: Bedeutet, dass die meisten Elemente der Benutzeroberfläche interaktiv sind und der Bildschirm auf Benutzereingaben reagiert.
- Estimated Input Latency: Misst, wie lange es dauert, bis eine Seite auf Benutzereingaben reagiert. Je geringer die Latenz, desto schneller fühlt sich die Seite an. Das Ziel für die Eingangslatenz beträgt weniger als 50 Millisekunden.

Accesibility

Dieser Abschnitt des Berichts enthält Überprüfungen für eine Seitenüberschrift. Dazu gehört auch, ob Hintergrund- und Vordergrundfarben einen ausreichenden Kontrast aufweisen, ein Dokumenttitel-Tag, Linknamen und ob das Ansichtsfenster vom Betrachter skalierbar ist.

Best Practices

In diesem Abschnitt werden HTTPS, Anwendungscache und die Sicherheit von Cross-Origin-Links überprüft. Außerdem werden Anforderungen für Geolocation-Berechtigungen, anfällige JavaScript-Bibliotheken, veraltete APIs und die Frage, ob Benutzer in Kennwortfelder einfügen können, untersucht. Zu berücksichtigen ist auch, wie Bilder mit dem richtigen Seitenverhältnis angezeigt werden.

SEO

Der SEO-Test ist eher simpel gehalten und ist daher nicht ausreichend, um aussagekräftige Informationen zur SEO der eigenen Seite zu treffen. Google möchte die von Lighthouse durchgeführten SEO-Überprüfungen aber noch erweitern. In der bisherigen Form sucht der Test nach Seiten-Tags und Statuscodes.

Mit der Google Lighthouse Analyse lassen sich also einige interessante Rückschlüsse ziehen vor allem hinsichtlich der Seitenperformance. Durch die Verwendung des Chrome Browsers hat der Nutzer standardmäßig Zugriff auf Lighthouse. Alternativ gibt es auch eine Node Version für die Kommandozeile.[31]

5.1.2 GTmetrix

GTmetrix ist ein kostenloses Tool, mit dem man die Geschwindigkeitsleistung einer Seite mithilfe von Google Page Speed und YSlow analysieren kann. GTMetrix generiert Ergebnisse für die Seite und bietet Empfehlungen zur Behebung eventuell auftretenden Schwächen. Eine Analyse startet immer über die Startseite (https://gtmetrix.com/) doch um möglichst genaue Testergebnisse zu bekommen, gilt es auf ein paar kleine Voreinstellungen zu achten. Bei der Test Location, die nur nach kostenloser Registrierung geändert werden kann, empfiehlt es sich je nach Vorhaben den Standort anpassen. Während es sich zum lokalen Testen empfiehlt einen Testserver auszuwählen, welcher sich in der Nähe des physischen Standorts des Servers der zu testenden Seite befindet, werden für einen globalen Test lieber vier oder fünf Testserver an wichtigen Standorten auf der ganzen Welt ausgewählt. Wird diese Auswahl nicht manuell getroffen findet eine zufällige Wahl des Servers statt. Des Weiteren empfiehlt es sich bei den Tests nicht nur die Homepage zu untersuchen, sondern auch die Unterseiten, um ein möglichst klares Bild zum Stand der eigenen Webseite zu bekommen. Für die kostenlose Nutzung bietet GTMetrix drei Analyse Größen an:

PageSpeed And YSlow:

PageSpeed und YSlow bieten im Allgemeinen den gleichen Service an, es gibt jedoch Unterschiede in ihren Berechnungen. Jeder Dienst analysiert eine Seite anhand einer Reihe von Regeln, von denen er glaubt, dass sie für die Seitengeschwindigkeit und -leistung am relevantesten sind. Viele Regeln überschneiden sich wodurch das Ergebnis ungefähr vergleichbar ist.

