HOCHSCHULE FLENSBURG

BACHELOR THESIS

VERGLEICH VON UNTERSCHIEDLICHEN HERANGEHENSWEISEN AN CSS ANHAND EINER INDIVIDUELL GESTALTETEN WEBSEITE

DORIEN GRÖNWALD

Matrikel-Nummer: 630688
Studiengang: Medieninformatik
Betreuer*in und Erstbewerter*in: Tobias Hiep
Zweitbewerter*in: Uwe Zimmermann
Ausgabedatum: 06.08.2021
Abgabedatum: 06.10.2021

ABSTRACT

Die Entwicklung einer Webseite fordert viel Zeit und setzt ein fundiertes Vorwissen über die genutzten Mittel voraus. Aus diesem Grund sind Webentwickelnde fortlaufend auf der Suche nach neuen Technologien, Programmen oder Erweiterungen, die eine Webseitenentwicklung vereinfachen und optimieren, ohne dabei Abstriche im Endergebnis zu erhalten. Eine mögliche Lösung dafür stellen CSS-Frameworks wie Bootstrap oder TailwindCSS dar.

Das Ziel dieser Thesis ist es herauszufinden, **welche Vor- und Nachteile die unterschiedlichen CSS-Frameworks im Vergleich zu handgeschriebenen CSS bei der Umsetzung einer individuell gestalteten Webseite bieten**. Um diese Fragestellung beantworten zu können, wurde ein praktischer Vergleich von Bootstrap und TailwindCSS sowie handgeschriebenen CSS durchgeführt.

Der Vergleich zeigte, dass Bootstrap und TailwindCSS aufgrund ihrer unterschiedlichen Ansätze individuelle Vor- und Nachteile aufweisen, sodass diese nicht auf alle CSS-Frameworks verallgemeinert werden können. Allerdings wurde durch die Erkenntnisse verdeutlicht, dass die Umsetzung mit TailwindCSS ein besseres und qualitativ hochwertigeres Endergebnis liefern konnte. Zusammenfassend kann jedoch gesagt werden, dass eine Umsetzung mit handgeschriebenen CSS aufgrund der hohen Einflussnahme des Entwickelnden deutlich individueller und persönlicher ist und explizite Designvorgaben leichter umsetzbar sind.

GLIEDERUNG

ANMERKUNG ZUR GESTALTUNG

Die in Kapitel *3 Konzeption* entwickelte Gestaltung wurde mithilfe des Programms Adobe XD erstellt. Dieses Programm bietet die Funktion mittels eines Links auf die erstellten Benutzeroberflächen zugreifen zu können. Dies ist der Link zu der individuellen Gestaltung:

https://xd.adobe.com/view/4ef6ca0f-5b49-4564-b6fd-4297a1e7ca35-1ba5/

ANMERKUNG ZUR UMSETZUNG

Im Laufe dieser Thesis wurden insgesamt drei verschiedene Umsetzungen der gleichen Gestaltung erzeugt. Die Ergebnisse können unter bestimmten Domains abgerufen werden und der entsprechende Quellcode dazu liegt bei dem Versionsverwaltungssystem GitHub.

Umsetzung mit CSS:
Domain: https://groenwald-thesis-css.netlify.app/
GitHub Repository: https://github.com/doriengr/thesis-css

Umsetzung mit TailwindCSS:
Domain: https://groenwald-thesis-tailwindcss.netlify.app/
GitHub Repository: https://github.com/doriengr/thesis-tailwindcss

Umsetzung mit Bootstrap:
Domain: https://groenwald-thesis-bootstrap.netlify.app/
GitHub Repository: https://github.com/doriengr/thesis-bootstrap

1 EINLEITUNG

„Be aware that the development tool landscape is ever-shifting. Tools come and go in rapid-fire fashion, with the whole development community jumping on one framework bandwagon, then moving to the next new thing.“ [1]

Mit diesen knappen Worten beschreibt die Autorin und Webentwicklerin Jennifer Robbins die Schnelllebigkeit der Webentwicklung. Es werden immer neue Werkzeuge, Programme oder Erweiterungen entwickelt, die die Entwicklung einer Webseite schneller, effizienter und erfolgversprechender gestalten sollen. Viele dieser Hilfsmittel haben sich erfolgreich in den Alltag eines Entwickelnden integriert und gelten heute als erprobte Methode.

Die herkömmliche Herangehensweise eine Webseite zu entwickeln, entspricht einem Zusammenspiel aus der Auszeichnungssprache *HyperText Markup Language* (**HTML**) und handgeschriebenen *Cascading Style Sheets* (**CSS**). [2] Diese beiden Werkzeuge haben sich schon mehrfach bewiesen und werden von Anfang an stetig weiterentwickelt, um den Bedürfnissen der Entwickelnden und der Zeit gerecht zu werden. Auffällig ist jedoch, dass die Entwicklung einer Webseite und deren Optimierung im Browser viel Zeit fordert und fundiertes Vorwissen über die genutzten Mittel voraussetzt. Demnach sind Webentwickelnde fortlaufend auf der Suche nach neuen Hilfsmitteln und neuen Herangehensweisen, die die Arbeitsweisen grundlegend verbessern und ein qualitativ hochwertiges und exklusives Ergebnis erzielen können.

Eine mögliche Lösung dafür stellen CSS-Frameworks wie Bootstrap oder TailwindCSS dar. Diese CSS-Frameworks basieren in ihrer Funktionsweise auf CSS, nutzen aber zwei unterschiedliche Ansätze, die eine verbesserte und schnellere Arbeitsweise versprechen. Diese Ansätze basieren grundsätzlich auf vordefinierten Inhalten, die diese Frameworks mitliefern. Fraglich ist, ob die gestalterische Freiheit von Entwickelnden durch diese vordefinierten Inhalte eingeschränkt wird und ob die behaupteten Vorteile der CSS-Frameworks der Realität entsprechen.

Daraus erschließen sich die Leitfrage und das Ziel dieser Arbeit. Demnach soll herausgefunden werden, **welche Vor- und Nachteile die unterschiedlichen CSS-Frameworks im Vergleich zu handgeschriebenen CSS bei der Umsetzung einer Webseite mit einer individuellen Gestaltung bieten**.

1 Robbins, Jennifer Niederst: Learning Web Design: A Beginner’s Guide to Html, Css, Javascript, and Web Graphics, 5. Aufl., Sebastopol, USA: O’Reilly Media, 2018, S. 571
2 Vgl. Hahn, Martin: Webdesign: Das neue Handbuch zur Webgestaltung, 3. Aufl., Bonn, Deutschland: Rheinwerk Design, 2020, S. 31.

Das Ziel der Thesis ist es daher herauszufinden, ob die Umsetzung einer individuell gestalteten Webseite mit CSS-Frameworks grundsätzlich möglich ist. Dafür soll während der Umsetzung ermittelt werden, welche Vor- und Nachteile die jeweiligen Herangehensweisen bieten.

Um diese Fragestellung beantworten zu können, wird ein praktischer Vergleich durchgeführt. Wie bereits in der Leitfrage formuliert, wird eine individuelle Designvorlage einer Webseite sowohl mit der herkömmlichen Entwicklungsweise CSS als auch mit den CSS-Frameworks TailwindCSS und Bootstrap umgesetzt.

Zunächst wird die theoretische Grundlage anhand bestehender fachlicher Literatur untersucht, sodass ein grundlegendes Verständnis für die Thematik aufgebaut wird. Dafür werden die drei Bereiche der Frontend-Entwicklung erklärt und der Komponenten basierte Ansatz sowie der Utility-Klassen basierte Ansatz hinter den CSS-Frameworks erörtert.

Im Anschluss daran werden die etablierten Designmuster innerhalb des heutigen Webdesigns aufgezeigt. Diese werden innerhalb der Erstellung des individuellen Screendesigns gezielt gebrochen. Gleichzeitig werden alle Designentscheidungen erklärt und beschrieben, sodass diese nachvollziehbar werden.

Darauf basierend werden im vierten Kapitel alle drei Umsetzungen der individuellen Gestaltung schriftlich begleitet. Dabei werden die unterschiedlichen Herangehensweisen anhand mehrerer Kriterien aus verschiedenen Bereichen der Webentwicklung tiefergreifend beleuchtet. Die Ergebnisse dieser Kriterien sind die Basis für den im fünften Kapitel erstellten Vergleich. Hier werden die Erkenntnisse gegenübergestellt und die daraus gezogenen Ergebnisse werden im Fazit abschließend formuliert und reflektiert.

2 TECHNISCHE GRUNDLAGEN

Um die Stärken und Schwächen der unterschiedlichen Herangehensweisen an die Entwicklung einer Webseite gegenüberstellen zu können, muss als Erstes thematisiert werden, wie die Entwicklung einer Webseite funktioniert und welche Werkzeuge dafür verwendet werden können. Aus diesem Grund befasst sich dieses Kapitel mit den technischen Grundlagen und den Hilfsmitteln, die für eine Webseitenentwicklung benötigt werden. Daraus soll ein grundlegendes Verständnis entstehen, welcher Entwicklungsansatz herkömmlich für die Umsetzung einer Website genutzt und angewendet wird. Darauf basierend werden im nächsten Schritt zwei neuere Herangehensweisen untersucht, indem deren Ansätze und möglicher Nutzen beleuchtet werden.

2.1 HYPERTEXT MARKUP LANGUAGE

Im Jahr 1990 entwickelte Tim Berners-Lee die „HyperText Markup Language". Auf der *HyperText Markup Language* – kurz **HTML** – basiert das World Wide Web, da es den Baustein für den grundlegenden Transport von strukturierten Inhalten legt. Statt HTML können auch andere Markup-Technologien verwendet werden, wie z. B. SVG oder XML. Allerdings ist HTML heutzutage die am meisten verbreitete Markup-Sprache. [1]

Nach der Veröffentlichung von HTML wurde in den darauffolgenden Jahren deren erste Spezifikation veröffentlicht. [2] Eine Spezifikation beschreibt die Eigenschaften einer Technologie und definiert, wie diese korrekt umgesetzt werden. Sie dient als Leitfaden für Entwickelnde und wird unter anderem vom *World Wide Web Consortium* (**W3C**) und der *Web Hypertext Application Technology Working Group* (**WHATWG**) festgelegt. [3] Im Laufe der Jahre wurden die Auszeichnungsmöglichkeiten von HTML immer weiter ausgebaut. Im Jahr 2019 wurde zwischen der W3C und der WHATWG ein Abkommen geschlossen, dass sie sich nur noch auf eine einzige HTML-Version konzentrieren werden. [4] Diese HTML-Version, auch bekannt als HTML5, wird als „Living Standard" bezeichnet. HTML5 wird stetig weiterentwickelt und neue

1 Vgl. What is CSS? - Learn web development | MDN: in: MDN Web Docs, 02.07.2021, https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS#what_is_css_for (abgerufen am 06.08.2021).
2 Vgl. CERN: Tags used in HTML, in: info.cern.ch, o. D., http://info.cern.ch/hypertext/WWW/MarkUp/Tags.html (abgerufen am 13.08.2021).
3 Vgl. Spezifikation – SELFHTML-Wiki: in: SELFHTML-Wiki, o. D., https://wiki.selfhtml.org/wiki/Spezifikation (abgerufen am 07.08.2021).
4 Vgl. WHATWG: HTML Standard, Edition for Web Developers, in: WHATWG, o. D., https://html.spec.whatwg.org/dev/introduction.html#history-2 (abgerufen am 13.08.2021).

Funktionen können hinzugefügt werden. Der Standard und die Spezifikation der HTML-Version bleiben jedoch gleich. [5]

2.1.1 BEDEUTUNG HINTER DEM BEGRIFF HTML

Um die Funktionsweise von HTML besser nachvollziehen zu können, hilft ein genauerer Blick auf die Hintergründe der Namensgebung. Der Begriff „HyperText“ beschreibt die Funktion mithilfe von Links auf andere Seiten im Web oder auf eigene Unterseiten verweisen zu können. [6] „Markup“ hingegen bezieht sich auf die Funktionsweise, die Inhalte einer Webseite für die Anzeige im Browser je nach Inhaltstyp korrekt zu kommentieren. [7] Da dies das Hauptaugenmerk der HyperText Markup Language ist und die Auszeichnungen für die Generierung einer Webseite unbedingt benötigt werden, soll darauf etwas tiefgreifender eingegangen werden.