Waterfall

Die Wasserfall Ansicht ist ein nützliches Tool um Engpässe in der Geschwindigkeit einer Website zu lokalisieren. Jeder Balken im Wasserfall zeigt alle Schritte für jedes Asset und wie lange sie gedauert haben. [32]

[31] Vgl. Michael Philipps https://www.greengeeks.com/blog/2019/08/15/google-lighthouse-how-you-use-it/, Stand 28.05.2020

[32] Vgl. Daniel Pataki (2020) https://winningwp.com/gtmetrix/, Stand 28.05.2020

5.2 Ergebnisse Lighthouse

Mit dem von Google bereit gestellten Tool Lighthouse, ergaben sich folgende Ergebnisse:

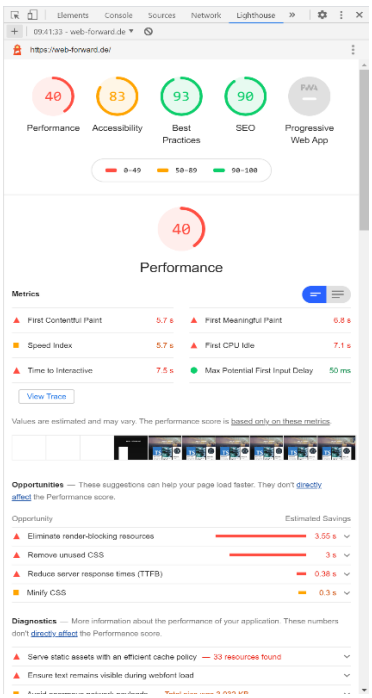

Abbildung 23: Webforward Index mit Lighthouse

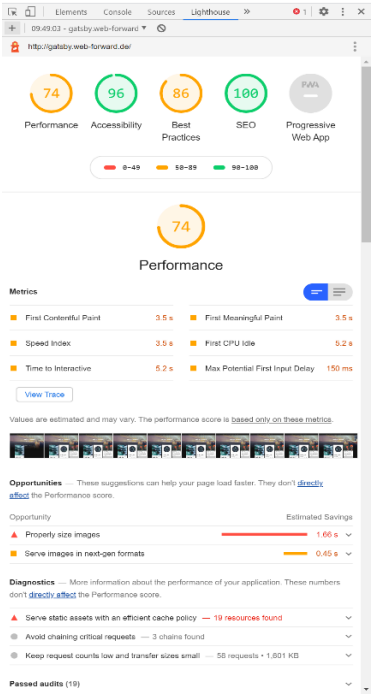

Abbildung 24: GatsbyWebforward Index mit Lighthouse

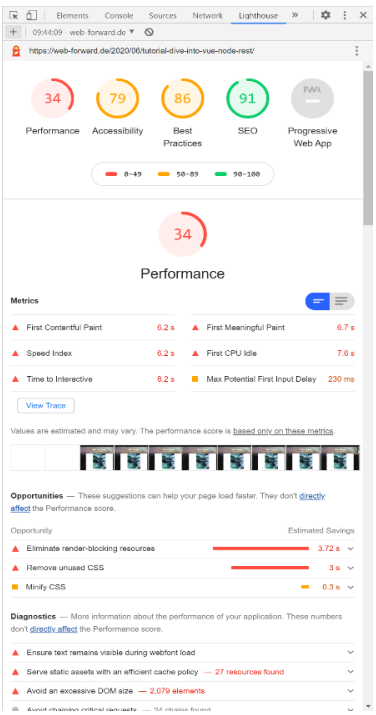

Abbildung 25: Webforward Blog mit Lighthouse

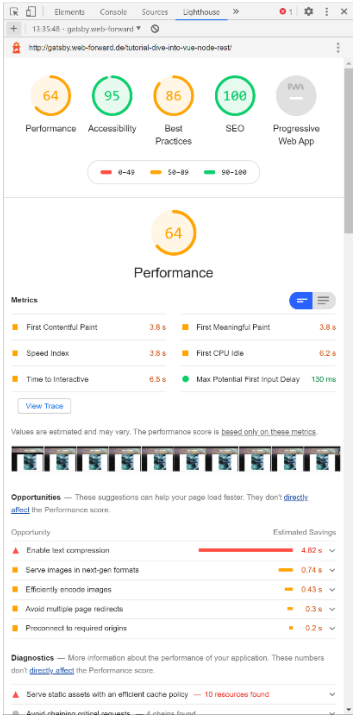

Abbildung 26: GatsbyWebforward Blog mit Lighthouse

5.3 Ergebnisse GTMetrix

Mit GTMetrix ergaben sich folgende Ergebnisse:

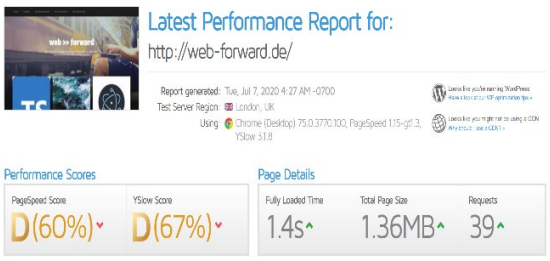

Abbildung 27: Webforward Index mit GTMetrix

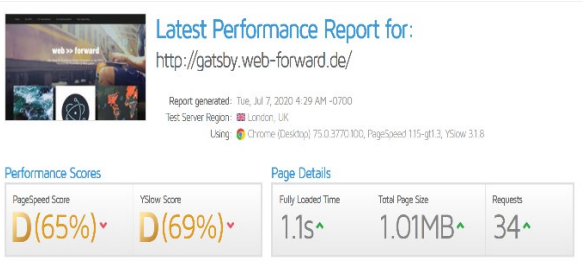

Abbildung 28: GatsbyWebforward Index mit GTMetrix

Abbildung 29: Webforward Blog mit GTMetrix

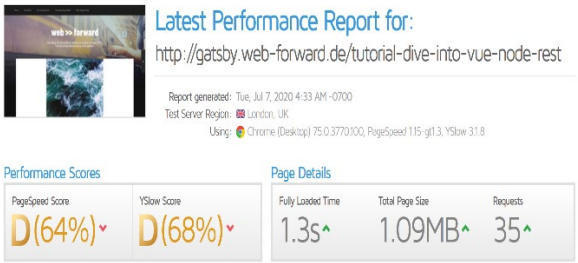

Abbildung 30: GatsbyWebforward Blog mit GTMetrix

5.4 Vergleich der Ergebnisse

Das sich die Seitenladegeschwindigkeit deutlich verringert hat ist schon spürbar, wenn man die beiden Seiten aufruft. Während die Ausgangsseite (https://web-forward.de/) einige Augenblick braucht bis sie geladen ist, ist die mit Gatsby nachgebaute Seite gefühlt unmittelbar erreichbar. Die Ergebnisse der Messungen mit Google Lighthouse und GTMetrix bestätigen dieses Gefühl.

Die Untersuchung der Seiten mit Lighthouse zeigen, dass sich die Seiten Performance knapp verdoppelt hat von den Werten 40 (Index-Seite) und 34 (Blog-Artikel) auf die Werte 74 bzw 64. Wie weiter oben bereits erwähnt wurde sind Werte zwischen 0-49 schlecht (rot dargestellt) während Werte zwischen 50-89 durchschnittlich gut sind (orange dargestellt). Durch die Rückgabe statischer Seiten verbessert sich zudem das SEO einer Seite, sodass ein Webcrawler nun alle vorhandenen Seiten mit samt ihren statischen Inhalten durchlaufen kann. Zudem können innerhalb von Gatsby durch ein Plugin alle für einen Web-Crawler benötigte Dateien hinzugefügt werden. Die Verbesserung von 90/91 auf 100 untermauert

diese Aussage. Jedoch ist das Ganze mit Vorsicht zu genießen, da die getätigten Maßnahmen innerhalb der SEO-Bewertung nicht automatisch zu einer signifikanten Verbesserung der SEO führen. Durch die vorgenommenen Maßnahmen werden nur die Mindestanforderungen erfüllt.

Mit GTMetrix lassen sich ähnlich gute Testergebnisse erzielen. Für die Index Seite verbessert sich der Performance Score durch die Nutzung von Gatsby um 5 Prozent von 60 auf 65, wobei die Fully Loaded Time von 1,4s auf 1,1s sinkt. Beim Testen des Blog Artikels ist der Unterschied in der Fully Loaded time sogar noch deutlicher zu sehen, von 2,5s auf 1,3s. Der Performance Score der Blog-Seite mit Gatsby ist um 3 Prozent besser.

Die Ergebnisse unserer Messungen zeigen, dass durch die statischen Seitengenerierung mit Hilfe von GatsbyJS definitiv Performanceverbesserungen erzielbar sind. Die Testergebnisse müssen aber trotzdem kritisch betrachtet werden, da die Seitenperformance wie einleitend bereits erwähnt wurde, von vielen weiteren Faktoren abhängt, die im Rahmen unserer Messungen nicht berücksichtigt wurden. Mögliche Faktoren sind zum Beispiel Optimierungen in der Umgebung des eingesetzten Content Management Systems oder Optimierungen der Bilder und Minification von JavaScript oder CSS. Zudem wäre interessant zu erfahren, welche Vorteile GatsbyJS bei größeren Ausgangsseiten mit sich bringt. Dadurch, dass die betrachtete Ausgangsseite eine noch überschaubare Anzahl an Post hatte, entsteht beim Server-seitigen Rendern nicht ein zu großer Zeitverlust.