2.1.2 SEMANTIK

HTML wird aufgrund ihrer Fähigkeit, Inhaltstypen von Webseiten kommentieren bzw. beschreiben zu können [8], auch als Auszeichnungssprache bezeichnet. [9] Ein HTML-Dokument enthält neben dem eigentlichen Inhalt einer Webseite sogenannte HTML-Tags bzw. -Elemente. Diese lassen sich an den spitzen Klammern (<,>) erkennen und beschreiben die Inhalte, geben ihnen eine Struktur und eine Bedeutung. Das Besondere daran ist, dass der Browser diese HTML-Elemente nicht mit anzeigt, sondern lediglich ihren Inhalt ausliest und optisch kennzeichnet. Somit bleibt die eigentliche Auszeichnung der HTML-Tags verborgen. [10]
Bei der Anwendung von Auszeichnungen ist die strukturelle Bedeutung – auch Semantik genannt – zu beachten, um einen korrekten Quellcode zu erhalten und um eine korrekte Browserauslesung sicherzustellen. Dadurch können zum Beispiel Ladezeiten und die Erreichbarkeit von Webseiten verbessert werden. [11] Folglich ist die Semantik ein wichtiger Bestandteil der Markup-Funktion von HTML. Laut Martin Hahn wird Semantik wie folgt definiert: „Semantik (auch Bedeutungslehre genannt) beschäftigt sich mit der Beziehung zwischen einem Zeichen und der Bedeutung dieses Zeichens. Übertragen auf HTML bedeutet es, dass die Elemente entsprechend ihrer inhaltlichen Bedeutung eingesetzt werden.“ [12] Um Semantik anhand eines

5 Vgl. WHATWG: FAQ — WHATWG, in: WHATWG, o. D., https://whatwg.org/faq#living-standard (abgerufen am 13.08.2021).
6 Vgl. HTML: HyperText Markup Language | MDN: in: MDN Web Docs, 28.07.2021, https://developer.mozilla.org/de/docs/Web/HTML (abgerufen am 06.08.2021).
7 Vgl. HTML: HyperText Markup Language | MDN, 2021.
8 Vgl. Laborenz, Kai: CSS: Das umfassende Handbuch. Inkl. Responsive Webdesign, Animationen, Sass, 3. Aufl., Bonn, Deutschland: Rheinwerk Computing, 2016, S. 25.
9 Vgl. Hahn, 2020, S. 31.
10 Vgl. Robbins, 2018, S. 24.
11 Vgl. Hahn, 2020, S. 33.
12 Hahn, 2020, S. 31.

Beispiels zu verdeutlichen, kann das <h1>-Element genannt werden. Es kennzeichnet die wichtigste Überschrift einer Seite und sollte somit nur einmal pro HTML-Dokument verwendet werden. [13]

2.2 CASCADING STYLE SHEETS

Nachdem im Jahr 1990 HTML erfunden wurde, stellte 1994 der norwegische Informatiker Håkon W. Lie die Stylesheet-Sprache Cascading Style Sheets vor. [14] Seitdem spielen die *Cascading Style Sheets* – kurz **CSS** – eine große Rolle in der Webentwicklung, da mittels CSS die Gestaltung einer Webseite möglich gemacht wurde. Mit der Veröffentlichung von HTML 4.0 im April 1998 war es nun auch möglich, die Stylesheet-Sprache in ein HTML-Dokument einzubinden. [15] Seitdem lassen sich die optischen Kennzeichnungen der HTML-Elemente nach Belieben erweitern und manipulieren. Kai Laborenz vergleicht CSS in seinem Buch „CSS – Das umfassende Handbuch“ als „Kleidung einer Webseite [...] – ohne Stylesheets sieht man nur den nackten unformatierten Inhalt. Wie bei Kleidung kann eine Website mit verschiedenen Stylesheets völlig unterschiedlich aussehen [...] – auch wenn das HTML-Gerüst und die Inhalte gleich bleiben. “ [16]

2.2.1 EINBINDUNG VON STYLESHEETS

Die Struktur und die Gestaltung einer Webseite werden strikt voneinander getrennt. [17] Die Struktur wird durch das HTML definiert, wohingegen die Gestaltung der Website in mindestens einem Stylesheet festgehalten wird. Die Stylesheets werden im <head>-Bereich des HTML-Dokuments verlinkt. [18]

2.2.2 USER-AGENT-STYLESHEETS

Ausgezeichnete HTML-Elemente werden im Browser automatisch ihrer Bedeutung entsprechend angezeigt, auch wenn kein eigenes Stylesheet verlinkt wurde. Dadurch wird vom Browser sichergestellt, dass bei einer Abwesenheit eines Stylesheets ein Aussehen basierend auf der Semantik generiert wird und der Inhalt weiterhin lesbar bleibt. [19] Aufgrund dessen werden beispielsweise Überschriften größer dargestellt als normaler Fließtext

13 Vgl. Hahn, 2020, S. 31.
14 Vgl. Wium Lie, Håkon: Cascading HTML Style Sheets -- A Proposal, in: World Wide Web Consortium (W3C), o. D., https://www.w3.org/People/howcome/p/cascade.html (abgerufen am 07.08.2021).
15 Vgl. HTML 4.0 Specification: in: World Wide Web Consortium (W3C), o. D., https://www.w3.org/TR/1998/REC-html40-19980424/ (abgerufen am 07.08.2021).
16 Laborenz, 2016, S. 17.
17 Vgl. Hahn, 2020, S. 31.
18 Vgl. Laborenz, 2016, S. 42.
19 Vgl. Hahn, 2020, S. 32.

oder Links werden unterstrichen und sind blau. [20] Die vom Browser generierten Grundgestaltungen werden User-Agent Stylesheets genannt. Sie unterscheiden sich von Browser zu Browser, können aber durch eigenes CSS überschrieben werden. [21]

2.2.3 FUNKTIONSWEISE VON CSS

Die Funktionsweise von CSS basiert darauf, Regeln zu definieren, die eine Ansammlung von Befehlen zugewiesen bekommen. [22] Kai Laborenz beschreibt die Funktionsweise wie folgt: „Eine CSS-Anweisung besteht immer aus zwei Teilen – dem Selektor und der Deklaration. Der Selektor beschreibt, was (also welches Element, [...]) formatiert werden soll, und die Deklaration bestimmt, wie dies geschehen soll (also z. B. welche Farbe das Element haben soll).“ [23]

Dabei gibt es eine Vielzahl an unterschiedlichen Selektoren: Element-, Klassen-, ID-, Attribut- und Universalselektoren, Kombinatoren und Pseudoklassen bzw. -elemente.

Kaskadierung:
Nachdem nun thematisiert wurde, dass unterschiedliche HTML-Elemente mithilfe von unterschiedlichen Selektoren angesprochen werden können, bleibt die Frage, wann welche Deklarationen greifen. Dies ist besonders dann von großer Bedeutung, wenn mehrere Deklarationen für das gleiche Element an unterschiedlichen Orten gleichzeitig vorhanden sind. Dadurch beeinflussen sich die Deklarationen untereinander. Diese Funktion wird als Kaskade bezeichnet und macht CSS sowohl mächtiger als auch komplexer. [24]

Die MDN Web Docs von Mozilla Firefox gibt drei Hauptquellen an, die die Kaskade von CSS bilden:

1. User-Agent-Stylesheets: Deklarationen, die durch den Browser definiert werden
2. Author-Stylesheets: Von Entwickelnden definierte Deklarationen in einem oder mehreren verlinkten Stylesheets
3. User-Stylesheets: Die Anwendenden einer Webseite können definieren, welche Deklarationen überschrieben werden sollen

20 Vgl. MDN Web Docs: What is CSS? - Learn web development | MDN, in: MDN Web Docs, 02.07.2021, https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS (abgerufen am 25.08.2021).
21 Vgl. Introducing the CSS Cascade - CSS: Cascading Style Sheets | MDN: in: MDN Web Docs, 11.06.2021, https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade#origin_of_css_declarations (abgerufen am 07.08.2021).
22 Vgl. What is CSS? - Learn web development | MDN: in: MDN Web Docs, 02.07.2021, https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS#css_syntax (abgerufen am 07.08.2021).
23 Laborenz, 2016, S.23.
24 Vgl. MDN Web Docs: Introducing the CSS Cascade - CSS: Cascading Style Sheets | MDN, in: MDN Web Docs, 13.08.2021, https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade (abgerufen am 13.08.2021).

Schlussendlich filtert die Kaskade alle Deklarationen für jedes definierte HTML-Element heraus und untersucht mithilfe der Spezifität, welche Deklaration als Erstes und welche als Letztes angewendet wird. [25]

Spezifität:
Jede Deklaration weist eine vordefinierte Spezifität auf. Diese bestimmt, wie hochwertig die jeweilige Deklaration vom Browser interpretiert wird.

Für die Spezifität gibt es klare, festgelegte Regeln. Allgemein kann gesagt werden: Je spezifischer eine Anweisung, desto höherwertiger ist diese. Bei Anweisungen gleichen Wertes gelten die später definierten als höherwertig. [26] Es gibt jedoch auch Mittel, wodurch bestimmte CSS-Anweisungen eine höhere Wertigkeit erlangen können. In solchen Fällen wird die `!important`-Regel angewendet. Mit dieser Flagge lassen sich alle anderen definierten Stildeklarationen überschreiben. In den MDN Web Docs wird jedoch davon abgeraten diese Regel zu verwenden, „[...] da es das Debuggen erschwert, weil die normale Kaskadierung der Stylesheets dadurch unterbrochen wird." [27] Als „Debugging" wird die Fehlersuche in einem Quellcode bezeichnet. [28]

MDN Web Docs liefert zudem eine Liste, die die Selektoren nach ihrer Spezifität ordnet. Die Liste ist in absteigender Reihenfolge angeordnet: [29]

1. Inline-Styles
2. ID-Selektoren
3. Pseudoklassen
4. Attributselektoren
5. Klassenselektoren
6. Elementselektoren
7. Universalselektoren

Weitere Funktionen:
Neben den unterschiedlichen Selektoren hat CSS noch weitere Funktionen wie die Umsetzung von Media Queries oder Animationen. Diese werden jedoch bei Bedarf im weiteren Verlauf der Arbeit näher untersucht bzw. als bereits bekannt vorausgesetzt.

Zudem gibt es die sogenannten CSS-Eigenschaften, die bestimmte Gestaltungsangaben definieren. Eine vollständige Liste aller CSS-Eigenschaften ist auf der Webseite des W3C einsehbar.

25 Vgl. MDN, 2021.
26 Vgl. Laborenz, 2016, S.79.
27 Spezifität - CSS | MDN: in: MDN Web Docs, 30.07.2021, https://developer.mozilla.org/de/docs/Web/CSS/Specificity#die_!important_ausnahme (abgerufen am 06.08.2021).
28 Vgl. Dudenredaktion: Debugging, in: Duden, o. D., https://www.duden.de/rechtschreibung/Debugging (abgerufen am 13.08.2021).
29 Vgl. MDN, 2021

Wie schon im Abschnitt *2.1 Hypertext Markup Language* angesprochen, entwickelt W3C nicht nur die Spezifikationen für HTML, sondern ebenfalls für CSS. [30] Die aktuelle Version von CSS ist CSS3, wobei auch hier im Laufe der Zeit weitere Funktionen und CSS-Eigenschaften hinzugefügt werden können. [31]

2.3 JAVASCRIPT

Der dritte Bestandteil bei der Umsetzung einer Webseite ist neben HTML und CSS häufig *JavaScript* – kurz **JS** – und wird zur Ausführung bestimmter Funktionalitäten eingesetzt. Diese drei Komponenten des Webdesigns bzw. der Webentwicklung werden mit dem Begriff *Frontend* zusammengefasst. Sie bauen aufeinander auf und ergänzen sich. [32]

JavaScript findet auf Webseiten als Skriptsprache eine Verwendung, indem sie häufig für Animationen, für die Verbesserung der User Experience oder für Reaktionen auf Interaktionen des Anwendenden eingesetzt wird. [33] Jennifer Robbins beschreibt in ihrem Buch „Learning Web Design" JavaScript wie folgt: „It is a client-side scripting language, which means it runs on the user's machine and not on the server, as other web programming languages such as PHP and Ruby do." [34] Dies hat zur Folge, dass die Ausführung von JS von den Fähigkeiten und Einstellungen des Browsers abhängig ist. Die Anwendenden sind sogar dazu in der Lage, das Ausführen jeglicher JS-Dateien im Browser komplett zu unterbinden. [35]

Auf jedes HTML-Element einer Webseite kann JS mithilfe des *Document Object Model's* (**DOM**) zugreifen. Mit sogenannten EventListenern kann das Verhalten der Anwendenden abgefragt und darauf entsprechend reagiert werden, indem z. B. bestimmte CSS-Deklarationen hinzugefügt oder gelöscht werden. Beispielsweise können somit Navigationen ausgeklappt bzw. eingefahren werden. Ein weiterer Anwendungsbereich von JavaScript ist das Generieren von Feedback für den Anwendenden, indem es Anfragen des Servers verarbeitet und eine daraus resultierende Darstellung erstellt, ohne den Browser neu laden zu müssen. Ein Beispiel hierfür ist ein fehlgeschlagener Login. [36]

Der selbst erstellte JS-Quellcode wird in einer JS-Datei festgehalten. Diese

30 Vgl. About W3C: in: World Wide Web Consortium (W3C), o. D., https://www.w3.org/Consortium/ (abgerufen am 06.08.2021).
31 Vgl. Hahn, 2020, S. 32.
32 Vgl. Hahn, 2020, S. 31.
33 Vgl. Hahn, 2020, S. 33.
34 Robbins, Jennifer Niederst: Learning Web Design: A Beginner's Guide to Html, Css, Javascript, and Web Graphics, 5. Aufl., Sebastopol, USA: O'Reilly Media, 2018, S. 593
35 Vgl. Robbins, 2018, S. 593
36 Vgl. Robbins, 2018, S. 595

wird mittels eines entsprechenden HTML-Elemens am Ende des `<body>`-Elements verlinkt. [37]

2.4 CSS-FRAMEWORKS

Neben der herkömmlichen Art und Weise, Webseiten mit reinem, handgeschriebenen CSS umzusetzen, gibt es die Möglichkeit, CSS-Frameworks dafür zu verwenden. CSS-Frameworks sollen Webentwickelnden dabei behilflich sein, Projekte einfach und mit minimalem Codeaufwand umsetzen zu können. Dafür werden vordefinierte Konzepte, Module und standardisierte Kriterien verwendet. [38] Diese neuen Umsetzungsmöglichkeiten basieren auf den Funktionen von CSS, bieten jedoch zusätzlich neue Wege für eine Webseitenumsetzung und -gestaltung.

Die ersten CSS-Frameworks sind in der Mitte der 2000er Jahre veröffentlicht worden. Beispiele dafür sind CSS-Frameworks wie Blueprint, YAML oder YUI Grids. Diese CSS-Frameworks lieferten beispielsweise vordefinierte Rastersysteme. [39]

Unter den heutzutage meistgenutzten CSS-Frameworks fällt unter anderem das Framework Bootstrap. Es wurde von zwei Entwicklern des Mikrobloggingdienstes Twitter entwickelt, um die Umsetzung dieses Projektes zu vereinfachen. Im Jahre 2011 wurde der Quellcode dazu als „Open Source“ veröffentlicht. Dies bedeutete, dass der Quellcode frei zugänglich und öffentlich einsehbar wurde und damit auch für andere Entwickelnde nutzbar wurde. [40]

Im Jahr 2017 wurde ein weiteres CSS-Framework namens TailwindCSS veröffentlicht. Seitdem bekommt auch dieses CSS-Framework immer mehr Aufmerksamkeit. [41] Weitere nennenswerte Beispiele werden in Abschnitt *2.4.1.3 Beispiele von Komponenten basierten CSS-Frameworks* und *2.4.2.3 Beispiele von Utility-Klassen basierten Frameworks* aufgezählt.

Bootstrap und TailwindCSS repräsentieren zwei unterschiedliche Ansätze, wie ein CSS-Framework aufgebaut und eingesetzt werden kann. Diese beiden Herangehensweisen sollen näher betrachtet werden, indem ihre unterschiedlichen Ansätze und ihr möglicher Nutzen vorgestellt werden.

37 Vgl. Robbins, 2018, S. 597
38 Vgl. Shenoy, Aravind/Anirudh Prabhu: CSS Framework Alternatives: Explore Five Lightweight Alternatives to Bootstrap and Foundation with Project Examples, 2018, doi:10.1007/978-1-4842-3399-3, S. 2.
39 Vgl. Blueprint: A CSS Framework | Spend your time innovating, not replicating: in: BlueprintCSS, o. D., http://blueprintcss.org/ (abgerufen am 07.08.2021).
40 Vgl. Dudenredaktion: Open Source Sofware, in: Duden, o. D., https://www.duden.de/rechtschreibung/Open_Source_Software (abgerufen am 13.08.2021).
41 Vgl. tailwindlabs: Release v0.1.0 · tailwindlabs/tailwindcss, in: GitHub, o. D., https://github.com/tailwindlabs/tailwindcss/releases/tag/v0.1.0 (abgerufen am 19.08.2021).

Dabei ist Bootstrap ein Repräsentant für Komponenten basierte CSS-Frameworks und TailwindCSS für Utility-Klassen basierte CSS-Frameworks.

2.4.1 KOMPONENTEN BASIERTE CSS-FRAMEWORKS

Wie die Bezeichnung schon vermuten lässt, basiert dieser Framework-Ansatz auf die Nutzung von vordefinierten Komponenten. [42]

2.4.1.1 DEFINITION VON KOMPONENTEN

Tom Barker beschreibt in seinem Buch „High Performance Responsive Design" Komponenten basierte CSS-Frameworks wie folgt: „In these frameworks, there will generally be predefined CSS that describes the styling of a module such as a button or a grid, or even complex UI elements such as accordions and sliders and guided navigation." [43]

Die von Barker beschriebenen vordefinierten Module werden auch Komponenten genannt. Sie sind von einem CSS-Framework mitgelieferte CSS-Klassen, deren Namen ihre Funktion beschreiben. [44] Eine CSS-Klasse mit der Bezeichnung `btn` wäre beispielsweise eine Komponente für alle Buttons. Diese CSS-Klassen haben meistens mehrere vorgeschriebene CSS-Eigenschaften, die die Gestaltung der Komponente bestimmen. [45]

```
<!-- HTML -->
<button type="button" class="btn">Text</button>

// CSS
.button {
  display: block;
  background-color: $color-extra-light;
  color: $color-green;
  border: 1px solid $color-brown;
  padding: $spacing-10 $spacing-15;

  &:hover {
    background-color: $color-brown;
    color: $color-extra-light;
  }
}
```

Abb. 2.1: Button-Komponente, Quelle: Eigene Darstellung

Die Klassennamen der Komponenten sind demnach die Klassenselektoren. Dadurch kann die Komponente auf das eigene HTML-Element angewendet werden, indem man die Komponentenbezeichnung im `class`-Attribut des HTML-Elements angibt (vgl. Abb. 2.1). [46]

2.4.1.2 NUTZEN VON KOMPONENTEN

Nachdem nun aufgezeigt wurde, was sich genau unter einer Komponente verbirgt, soll nun erörtert werden, welchen Nutzen sie hat.

42 Vgl. Hahn, 2020, S. 312.
43 Barker, Tom: High Performance Responsive Design: Building Faster Sites Across Devices, Sebastopol, USA: O'Reilly Media, 2014, S. 129.
44 Vgl. Rappin, Noel: Modern CSS with Tailwind: Flexible Styling without the Fuss, Raleigh, USA: The Pragmatic Programmers, LLC, 2021, S. IX.
45 Vgl. Rappin, 2021, S. IX.
46 Vgl. Barker, 2014, S. 129.

Komponenten beschreiben hauptsächlich Module, die häufig eine Verwendung finden, wie z. B. das schon genannte Exempel eines Buttons. Durch die Verwendung eines Komponenten basierten CSS-Frameworks können die Entwickelnden auf ein Repertoire an vordefinierten Codeschnipseln zugreifen, ohne dafür selbst Code generieren zu müssen. Die verschiedenen Komponenten können mehrfach verwendet werden und auf einer Webseite beliebig die Position wechseln, sodass ein modulares Design entsteht. Die bereits vordefinierten Komponenten können beispielsweise als ein Frontend-Styleguide angesehen werden. Darin wären die Gestaltungen und Formatierungen bestimmter HTML-Elemente schon definiert und zusammengefasst. [47] Dadurch lassen sich komplette Webseitenstrukturen einfach zusammen klicken und kopieren, ohne viel eigenes CSS schreiben zu müssen. Dies spart nicht nur Zeit in der Entwicklung, was besonders für Webentwicklungsunternehmen von Vorteil sein könnte, sondern schmälert auch die benötigten Vorkenntnisse und somit die benötigte Kompetenz der Entwickelnden. Gleichzeitig könnten Komponenten basierte CSS-Frameworks als leichte Einstiegsmöglichkeit gesehen werden, um mit der Thematik der Frontend-Entwicklung vertrauter zu werden (vgl. *4.4.14 Vorkenntnisse*).

Die Verwendung von CSS-Komponenten minimiert jedoch nicht nur den Zeitaufwand der Entwickelnden während der Generierung von eigenen CSS-Deklarationen, sondern auch beim Testen des Codes. Martin Hahn fasst dies wie folgt zusammen: „Bei gut ausgearbeiteten Frameworks sind diese [Komponenten] schon auf Browserkompatibilität getestet und mögliche Browser-Bugs behoben.“ [48]

2.4.1.3 BEISPIELE VON KOMPONENTEN BASIERTEN CSS-FRAMEWORKS

Es gibt viele Beispiele von CSS-Frameworks, die auf vordefinierte Komponenten basieren. Die meistverwendeten Komponenten basierte CSS-Frameworks lassen sich anhand einer Statistik von „State of CSS“ aus dem Jahr 2020 ablesen. Bei der Befragung haben insgesamt 11.492 Menschen aus 102 Ländern teilgenommen. [49] 86% der Befragten gaben an, dass sie das Framework Bootstrap verwenden. [50] Damit ist dieses das meistverwendete Komponenten basierte CSS-Framework. An zweiter Stelle steht MaterializeCSS, da 33% der Befragten angegeben haben, dieses Framework zu verwenden. MateralizeCSS basiert auf der Philosophie des Google Material Designs und stellt eine *User-Interface*-Komponenten-Bibliothek zur Verfügung. [51] Als **UI**-Komponenten werden alle Elemente bezeichnet, die zur

47 Vgl. Hahn, 2020, S. 144, S.145.
48 Hahn, 2020, S. 312.
49 Vgl. The State of CSS 2020: Demographics: in: State of CSS 2020, o. D., https://2020.stateofcss.com/en-us/demographics/ (abgerufen am 06.08.2021).
50 Vgl. The State of CSS 2020: CSS Frameworks
51 Vgl. Shenoy/Prabhu, 2018, S. 7.

Interaktion mit einer Webseite oder Applikation beitragen. [52] Die Befragten gaben ebenfalls an, dass 28% von ihnen Foundation, 19% von ihnen Bulma und 17% von ihnen Semantik UI verwenden.