Dennoch kann also festgehalten werden das durch Gatsby die Performance verbessert werden kann, jedoch ist es durch unsere Tests nicht möglich eine genaue Aussage zu treffen in welchen Größen Verhältnissen eine Verbesserung machbar ist.

6. Zusammenfassung

Im Mittelpunkt der Arbeit stand die Performanceverbesserung mittels dem statischen Seitengenerators GatsbyJS anhand eines Fallbeispiels (https://web-forward.de/). Dabei sollten auch die Einsatzmöglichkeiten und Grenzen des Frameworks untersucht werden.

Gatsby ist ein statischer Webseiten-Generator, der mit ReactJS gebaut wurde und von GraphQL unterstützt wird. Mit diesem Tool lassen sich Webseiten bei Änderungen vollständig generieren und auf einem Server oder CDN hochladen. Gatsby holt Daten für die Seite aus einer Vielzahl von Quellen, einschließlich existierender Seiten, API-Aufrufe und Flatfiles über GraphQL und erstellt die statische Seite basierend auf den angegebenen Konfigurationseinstellungen. Nutzer des Frameworks können sich so die Eigenschaften von

statischen Seiten, wie höhere Geschwindigkeiten und zusätzliche Sicherheit, zu Nutze machen.

Der Einstieg in die Entwicklung mit dem Seitengenerator wird durch eine sehr umfangreiche, gut und verständlich gehaltene Dokumentation erleichtert. Zu finden ist diese auf der offiziellen Webseite der Betreiber (https://www.gatsbyjs.org/tutorial/). Dennoch bleibt zu erwähnen, dass es noch an einigem Wissen, das über die Dokumentation hinaus geht, Bedarf. Kenntnisse in JavaScript, React und GraphQL sind eine Grundvoraussetzung, um mit Gatsby arbeiten zu können.

Wie der von uns im Kapitel 5 getätigte Vergleich bestätigt, sind die Performance Vorteile, die sich durch die Nutzung eines statischen Seitengenerators wie GatsbyJs ergeben, nicht von der Hand zu weisen. Dennoch ergeben sich Einschränkungen, die bei der Entscheidung für oder gegen den Einsatz von Statischen Seiten Generatoren bedacht werden müssen. Während es bei serverseitigen Webanwendungen ausreicht, neue Inhalte einfach in die Datenbank einzuspielen, muss bei Statischen Seiten Generatoren die ganze Webseite neu generiert und zum Webserver hochgeladen werden, um neue Inhalte sichtbar zu machen. Für Blogs oder News-Seiten, die nur hin und wieder neuen Content bekommen, ist dies kein großes Problem, zumal der Prozess des Generierens auch weitgehend automatisiert werden kann. Für sehr dynamische Webanwendungen, die ständig neuen Content darstellen sind Statische Seiten Generatoren dagegen nicht besonders geeignet. Als limitierender Faktor ist hier immer die Buildzeit einzubeziehen, die je nach Größe der Webseite auch mal bis zu 15min andauern kann. Da auf dem Server keine aktive Anwendung läuft, können auch keine Nutzereingaben verarbeitet werden. Es ist also nicht ohne weiteres möglich, Inhalte dynamisch von Nutzern anlegen und verändern zu lassen. Ein Online-Editor, wie ihn CMS anbieten oder eine Kommentar-Funktion bei einem Blog ist also nicht ohne Weiteres machbar. Wer auf seiner Webseite viel dynamischen Inhalt darstellen möchte, muss bei der Nutzung von Gatsby neu überdenken wie dieser kontrolliert und ausgeliefert werden soll. Viele Blogs, die auf statischen Seiten Generatoren basieren, binden eine Kommentarfunktion über Drittanbieter wie beispielsweise Disqus[33] ein.