Abb. 2.2: Nutzung CSS-Frameworks, Quelle: https://2020.stateofcss.com/en-US/technologies/css-frameworks/ (abgerufen am 15.09.2021)

2.4.2 UTILITY-KLASSEN BASIERTE CSS-FRAMEWORKS

Als Nächstes werden Utility-Klassen basierte CSS-Frameworks vorgestellt, die anstelle von vordefinierten Komponenten Utility-Klassen mitliefern. Dieser Ansatz hat erst in den letzten Jahren Fuß gefasst. [53]

2.4.2.1 DEFINITION VON UTILITY-KLASSEN

Ryan Lanciaux beschreibt in seinem Buch „Modern Frontend Architecture“ die Verwendung von Utility Klassen so: „By using a utility-first approach, developers build many small CSS classes or styling functions that can be used repeatedly throughout a codebase. Instead of applying styles directly to a component, we reference a number of utility class names to define how it is styled.“ [54] Im Vergleich zu Komponenten werden die Deklarationen nicht mehr in CSS-Klassen gebündelt, sondern einzeln dem bestimmten HTML-Element zugewiesen. Dies geschieht mittels vordefinierten CSS-Klassen, die jeweils eine einzelne CSS-Eigenschaft repräsentieren. Diese vordefinierten

52 Vgl. Hahn, 2020, S. 209.
53 Vgl. Lanciaux, Ryan: Modern Front-end Architecture: Optimize Your Front-end Development with Components, Storybook, and Mise en Place Philosophy, 1st ed., Ann Arbor, USA: Apress, 2021, S. 67.
54 Vgl. Lanciaux, 2021, S. 67.

CSS-Klassen werden in diesem Zusammenhang Utility-Klassen genannt. [55]

Die Umsetzung eines Buttons mithilfe von Utility-Klassen könnte wie folgt aussehen:

```
<!-- HTML -->
<button type="button" class="block bg-opacity-0 border border-dark py-3 px-4 mt-10 mx-auto transition-all duration-300 ease-in-out hover:bg-dark hover:text-beige-lightest">Text</button>
```

Abb. 2.3: Styling eines Buttons mittels Utilities, Quelle: Eigene Darstellung

2.4.2.2 NUTZEN VON UTILITY-KLASSEN

In diesem Abschnitt soll nun aufgezeigt werden, welchen Nutzen Utlilty-Klassen bieten sollen.

Da Utility-Klassen dem HTML-Element direkt zugewiesen werden und sie in einem gut ausgebauten CSS-Framework alle nötigen CSS-Eigenschaften repräsentieren können, müssen die Entwickelnden auch hier kein eigenes CSS mehr schreiben. Die Aufgabe des Entwickelnden ist es nun zu wissen, welche Utility-Klasse welcher CSS-Eigenschaft entspricht. Dies hat besonders dann einen Vorteil, wenn eine bereits bestehende Webseite, die mit einem Utility-Klassen basierten CSS-Framework umgesetzt worden ist, mit weiteren Gestaltungselementen ausgebaut werden soll. Durch die Wiederverwendbarkeit der Utility-Klassen muss im Gegensatz zu handgeschriebenen CSS kein neuer Quellcode geschrieben werden, da die Entwickelnden auf einen bereits bestehenden Quellcode zurückgreifen können. [56] Dadurch wird verhindert, dass der Quellcode einer Webseite wächst, sobald neue Inhaltselemente hinzugefügt werden. Das Zuweisen einzelner CSS-Eigenschaften bietet auch einen Vorteil bei der Fehlersuche im Code. Durch die direkte Anwendung von einzelnen Eigenschaften auf nur einem HTML-Element wird schnell deutlich, welche Auswirkung die jeweilige Utility-Klasse hat. [57]

Wie bei der Verwendung von vordefinierten Komponenten bieten vordefinierte Utility-Klassen die Erschaffung eines modularen Designs. Durch die Limitierung der CSS-Eigenschaften auf nur die Wichtigsten wird eine durchlaufende Konsistenz erschaffen. Dadurch kann ein eigenes Design System entstehen (vgl. *4.3.4 Vordefiniertes Gestaltungssystem*). [58]

55 Vgl. Rappin, 2021, S. IX.
56 Vgl. Utility-First - Tailwind CSS: in: TailwindCSS, o. D., https://tailwindcss.com/docs/utility-first (abgerufen am 06.08.2021).
57 Vgl. Rappin, 2021, S. X.
58 Vgl. Rappin, 2021, S. 17.

2.4.2.3 BEISPIELE VON UTILITY-KLASSEN BASIERTEN FRAMEWORKS

Die Statistik von „State of CSS 2020“ zeigt, dass viele der verwendeten CSS-Frameworks weiterhin Komponenten basierte CSS-Frameworks sind. Unter den 13 aufgelisteten Frameworks befinden sich lediglich zwei reine Utility-Klassen basierte CSS-Frameworks. Darunter zählen TailwindCSS und Tachyons. 26% der Befragten gaben an, dass sie TailwindCSS als Framework verwenden. Dies ist somit das meistverwendete Utility-Klassen basierte CSS-Framework (vgl. Abb. 2.2). Mit der Angabe, dass 6% der Befragten Tachyons verwenden, landet dieses Framework, laut der Befragung, auf Platz zwei der meistverwendeten Utility-Klassen basierten CSS-Frameworks. Tachyons wird als Framework vermarktet, mit dessen Hilfe sich komplette responsive Webseiten ausbauen lassen, mit so wenig CSS-Code, wie möglich. [59] Der Begriff responsiv wird von Duden wie folgt beschrieben: „antwortend; eine Reaktion zeigend oder darstellend“. Somit wird unter responsivem Verhalten das automatische Anpassen einer Webseitengestaltung an die Breite und Höhe des Bildschirmfensters verstanden. [60]

Weitere weniger bekannte Utility-Klassen basierte CSS-Frameworks sind unter anderem Shed.css, Basscss und Expressive CSS.

59 Vgl. @mrmrs: TACHYONS - Css Toolkit, in: Tachyons, o. D., https://tachyons.io/ (abgerufen am 10.08.2021).
60 Vgl. Dudenredaktion: Responsiv, in: Duden, o. D., https://www.duden.de/rechtschreibung/responsiv (abgerufen am 25.09.2021).

3 KONZEPTION

Die unterschiedlichen Herangehensweisen an CSS sollen anhand der Umsetzung einer individuell gestalteten Webseite miteinander verglichen werden. Um eine individuelle Gestaltung kreieren zu können, muss vorher abgesteckt werden, welche Rahmenbedingungen innerhalb des Webdesigns als etablierte Designmuster gelten. Um dies herausfinden zu können, wird anhand bestehender Literatur festgelegt, wie der grundlegende Aufbau einer Webseite aussieht. Diese Analyse dient als Grundlage dafür, bestimmen zu können, welche Parameter für eine individuelle Gestaltung gebrochen werden müssen.

Im darauffolgenden Abschnitt *3.2 Gestaltung* wird eine Thematik für die Webseite festgelegt, ein Moodboard erstellt und auf Grundlage dessen Gestaltungsparameter, wie Farben und Typografie, ausgewählt. Zudem wird beschrieben, wie die Inhalte der Webseite angeordnet werden sollen.

3.1 ETABLIERTE DESIGNMUSTER IM WEBDESIGN

„Natürlich sieht jede Seite – zumindest teilweise – anders aus, aber es gibt eine Reihe von Gemeinsamkeiten, die sich auf (fast) jeder Webseite wiederfinden. Diese Gemeinsamkeiten sorgen dafür, dass Webseiten einfacher und angenehmer zu bedienen sind.“ [1] So deutet Martin Hahn in seinem Buch über Webdesign an, dass gewisse Ähnlichkeiten im Aufbau von Webseiten ein bekanntes Phänomen sind. Diese Gemeinsamkeiten werden in diesem Abschnitt genauer untersucht.

3.1.1 ANATOMIE EINER WEBSEITE

Im Verlauf der letzten Jahre hat sich im Webdesign eine Reihe von Gestaltungskonventionen festgesetzt. Diese Konventionen beruhen auf den Erfahrungen und Erwartungen der Anwendenden. [2]

Diese Erwartungen zeugen von ähnlichen Grundstrukturen, die auf vielen Webseiten zu finden sind. Diese Grundstrukturen können auch als Grundgerüst einer Webseite verstanden werden, da die HTML-Struktur einer Webseite immer die gleichen Elemente aufweist. [3] Die Elemente werden stark vereinfacht in Abbildung 3.1 dargestellt und entsprechen den jeweiligen HTML-Tags `<header>`, `<main>` und `<footer>`.

1 Hahn, 2020, S. 271.
2 Vgl. Hahn, 2020, S. 278.
3 Vgl. Hahn, 2020, S. 272.

Header:
Als „Header“ wird der obere Bereich einer Webseite bezeichnet, in der sich häufig die Hauptnavigation und das Logo der Firma oder des Unternehmens befinden. [4] Da das Logo das visuelle Merkmal eines Unternehmens ist, Assoziationen hervorruft und einen starken Wiedererkennungswert hat, soll es prominent im Header dargestellt werden. Der Leserichtung entsprechend wird es demnach oft links platziert. [5] Die Hauptnavigation hingegen wird oft rechts neben oder linksbündig unter dem Logo gesetzt. Sie bietet eine Interaktionsmöglichkeit für den Anwendenden und dient als Strukturübersicht oder Orientierungspunkt. [6]

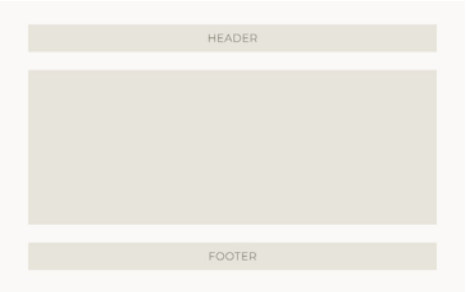


Abb. 3.1: Grundgerüst einer Webseite, Quelle: In Anlehnung an Hahn, 2020, S. 271.

Das untere Beispiel zeigt die häufige Anordnung im Header anhand einer realen Umsetzung:

Abb. 3.2: Header der Webseite „mehr als lernen“, Quelle: https://mehralslernen.org/ (abgerufen am 10.08.2021)

Main:
Im HTML-Tag `<main>` werden alle weiteren Inhalte einer Webseite zusammengefasst. Dieser Bereich wird auch oft Inhaltsbereich genannt und wird häufig unter dem Header platziert. Dabei werden wichtige Inhalte weiter oben und ausführlichere Details weiter unten angeordnet. [7]

Footer:
Der „Footer“ – auch Fußzeile genannt – bildet den „inhaltlichen und gestalterischen Abschluss“. [8] In der Fußzeile finden ergänzende Inhalte, weitere Navigationsmöglichkeiten oder die Metanavigation einen Platz. Sowohl der Footer als auch der Header sind oft auf allen Unterseiten einer Webseite identisch. [9]

3.1.2 HÄUFIG VERWENDETE LAYOUTS UND RASTER

Neben den Ähnlichkeiten in der Anatomie gibt es auch Gemeinsamkeiten bei der Anordnung einzelner Elemente. Als Grundlage dafür können einheitliche Raster dienen. Sie helfen bei der Bewahrung einer Struktur, die den Blicklauf des Anwendenden lenkt. Auch hier gibt es ein häufig verwendetes Layout, das

4 Vgl. Hahn, 2020, S. 272.
5 Vgl. Hahn, 2020, S. 272, S.273.
6 Vgl. Hahn, 2020, S.274.
7 Vgl. Hahn, 2020, S. 274.
8 Hahn, 2020, S. 276.
9 Vgl. Hahn, 2020, S. 276.

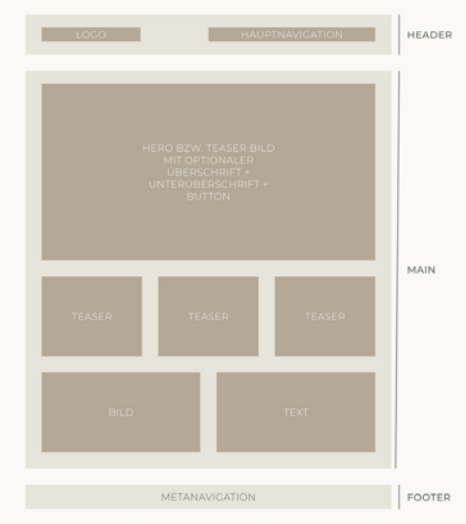


Abb. 3.3: Häufig verwendetes Layout
Quelle: In Anlehnung an Hahn, 2020, S. 278.

auf vielen unterschiedlichen Webseiten wieder zu finden ist (vgl. Abb. 3.3). [10] Auf dieses Layout wird im Folgenden weiter eingegangen.

Wie im Abschnitt *3.1.1 Anatomie einer Webseite* bereits angesprochen, werden die Inhaltstypen im Inhaltsbereich nach ihrer Wichtigkeit sortiert. Unter der Kopfzeile befindet sich der beim Öffnen einer Seite direkt sichtbare Bereich. [11] Der dort platzierte Inhalt ist somit der erste Blickfang und erlangt dadurch viel Aufmerksamkeit. Er dient zudem als einleitendes Element und soll einen Vorgeschmack auf die Thematik geben. [12] Dieser sogenannte Hero-Bereich beginnt oft mit einem großen Foto oder einer Illustration. (vgl. Abb. 3.3). [13]

Abbildung 3.4 zeigt anhand der Webseite von BMW einen exemplarischen Einsatz von einem Hero-Foto.

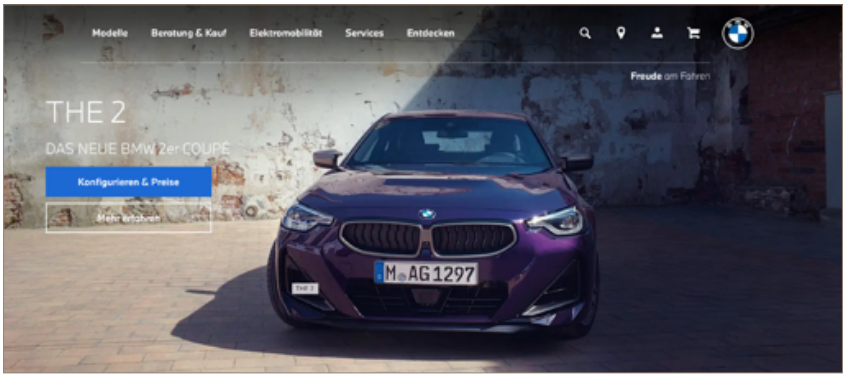


Abb. 3.4: Hero auf Webseite von BMW,
Quelle: https://www.bmw.de/de/home.html (abgerufen am 10.08.2021)

Unter dem einleitenden Bereich werden detailreichere Inhalte dargestellt. Wenn mehrere Inhalte des gleichen Typs angezeigt werden sollen, wird gern das „Card-Design“ angewendet. Dabei werden die gleichartigen Inhalte jeweils von einem Container umschlossen, deren Optik oft an „Karten“ erinnern lassen. [14] Zudem liegt ihnen meistens ein offensichtliches Raster zugrunde. [15] Auch die Teaserblöcke in Abbildung 3.3 sind dem „Card-Design“ nachempfunden und sind an einem dreispaltigen Raster angeordnet.

10 Vgl. Hahn, 2020, S. 277.
11 Vgl. Hahn, 2020, S. 291.
12 Vgl. Hahn, 2020, S. 293.
13 Vgl. Hahn, 2020, S. 274.
14 Vgl. Hahn, 2020, S. 287, S. 288.
15 Vgl. Hahn, 2020, S. 288.

Diese Gestaltung wird gern angewendet, da es sich leicht responsiv umsetzen lässt. Mithilfe des CSS-Grid-Systems lässt sich die Spaltenanzahl je nach Bildschirmgröße verändern, sodass die Karten auf mobilen Endgeräten untereinander und bei breiteren Bildschirmen nebeneinander oder in einem mehrspaltigen Raster angeordnet werden können. [16] Ein Anwendungsbeispiel des „Card-Designs“ wird in Abbildung 3.5 dargestellt. Neben der Anordnung in einem mehrspaltigen Raster, können Inhalte auch einfach nur nebeneinander angeordnet werden (vgl. Abb. 3.6). Dies kann auch mithilfe des CSS-Modells Flexbox umgesetzt werden. [17]

Zusammengefasst kann gesagt werden, dass diese Konventionen im Aufbau einer Webseite zu alltäglichen Layouts führen und somit auch zu einer ähnlichen Gestaltung vieler Webseiten. [18] Es kann zu positiven Effekten führen, da die Erwartungen der Anwendenden erfüllt werden, allerdings geht damit auch ein Teil der Individualität und der persönlichen Atmosphäre verloren.

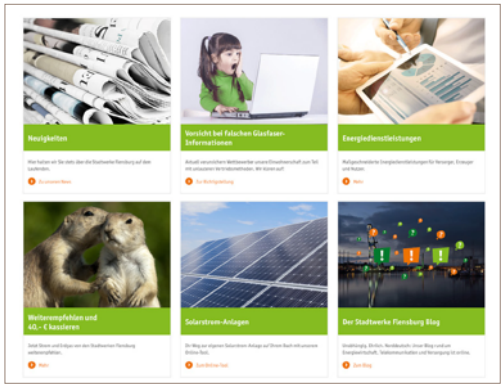


Abb. 3.5 (links): „Card-Design“ Stadtwerke Flensburg, Quelle: https://www.stadtwerke-flensburg.de/#/acquisition (abgerufen am 14.08.2021)

Abb. 3.6 (rechts): Nebeneinander angeordneter Text mit Bild, Quelle: https://www.paypal.com/de/home (abgerufen am 14.08.2021)

3.2 GESTALTUNG

Zunächst wird die Thematik der Webseite festgelegt, damit darauf basierend ein Moodboard erstellt werden kann. Anhand dieses Moodboards werden Gestaltungsangaben wie Farben, Typografie und ein Logo ausgewählt.

3.2.1 THEMATIK & MOODBOARD

Die zu gestaltende Webseite erfüllt keinen realen Zweck, sondern dient rein als Gestaltungsvorlage für die unterschiedlichen Umsetzungen mit den drei verschiedenen Herangehensweisen an CSS. Warum ist es dann wichtig, eine Thematik zu definieren und nicht nur generierten Platzhaltertext zu verwenden?

16 Vgl. Hahn, 2020, S. 287.
17 Vgl. Flexbox - Lerne Webentwicklung | MDN: in: MDN Web Docs, 01.08.2021, https://developer.mozilla.org/de/docs/Learn/CSS/CSS_layout/Flexbox (abgerufen am 01.08.2021).
18 Vgl. Hahn, 2020, S. 291.

Der/die Anwendende nimmt beim Öffnen einer Webseite als Erstes die allgemeine Atmosphäre und Ausstrahlung wahr. Die Gestaltung einer Webseite hat jedoch nicht nur den Zweck, einen ersten guten Eindruck zu verleihen, sondern soll auch bewusst und unterbewusst eine Botschaft kommunizieren. [19] Demnach muss die zu vermittelnde Botschaft klar sein, bevor eine ansprechende Gestaltung erstellt werden kann. Dies ist nicht mit inhaltslosen Platzhaltertexten möglich.

Um Innovation ausstrahlen zu können, müssen gewisse Konventionen im Webdesign gebrochen werden, wie eine spezielle Anordnung oder auffällige Farbkombinationen. Diese Form der Innovation passt gestalterisch jedoch nicht zu konservativen Themen, sodass sie beispielsweise für die Umsetzung einer Website für einen Steuerberater wegfällt. Um einen modernen Stil in der Gestaltung einnehmen zu können, sollten die verwendeten Inhalte ein aktuelles Thema widerspiegeln.

Ein aktuelles Thema, das in den letzten Jahren einen gesellschaftlichen Aufschwung erlebt hat, ist die vegane Ernährung. Auf der Webseite von Statista wird beschreiben, dass die Anzahl der Menschen, die in Deutschland wohnen und sich selbst als Veganer betiteln, laut der Allensbacher Markt- und Werbeträger-Analyse um 180.000 Personen im Vergleich zum Vorjahr gestiegen ist. [20] Da dies auch in der Gastronomie immer deutlicher berücksichtigt wird [21], ist dies eine Thematik, die sowohl eine gewisse Aktualität mit sich bringt als auch genug Freiraum bietet, eine ansprechende, moderne Gestaltung zu finden. Somit wurde als fiktives Thema eine Corporate Webseite für ein veganes Café in der Stadt Flensburg ausgewählt.

Eine Corporate Webseite dient zur Selbstdarstellung und vertritt den Onlinewerbeauftritt für z. B. ein Unternehmen. [22] In diesem Fall tritt die Corporate Webseite in Form eines OnePagers auf. Als OnePager wird eine einzelne Seite ohne weitere Unterseiten bezeichnet, sodass alle Inhalte kompakt auf einer Seite zur Verfügung gestellt werden. Dadurch wird verhindert, dass die Inhalte zu umfangreich werden.[23]

Nachdem nun die Thematik feststeht, kann ein Moodboard erstellt werden. Dabei können verschiedene Visualisierungen durch grafische Elemente wie Farben, Bilder, Illustrationen und Typografien ausgetestet werden. [24] Abbildung 3.7. zeigt das fertige Moodboard.

19 Vgl. Hahn, 2020, S. 29.
20 Vgl. Statista: Umfrage in Deutschland zur Anzahl der Veganer bis 2020, in: Statista, 16.07.2021, Zitiert nach de.statista.com, https://de.statista.com/statistik/daten/studie/445155/umfrage/umfrage-in-deutschland-zur-anzahl-der-veganer/ (abgerufen am 07.08.2021).
21 Vgl. Statista: Anzahl veganer Gastronomiebetriebe in Deutschland bis 2021, in: Statista, 01.07.2021, https://de.statista.com/statistik/daten/studie/381076/umfrage/anzahl-veganer-gastronomiebetriebe-in-deutschland/ (abgerufen am 07.08.2021).
22 Vgl. Hahn, 2020, S. 647.
23 Vgl. Hahn, 2020, S. 298.
24 Vgl. Hahn, 2020, S. 98.

Abb. 3.7: Moodboard, Quelle: Eigene Darstellung, verwendete Bilder sind von unsplash.com

3.2.2 DESIGNELEMENTE

Basierend auf dem Moodboard und den festgelegten Adjektiven bzw. Attributen, kann nun ein Screendesign erstellt werden.

3.2.2.1 FARBEN

„Ein Webdesign, das als erfolgreiches Marketing-Instrument agieren will, muss sensibel für die kulturellen und instinktiven Wirkungen und Bedeutungen der einzelnen grafischen Elemente sein. Denn Farbe hat auf den Menschen eine enorme (unbewusste) emotionale Wirkung und beeinflusst unser Denken und Handeln.“ [25] So schreibt Marin Hahn der Farbwirkung eine sehr bedeutende Rolle zu. Um demnach eine passende Farbwelt auswählen zu können, müssen Assoziationen, Sättigung, Farbton und Farbtemperatur berücksichtigt werden. [26]

Für das Design des veganen Cafés wurde eine natürliche Farbwelt mit neutralen Farben ausgewählt. Da im Veganismus auf tierische Produkte verzichtet wird, greifen Veganer zu pflanzlichen und natürlichen Alternativen. Um diese Natürlichkeit und Naturverbundenheit auch in der Farbwelt aufgreifen zu können, wurden unterschiedliche Erdtöne als Farben der Marke ausgewählt. Da Brauntöne oft mit Begriffen, wie Erde, Holz, Natur, Wärme und Bodenständigkeit in Verbindung gebracht werden, [27] entwickelt sich durch diese Farbumgebung eine passende Ausstrahlung, die potenziellen Kunden das Herausstellungsmerkmal des Cafés– die veganen Produkte – verdeutlichen sollen.

Insgesamt wurden sechs unterschiedliche neutrale Farbtöne für die Gestaltung ausgewählt. Diese sind in Abbildung 3.8 aufgelistet. Der erste Farbton links ist der hellste Farbton der Farbpalette und dient als Hintergrundfarbe. Dagegen ist rechts der dunkelste Farbton – die Schriftfarbe – abgebildet. Dieser Kontrast zwischen Text- und Hintergrundfarbe wird auch als Text-Hintergrund-Kontrast bezeichnet und ist ausschlaggebend für die Leserlichkeit der Texte. Der am meisten eingesetzte Text-Hintergrund-Kontrast ist der Kontrast zwischen Schwarz und Weiß, da dies der stärkste Hell-Dunkel-Kontrast ist. [28] Um sich davon abzugrenzen und eine persönlichere Gestaltung zu erhalten, wurden hier zwei andere kontrastreiche Farbtöne ausgewählt, die besser in die ausgewählte, natürliche Farbwelt passen.