Gatsby hat in einer Vielzahl von Einstellungen Akzeptanz gefunden. Beispielsweise die Homepage von Airbnb's Data Science and Engineering Blog wird von Gatsby betrieben, obwohl die eigentlichen Blogeinträge auf Medium[34] gehostet werden. Auch weitere namenhafte Firmen wie Nike oder Braun setzen auf das Framework. [35]

[33] Disqus ist ein Online-Dienst, der eine zentralisierte Diskussionsplattform für Websites anbietet
[34] Medium ist eine Publishing-Plattform
[35] Vgl. https://www.gatsbyjs.org/showcase, Stand 01.07.2020

Bezogen auf unser Fallbeispiel, dem Blog (https://web-forward.de/), wäre eine Umstellung auf Gatsby durchaus sinnvoll. Da die Seite fast ausschließlich aus statischem Inhalt besteht, die Kommentarfunktion nur wenig bis gar nicht genutzt wird und nur selten neuer Content eingepflegt wird, würde die Seite aus unserer Sicht nur davon profitieren. Da Wordpress weiterhin als Datenquelle genutzt wird (als Headless CMS) müsste sich der Contentersteller nicht mal groß umgewöhnen. Nur der Workflow wäre dann ein etwas anderer. Eine wirkliche 1:1 Vorschau ist schwierig und die Neugenerierung der Webseite benötigt eine gewisse Zeit. Zudem würde es dann Sinn machen den Build- und Deploy-Prozess (Auslieferung an einen Webserver) zu automatisieren.

Literaturverzeichnis

Hauser Benjamin (2007): Erstellen von dynamischen Websites: Einsatz von Webservern & Datenbanken unter Windows CE

Kollmann Tobias, Häsel Matthias (2007): Web 2.0: Trends und Technologien im Kontext der Net Economy

Erlhofer Sebastian (2011), Suchmaschinen Optimierung das umfassende Handbuch, 9. Auflage

Fowler, S. Daniel (2014): How many websites are there in the world?, https://tekeye.uk/computing/how-many-websites-are-there, Stand 13.05.2020

Adobe Dreamwaver Benutzerhandbuch (2017), https://helpx.adobe.com/de/dreamweaver/using/web-applications.html, Stand 16.05.2020

Graack Christoph (2011), https://blog.kompaktdesign.com/webdesign/statisch-vs-dynamisch, Stand 10.05.2020

Sebastian Schürmanns (2020), https://cmsstash.de/empfehlungen/einfuhrung-cms, Stand 23.05.2020

Jakob Nielsen (1993), https://www.nngroup.com/articles/response-times-3-important-limits/, Stand 16.05.2020

Amit Singhal (2010), https://webmasters.googleblog.com/2010/04/using-site-speed-in-web-search-ranking.html, Stand 17.05.2020

Zhiheng Wang and Doantam Phan (2018), https://webmasters.googleblog.com/2018/01/using-page-speed-in-mobile-search.html, Stand 17.05.2020

Zac Gordon (2019), https://javascriptforwp.com/what-is-gatsby-js-and-why-use-it, Stand 18.05.2020

Shaumik Daityari (2020), https://kinsta.com/de/blog/gatsby-wordpress, Stand 20.05.2020

Ibad Ur Rehman (2020), https://www.cloudways.com/blog/use-react-with-wordpress-to-create-headless-cms/#benefitsofheadlesscms, Stand 22.05.2020

Michael Philipps (2020), https://www.greengeeks.com/blog/2019/08/15/google-lighthouse-how-you-use-it/, Stand 28.05.2020

Daniel Pataki (2020), https://winningwp.com/gtmetrix/, Stand 28.05.2020

o.V., https://moz.com/learn/seo/page-speed, Stand 17.05.2020

o.V., https://www.gatsbyjs.org, Stand 13.05.2020

o.V https://www.gatsbyjs.org/showcase, Stand 01.07.2020

o.V., https://www.ionos.de/digitalguide/hosting/cms/headless-cms-was-sind-die-vorteile, Stand 23.05.2020

o.V., https://www.thinkwithgoogle.com/marketing-resources/data-measurement/mobile-page-speed-new-industry-benchmarks, Stand 01.07.2020

o.V., uxplanet.org (2019), https://uxplanet.org/how-page-speed-affects-web-user-experience-83b6d6b1d7d7, Stand: 16.05.2020

o.V., https://trends.builtwith.com/cms, Stand 15.04.2020

Eidesstattliche Erklärung

Ich versichere durch meine Unterschrift, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt, alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen, als solche kenntlich gemacht und mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit

hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Ort, Datum	Unterschrift Student/-in