25 Hahn, 2020, S. 358.
26 Vgl. Hahn, 2020, S. 359.
27 Vgl. Hahn, 2020, S. 382.
28 Vgl. Hahn, 2020, S. 457.

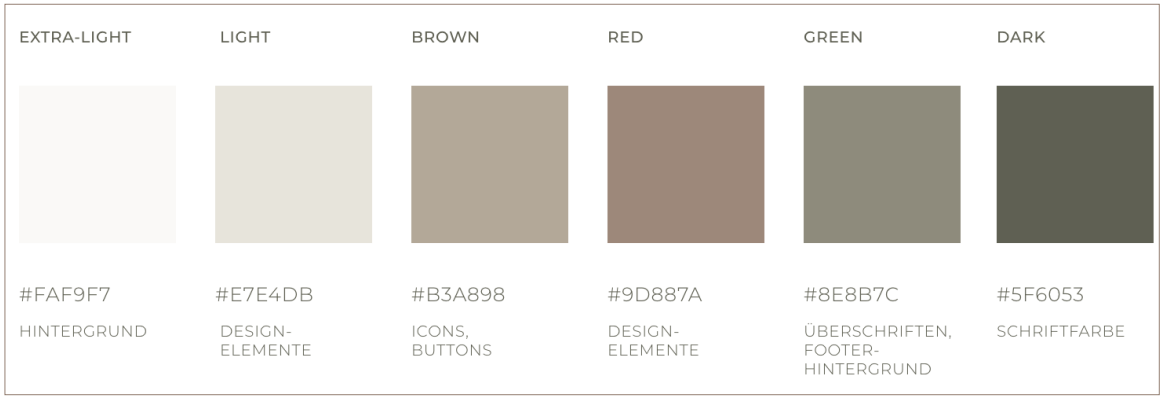


Abb. 3.8: Ausgewählte Farbwelt, Quelle: Eigene Darstellung

Für unauffällige Designelemente wurde der zweite Farbton von links verwendet, da dieser sich nicht zu sehr von der Hintergrundfarbe absetzt. Die drei weiteren Farben sind ebenfalls erdige Töne, die allerdings verschiedene Farbnuancen aufweisen. Dies sorgt für Abwechslung und nötige Abgrenzung zu anderen Elementen.

Damit diese Töne dezent, zurückhaltend und elegant wirken, ist ihre Sättigung heruntergesetzt. Die Gestaltung zielt durch das Zusammenspiel aus ungesättigten Farben darauf ab, Eleganz, Minimalismus und Modernität auszustrahlen.

Bei der Auswahl der Farben ist jedoch nicht nur ihr Zusammenspiel wichtig, sondern auch der Kontrast. Dieser stellt die Leserlichkeit der Texte sicher. Besonders Anwendende mit einer Farb- oder Sehschwäche sind auf ein gutes Kontrastverhältnis angewiesen. Für die Ermittlung der Stärke von Farbkontrasten wurden die *Web Content Accessibility Guidelines* (**WCAG**) 2.0 entwickelt. Diese legen fest, wie groß der Kontrast sein sollte, um die Leserlichkeit auch unter schwierigen Bedingungen weiterhin gewährleisten zu können. [29] Es gibt zudem Online-Tools, die die Farbkontraste testen und auflisten. Eines dieser Tools ist das „Contrast-Grid“ von eightshapes.com, mithilfe dessen sich Kombinationen von Vorder- und Hintergrundfarben, auf Basis der WCAG 2.0, testen lassen. [30]

Das Ergebnis ist in Abbildung 3.9 zu sehen. Die dort aufgezeigten Beschränkungen sind beim Erstellen des Designs zu berücksichtigen.

29 Vgl. Hahn, 2020, S. 458.
30 Vgl. EightShapes: EightShapes Contrast Grid, in: EightShapes, o. D., https://contrast-grid.eightshapes.com/?background-colors=&foreground-colors=%23faf9f7%2C%20Extra-Light%0D%0A%23e7e4db%2C%20Light%0D%0A%23b3a898%2C%20Brown%0D%0A%239D887A%2C%20Red%0D%0A%238E8B7C%2C%20Green%0D%0A%235f6053%2C%20Dark%0D%0A%0D%0A&es-color-form__tile-size=compact (abgerufen am 12.08.2021).

Abb. 3.9: Contrast-Grid
Quelle: https://contrast-grid.eightshapes.com/?background-colors=&foreground-colors=%23faf9f7%2C%20Extra-Light%0D%0A%23e7e4db%2C%20Light%0D%0A%23b3a898%2C%20Brown%0D%0A%239D887A%2C%20Red%0D%0A%238E8B7C%2C%20Green%0D%0A%235f6053%2C%20Dark%0D%0A%0D%0A&es-color-form__tile-size=compact

3.2.2.2 TYPOGRAFIE

Nicht nur individuelle Farben sind für eine Corporate Webseite von Bedeutung, sondern auch die Textgestaltung und die dazu passenden typografischen Mittel.

Als Schrift für lange Fließtexte auf einer Webseite sollte eine leserliche Schrift ausgewählt werden. Dabei sollte die emotionale Wirkung bei der Schriftauswahl jedoch nicht außen vor bleiben. [31] Allgemeine Konventionen besagen, dass auf Bildschirmen mit kleiner Schriftgröße serifenlose Schriften besser lesbar sind. Demnach sollte im Web auf serifenlose Schriften zurückgegriffen werden. Sobald die Schriftgrößen allerdings größer werden, ist kaum ein Unterschied erkennbar. [32]

Da die Wirkung von Serifenschriften als anmutig und klassisch bezeichnet wird [33] und dies zum gewünschten modernen „Look & Feel“ der Gestaltung passt, wurde trotz der Konvention eine Serifenschrift für Fließtexte ausgewählt. Zudem dient dies als Abgrenzung zu den etablierten Typografie-Konventionen. Dementsprechend wurde `Adobe Caslon Pro` als Fließtextschrift ausgewählt. Dies ist eine Schrift mit einer langen Historie und sie wird als

31 Vgl. Hahn, 2020, S. 474
32 Vgl. Hahn, 2020, S. 475
33 Vgl. Hahn, 2020, S. 475

stabil, sicher und zuverlässig beschrieben. [34] Um den Textblöcken jedoch etwas mehr Leichtigkeit zu verleihen, wurde ein großzügiger Zeilenabstand gewählt. [35]

Die Überschriften hingegen sollen sich mittels einer größeren Schriftgröße und einer auffälligen, lauten Schriftart abheben, um die Aufmerksamkeit auf sich ziehen. [36] Dafür wurde die Schriftart `IvyPresto Display` ausgewählt, die aufgrund der leichten Strichstärke, dem hohen Kontrast zwischen den Breiten der Serifen und ihrer hohen x-Höhe (Höhe der Kleinbuchstaben) gleichzeitig modern, elegant und auffallend wirkt.

Als dritte Schrift, die für Unterüberschriften und andere Hervorhebungen verwendet werden kann, wird die serifenlose Schrift `Montserrat` eingesetzt. Sie bringt sowohl Modernität als auch Klarheit mit sich und hebt sich somit von den anderen Schriften ab. [37] Durch die Erhöhung der Laufweite und ihren vermehrten Einsatz in Großbuchstaben wirkt ihre Ausstrahlung edler und großzügiger. [38]

Sowohl Adobe Caslon Pro, IvyPresto Display als auch Montserrat sind Schriften, die von Adobe Fonts für persönlichen und kommerziellen Gebrauch zur Verfügung gestellt werden. [39]

IvyPresto Display Thin	ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜ abcdefghijklmnopqrstuvwxyzäöü 1234567890
Adobe Caslon Pro Regular	ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜ abcdefghijklmnopqrstuvwxyzäöü 1234567890
Montserrat Medium	ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜ abcdefghijklmnopqrstuvwxyzäöü 1234567890

34 Vgl. Fonts.com: Adobe Caslon Font Family Typeface Story, in: Fonts.com, o. D., https://www.fonts.com/de/font/adobe/adobe-caslon/story (abgerufen am 12.08.2021).
35 Vgl. Hahn, 2020, S. 454
36 Vgl. Hahn, 2020, S. 476
37 Vgl. Hahn, 2020, S. 427
38 Vgl. Hahn, 2020, S. 467
39 Vgl. Adobe Fonts.com: IvyPresto Display from Ivy Foundry, in: Adobe Fonts.com, o. D., https://fonts.adobe.com/fonts/ivypresto-display#licensing-section (abgerufen am 12.08.2021), Vgl. Adobe Fonts.com: Adobe Caslon from Adobe Originals, in: Adobe Fonts.com, o. D., https://fonts.adobe.com/fonts/adobe-caslon#licensing-section (abgerufen am 12.08.2021), Vgl. Adobe Fonts: Montserrat from Google, in: Adobe Fonts, o. D., https://fonts.adobe.com/fonts/montserrat#licensing-section (abgerufen am 19.08.2021).

Abb. 3.10: Logo, Quelle: Eigene Darstellung

3.2.2.3 LOGO

Das Logo ist ein visueller Repräsentant der Marke und dient als Wiedererkennungsmerkmal. Für das Logo des veganen Cafés wurde eine Mischung aus Wort- und Bildmarke gewählt. Als Wortmarke dient der Name des fiktiven Cafés: „Flavour Earth“. „Flavour“ soll dabei den Aspekt verdeutlichen, dass es sich bei der Marke um ein Unternehmen handelt, das leckere Nahrungsmittel anbietet. „Earth“ stellt dagegen die Verbindung zur Naturverbundenheit und zum Veganismus her.

Da der Schriftzug der Wortmarke nicht zu aufdringlich und extravagant sein sollte, sondern eher von subtiler Anmutung, ist die Montserrat als Schriftart für die Wortmarke ausgewählt worden. [40] Um die Montserrat noch edler und großzügiger wirken zu lassen, wurde die Laufweite der Schrift erhöht.

Da in dem fiktiven Café hauptsächlich Kaffee, veganes Gebäck, Brot und Brötchen verkauft werden, ist als Symbol für die Bildmarke eine Getreideähre ausgewählt worden. Dadurch entsteht für den Betrachtenden direkt eine Assoziation mit Nahrungsmitteln, die aus Getreide herstellt werden können. Somit können potenzielle Kunden direkt erahnen, welche Nahrungsmittel angeboten werden könnten.

Damit die Gestaltung und die Atmosphäre der Marke auch im Logo wiedergegeben werden, wird auch hier die gleiche definierte Farbwelt verwendet, die in Abschnitt *3.2.2.1 Farben* erläutert wurde.

3.2.3 RASTER

Wie im Abschnitt *3.1.2 Häufig verwendete Layouts und Raster* schon angesprochen, hilft ein einheitlich verwendetes Raster dabei, die Inhalte auf einer Seite kohärent zueinander anzuordnen. [41] Deswegen sollte auch bei einer individuellen Gestaltung auf ein Raster bzw. ein Layout zurückgegriffen werden.

Auf Basis dessen ist ein achtspaltiges Raster mit individuellen Spaltenbreiten entstanden. Um ein responsives Verhalten gewährleisten zu können, muss sich dieses flexibel an breite und schmale Bildschirme anpassen können. Deswegen werden die Spaltenbreiten des Rasters nicht in Pixel angegeben, was einer festen Breitenangabe entsprechen würde, sondern in

40 Vgl. Wäger, Markus: ABC des Grafikdesigns: Grafik und Gestaltung visuell erklärt, 1. Aufl., Bonn, Deutschland: Rheinwerk Design, 2020, S. 302
41 Vgl. Hahn, 2020, S. 277

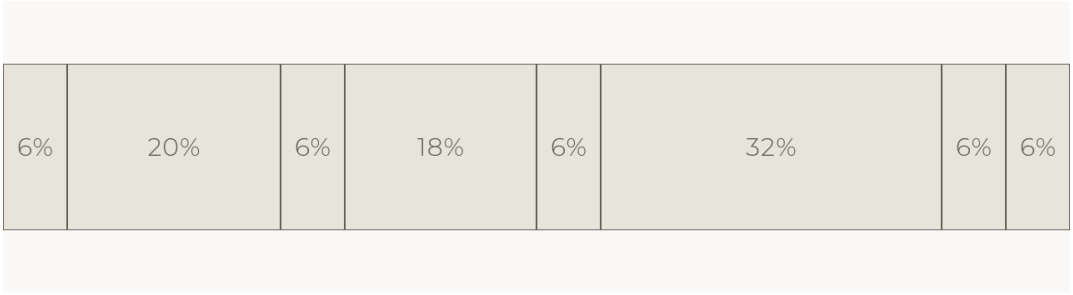


Abb. 3.11: Individuelles Raster, Quelle: Eigene Darstellung

Prozentangaben. [42] Mittels dieses Rasters kann zudem festgestellt werden, wie anpassungsfähig die Rastermöglichkeiten der unterschiedlichen Herangehensweisen sind.

3.2.4 ANORDNUNG DER ELEMENTE

Das Raster der Webseite füllt die komplette Viewportbreite (komplette Breite des Browserfensters) aus. [43] Die beiden äußeren Spaltenbreiten weisen die gleiche Breitenangabe auf, um Inhaltstypen wie beispielsweise Fließtexte in Spalte zwei bis einschließlich sieben anordnen zu können. Dadurch werden Fließtexte nicht direkt am Rand dargestellt und die identischen Spaltenangaben sorgen für eine ausbalancierte Darstellung.

Im Header gibt es nicht viele Möglichkeiten, die Inhalte innovativ anzuordnen, ohne die Benutzerfreundlichkeit einzuschränken. Aus diesem Grund ist die Anordnung auf kleinen Displays klassisch gehalten, indem das Logo links angeordnet ist und das Menü-Icon rechts (vgl. Abb. 3.12). Sobald sich die Bildschirmbreite jedoch vergrößert, rutscht das Logo in die Mitte und an den Seiten werden die vorher versteckten Interaktionselemente platziert (vgl. Abb. 3.13). In Abschnitt *3.1.1 Anatomie einer Webseite* wurde zuvor erörtert, dass diese Anordnung nicht häufig eingesetzt wird. Dadurch findet bereits in der Navigationsleiste eine Abhebung zu den etablierten Designmustern statt.

Abb. 3.12: Header Mobil (blaue Hilfslinen stellen das Raster dar), Quelle: Eigene Darstellung

Abb. 3.13: Header Desktop, Quelle: Eigene Darstellung

42 Vgl. Hahn, 2020, S. 305
43 Vgl. MDN Web Docs: Viewport - MDN Web Docs Glossary: Definitions of Web-related terms | MDN, in: MDN Web Docs, 05.11.2020, https://developer.mozilla.org/en-US/docs/Glossary/Viewport (abgerufen am 14.08.2021).

In Abschnitt *3.1.1 Anatomie einer Webseite* wurde zudem der häufige Einsatz von Hero-Bildern konkretisiert. Da dieser Bereich viel Aufmerksamkeit erlangt und ausschlaggebend für den ersten Eindruck der Gestaltung ist, sollte bei einer individuellen Gestaltung auf die Anwendung eines bildschirmfüllenden Fotos als Einleitung verzichtet werden. Als Ersatz dafür wurden hier mehrere unterschiedliche Inhaltselemente ausgewählt, die sich kachelartig in einem exklusiven Raster anordnen (vgl. Abb. 3.14). Diese kachelartige Anordnung wird im weiteren Verlauf der Gestaltung wiederverwendet, wie beispielsweise innerhalb der Sektion über die Geschichte des Cafés (vgl. Abb. 3.16). Bei der mobilen Ansicht werden die Inhalte untereinander angeordnet und inhaltslose, rein dekorative Inhalte werden versteckt (vgl. Abb. 3.15, vgl. Abb. 3.17). Diese Anordnung soll gleichzeitig testen, ob sich die verschiedenen Inhaltselemente bei den unterschiedlichen Herangehensweisen im Raster selbst gut verschieben lassen.

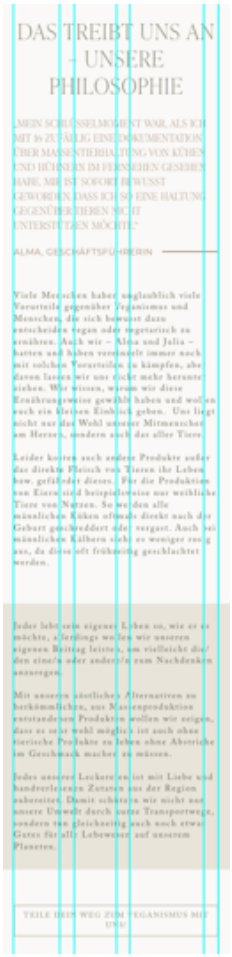
Abb. 3.14 (links): Hero Desktop, Abb. 3.15 (rechts): Hero Mobil
Quellen: Eigene Darstellung

Abb. 3.16 (links): Geschichte Desktop, Abb. 3.17 (rechts): Geschichte Mobil, Quellen: Eigene Darstellung

Da mit der Umsetzung dieser Gestaltung nicht nur getestet werden soll, ob individuelle Designentscheidungen mit den unterschiedlichen Herangehensweisen entwickelt werden können, sondern auch standardisierte Layouts, werden diese in dieser Gestaltung ebenfalls mit aufgegriffen. Ein Beispiel dafür wäre die Auflistung der Produkte in einem dreispaltigen Raster. Im Gegensatz zum individuellen Raster haben hier alle drei Spalten die gleiche Breite. Damit diese starre Gestaltung trotzdem zum individuellen Flair passt und etwas aufgelockert wird, wird die Darstellung der Elemente innerhalb der Spalten etwas durchmischt (vgl. Abb. 3.18). Auf kleineren Endgeräten wird das dreispaltige Raster aufgelöst, die Elemente werden untereinander angeordnet und die Auflistungen der Produkte werden eingeklappt (vgl. Abb. 3.19). Um ein weiteres etabliertes Designmuster mit aufzugreifen, wurde im Kontaktbereich der Webseite eine bildschirmfüllende Darstellung erstellt (vgl. Abb. 3.20).

Abb. 3.18 (oben mittig): Produktauflistung Desktop,
Abb. 3.19 (oben rechts): Produktauflistung Mobil,
Abb. 3.20 (oben links): Kontakt Desktop,
Abb. 3.21 (unten): Kontakt Mobil,
Quellen: Eigene Darstellung

Der Bereich über die Philosophie des Cafés soll zeigen, wie rein textliche Inhalte dargestellt werden können. Auch hier werden die einzelnen Blöcke mobil untereinander und bei größeren Bildschirmbreiten nebeneinander angeordnet (vgl. Abb. 3.22, Abb. 3.23). Variation zwischen den Textblöcken findet durch Einsatz von Typografie und farblich abgesetzten Hintergründen statt. Gleichzeitig werden die Elemente auch hier an dem individuellen Raster angeordnet.



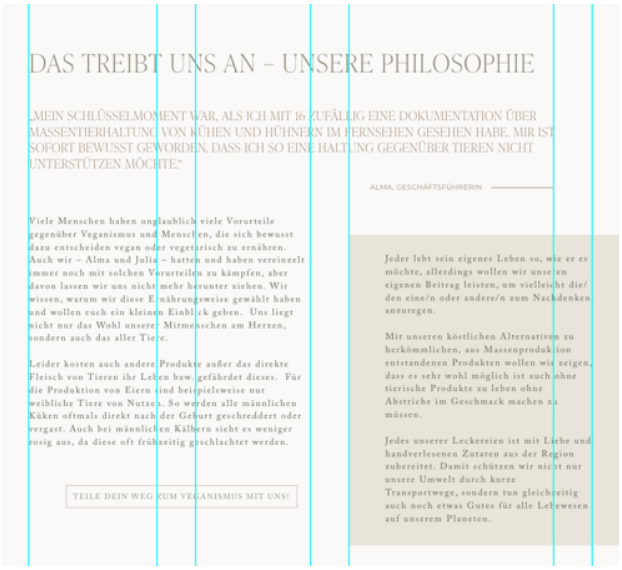


Abb. 3.22 (links): Philosophie Mobil, Abb. 3.23 (rechts): Philosophie Desktop, Quellen: Eigene Darstellung

Neben der Anordnung der eigentlichen Inhaltselemente wird eine Reihe von verschiedenen Designelementen eingesetzt. Dies sind unter anderem Kreiselemente, die an bestimmten Ecken von Bildern arrangiert werden oder farblich abgesetzte, rechteckige Hintergrundelemente. Ihre Funktion besteht darin, das angewendete Raster aufzulockern und den dadurch entstandenen, überflüssigen Weißraum passend zur Atmosphäre auszufüllen.

4 TECHNISCHE UMSETZUNG

Nachdem im vorherigen Kapitel die Gestaltung für die Webseite des fiktiven veganen Cafés erörtert wurde, wird nun die technische Umsetzung beleuchtet. Die Umsetzung dieser Webseite finden mit drei unterschiedlichen Herangehensweisen an CSS statt. Die erste Umsetzung entspricht einer Umsetzung mit handgeschriebenen, reinen CSS. Die zweite und dritte Umsetzung findet mit CSS-Frameworks statt, einmal mittels TailwindCSS als Utility-Klassen basiertes CSS-Framework und einmal mittels Bootstrap als Komponenten basiertes CSS-Framework. Dabei werden Aspekte wie die Funktionsweisen, die genutzten Werkzeuge, die Performance im Browser und die Code-Qualität des Quellcodes näher betrachtet. Die daraus gewonnenen Ergebnisse dienen als Basis für den im nächsten Kapitel erfolgenden Vergleich. Bevor jedoch die erste Herangehensweise untersucht wird, werden zwei allgemeine Strategien aus der Frontend-Entwicklung vorgestellt.

4.1 ALLGEMEINE STRATEGIEN

Für die Frontend-Entwicklung haben sich im Laufe der Jahre bestimmte, bereits mehrfach erprobte Methoden und Maßnahmen herauskristallisiert, die heutzutage als „Best Practice“ beschrieben werden. [1] Sie sorgen dafür, dass Webseiten für verschiedene Endgeräte besser angepasst und optimiert sind. Da diese Strategien zwei Vergleichspunkte darstellen und im weiteren Verlauf dieser Arbeit mehrfach angesprochen werden, werden sie in diesem Abschnitt einmal ausführlich erläutert.

4.1.1 RESPONSIVE WEBDESIGN

Im Mai 2010 veröffentlichte Ethan Marcotte im Webmagazin „A List Apart“ einen Artikel mit dem Titel „Responsive Webdesign“ [2] , der ein Umdenken in der Frontend-Entwicklung angeregt hat. Hier wurde die Idee vorgestellt, dass sich Webseiten an die Bildschirmbreite des genutzten Endgeräts anpassen sollten. Dadurch hat sich eine Strategie entwickelt, die heute als Norm in der Entwicklung angesehen wird. Davor wurden Webseiten statisch mit einer festen Pixelbreite umgesetzt, wie z. B. das 960-Grid-System. Im Gegensatz dazu werden heute Webseiten flexibel gestaltet, indem sogenannte

1 Vgl. Dudenredaktion: Best Practice, in: Duden, o. D., https://www.duden.de/rechtschreibung/Best_Practice (abgerufen am 14.08.2021).
2 Vgl. Marcotte, Ethan/Tom Greenwood/Preston So/Cathy Dutton/Erin Casali/Erin Casali: Responsive Web Design, in: A List Apart, 01.05.2019, https://alistapart.com/article/responsive-web-design/ (abgerufen am 14.08.2021).

Media-Queries eingesetzt werden. [3] Media-Queries helfen dabei, den Wirkungsraum von Deklarationen je nach Gerätetyp oder Parametern wie Bildschirmbreite oder -ausrichtung einzugrenzen. [4]

Mit dem Einsatz von Media-Queries und dem Aufkommen des Responsive Webdesigns, ist eine weitere Methode in den Fokus der Entwickelnden gerückt: *Mobile First*. „Mobile First bezeichnet ein Vorgehen, das von einer für Smartphones optimierten Darstellung ausgeht und sich dann nach und nach für die größeren Auflösungen anpasst [...]." [5] Mobile Endgeräte weisen mehr Beschränkungen hinsichtlich Bildschirmbreite, Arbeitsspeicher und Rechenleistung auf als Desktopgeräte. Dadurch ist es aus Entwicklungssicht einfacher, die Bedingungen für mobile Endgeräte als Standard anzunehmen und den Quellcode für größere Bildschirme zu erweitern, anstatt den Quellcode für mobile Geräte herabzustufen. [6]

4.1.2 PROGRESSIVE ENHANCEMENT

Die zweite Methodik, die bei der Entwicklung einer Webseite zu beachten ist, nennt sich *Progressive Enhancement*. Sie beschreibt den Ansatz, dass die Kernfunktionen einer Webseite möglichst vollständig nutzbar sein sollten. Wie bei Mobile First wird auch hier der Standard danach definiert, welche Funktionalitäten mit einem „schwachen" System ausführbar sind. Mit komplexeren und aufwendigeren Technologien, die nicht alle Systeme unterstützen, sollten keine grundlegenden Funktionen programmiert sein, sodass möglichst wenig Endgeräte von der Nutzung einer Webseite ausgeschlossen werden. [7]

Dazu gehört beispielsweise auch die rücksichtsvolle Verwendung von JavaScript, da es sich dabei um eine wie in Abschnitt *2.3 JavaScript* beschrieben, „client-side scripting language" handelt. Dadurch ist sie von den Eigenschaften und Fähigkeiten des Browsers des Anwendenden abhängig und könnte dadurch bei Beschränkungen entweder nur teilweise oder gar nicht ausgeführt werden. [8] Die Anwendenden können sogar bewusst entscheiden, dass das Ausführen von JavaScript im Browser komplett unterbunden wird. [9] Deswegen ist laut Progressive Enhancement auch hier der Ansatz zu verfolgen, dass grundlegende Funktionen einer Webseite möglichst nicht von

3 Vgl. Hahn, 2020, S.107
4 Vgl. MDN Web Docs: Using media queries - CSS: Cascading Style Sheets | MDN, in: MDN Web Docs, 13.08.2021, https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries (abgerufen am 14.08.2021).
5 Hahn, 2020, S.113
6 Vgl. MDN Web Docs: Mobile first - Progressive web apps (PWAs) | MDN, in: MDN Web Docs, 01.06.2021, https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Responsive/Mobile_first (abgerufen am 14.08.2021).
7 Vgl. Hahn, 2020, S.115
8 Vgl. Robbins, 2018, S. 593
9 Vgl. Hahn, 2020, S.33

JavaScript abhängig sein sollten. Allerdings ist es nicht das Ziel, JavaScript komplett wegfallen zu lassen, sondern es gezielt für Verbesserungen einzusetzen, die mit HTML oder CSS nicht umsetzbar sind.

Schlussendlich soll mit diesem Ansatz sichergestellt werden, dass so viele Endgeräte wie möglich eine Webseite aufrufen und im besten Fall ohne Beschränkungen nutzen können.

4.2 HANDGESCHRIEBENES CSS

Die erste Herangehensweise an CSS, die beleuchtet und anhand einer Umsetzung verglichen werden soll, ist die Umsetzung mit handgeschriebenen CSS. Dabei werden keine CSS-Frameworks verwendet, sodass nur auf die Funktionsweise von CSS und möglichen Präprozessoren zurückgegriffen wird. Da die grundlegenden Funktionsweisen schon aus Abschnitt *2.2 Cascading Style Sheets* bekannt sind, werden sie nicht ein weiteres Mal besprochen, sondern als Grundwissen vorausgesetzt.

4.2.1 WERKZEUGE & TOOLING

Ein Endergebnis kann bereits mit den einfachsten Mitteln erreicht werden, sodass eigentlich keine weiteren Werkzeuge oder Abhängigkeiten nötig wären. Dafür muss die Gestaltung in einer `styles.css` Datei überführt werden, die wiederum in einem HTML-Dokument verlinkt wird und schon ist die Webseite bereit. Es gibt jedoch eine Vielzahl fortgeschrittlicher Werkzeuge für CSS, die ihren Weg in den professionellen Workflow von Webentwickelnden gefunden haben. [10]

Um einen realistischen Vergleich der Herangehensweisen zu ermöglichen, soll für jede Herangehensweise der optimale Ausgangspunkt geschaffen werden. Aus diesem Grund wird für die Herangehensweise mit handgeschriebenen CSS nicht auf den oben beschriebenen Workflow zurückgegriffen, sondern es werden weitere Werkzeuge und Abhängigkeiten hinzugezogen. Diese sollen sowohl die Entwicklung vereinfachen als auch das Endergebnis optimieren.

Um externe Abhängigkeiten verwenden zu können, muss ein Paketmanager vorhanden sein, der die gewünschten Abhängigkeiten herunterladen, installieren, entpacken und updaten kann. Sowohl bei der Umsetzung mit handgeschriebenen CSS als auch bei den anderen Herangehensweisen wird `npm` als Paketmanager verwendet. `Npm` wird in der JavaScript-Laufzeitumgebung `node.js` verwendet und kann auf frei verfügbare

10 Vgl. Robbins, 2018, S. 572

Softwarelösungen zugreifen. [11] Sobald eine dieser Softwarelösungen in einem Projekt eingebunden wird, wird sie als Abhängigkeit bezeichnet. Alle verwendeten Abhängigkeiten werden in der `package.json` Datei mit einer Versionsnummer festgehalten. [12]

4.2.1.1 PROZESSOREN

Für die Entwicklung mit CSS gibt es neben CSS-Frameworks weitere nützliche Werkzeuge, die den Workflow verbessern und mehr Funktionalitäten bereitstellen. Diese lassen sich laut Jennifer Robbins und ihrem Buch „Learning Web Design“ in zwei Kategorien einteilen:

Präprozessoren:
In der ersten Kategorie werden Präprozessoren wie zum Beispiel Sass oder LESS zusammengefasst. Sie nutzen eine zeiteffizientere Syntax und konvertieren diese in die eigentliche Syntax von CSS. [13]

Postprozessoren:
In die zweite Kategorie fallen CSS-Optimierungstools, die reine CSS-Dateien verarbeiten und diese beispielsweise für den Browser optimieren oder die Dateigröße minimieren. [14]

In dieser Umsetzung wird auf den Präprozessor Sass zurückgegriffen. Laut der Umfrage von „State of CSS 2020“ gaben 89% der 11.492 Befragten an, dass sie Sass verwenden. Dieser Präprozessor hat somit im Vergleich zur Alternative LESS mit nur 57%, deutlich mehr Anwendende. [15]

Syntactically awesome style sheets – kurz **Sass** – weist drei große Unterschiede zu der Syntax von reinem CSS auf. Der erste Unterschied besteht darin, dass mittels Sass Verschachtelungen des Quellcodes möglich sind. Diese Syntax soll dabei helfen, Wiederholungen im Quellcode entgegenzuwirken. [16] Die zwei weiteren Unterschiede sind zusätzliche Funktionen, die eingesetzt werden können, um die Webentwicklung zu vereinfachen. Mithilfe von Sass lassen sich Variablen definieren, die überall im Code wiederverwendet werden können. [17] Es lassen sich jedoch nicht nur Variablen wiederverwenden, sondern mit der Nutzung von `@mixins` auch

11 Vgl. npmjs: Npm - a JavaScript package manager, in: npmjs, 12.08.2021, https://www.npmjs.com/package/npm (abgerufen am 14.08.2021)
12 Vgl. npmjs: Specifying dependencies and devDependencies in a package.json file | npm Docs, in: npmjs, o. D., https://docs.npmjs.com/specifying-dependencies-and-devdependencies-in-a-package-json-file/ (abgerufen am 14.08.2021)
13 Vgl. Robbins, 2018, S. 572
14 Vgl. Robbins, 2018, S. 572
15 Vgl. State Of CSS: The State of CSS 2020: Pre-/Post-processors, in: State Of CSS, 2020, https://2020.stateofcss.com/en-us/technologies/pre-post-processors/ (abgerufen am 17.08.2021).
16 Vgl. Robbins, 2018, S. 573
17 Vgl. Robbins, 2018, S. 574

ganze Deklarationsblöcke. [18] Darüber hinaus werden weitere Funktionalitäten hinzugefügt, wie die Ermöglichung der Verwendung von Programmierschleifen. [19]

Zudem wird `postcss` als Postprozessor verwendet. `postcss` ist laut der Statistik von „State of CSS 2020“ mit 48% der meistverwendete Postprozessor. 92% der Befragten haben darüber hinaus angegeben, dass sie mit diesem Werkzeug am zufriedensten sind. [20] Die Funktionsweise von `postcss` basiert auf die Analyse des vorhandenen CSS-Quellcodes und erstellt einen „Abstract Syntax Tree“, sodass es mittels Plugins ermöglicht wird, den Quellcode zu manipulieren. [21]

4.2.1.2 WEITERE ABHÄNGIGKEITEN

Grundsätzlich werden bei den installierten Abhängigkeiten zwischen `dependencies` und `devDependencies` unterschieden. Unter `dependencies` werden Abhängigkeiten zusammengefasst, die eine Anwendung unbedingt benötigt, auch während der Veröffentlichung auf einer Domain. Im Gegensatz dazu stehen die `devDependencies`, die nur lokal verwendet werden, um die Anwendung zu testen oder die Entwicklung zu optimieren. [22]

Die `dependencies` dieser Anwendung sind:

- `normalize-scss`: Die User-Agent-Sylesheets unterschiedlicher Browser werden untereinander angepasst (vgl. *4.2.3 Realisierung von Cross-Browser Rendering*) [23]
- `node-sass`: Ermöglicht die Verwendung von Sass und kompiliert Sass-Syntax in CSS-Syntax [24]
- `postcss`: Manipulierung von CSS-Dateien mithilfe von JS-Plugins [25]
- `postcss-cli`: Ausführung von `postcss` über die Kommandozeile [26]
- `purgecss`: Überprüfung des CSS-Codes auf nicht genutzte Deklarationen [27]

18 Vgl. Robbins, 2018, S. 575
19 Vgl. Lanciaux, 2021, S. 63.
20 Vgl. State Of CSS, 2020.
21 Vgl. Robbins, 2018, S. 577
22 Vgl. npmjs, o. D.
23 Vgl. npmjs: Npm: normalize-sass, in: npmjs, 30.08.2014, https://www.npmjs.com/package/normalize-sass (abgerufen am 22.09.2021).
24 Vgl. npmjs: Npm: node-sass, in: npmjs, 24.06.2021, https://www.npmjs.com/package/node-sass (abgerufen am 26.08.2021).
25 Vgl. npmjs: Npm: postcss, in: npmjs, 21.07.2021, https://www.npmjs.com/package/postcss (abgerufen am 26.08.2021).
26 Vgl. npmjs: Npm: postcss-cli, in: npmjs, 12.12.2020, https://www.npmjs.com/package/postcss-cli (abgerufen am 15.08.2021).
27 Vgl. PostCSS: PostCSS - a tool for transforming CSS with JavaScript, in: PostCSS, o. D., https://postcss.org/ (abgerufen am 15.08.2021).

Die `devDependencies` dieser Anwendung sind:

- `minify`: Minimieren von JS-Dateien [28]
- `nodemon`: Automatisiertes Neustarten der Anwendung bei Änderungen [29]
- `npm-run-all`: Ausführung mehrerer `npm`-Skripte gleichzeitig [30]
- `serve`: Erstellen eines lokalen Servers zum Laden statischer Dateien [31]
- `stylelint`: Überprüfung des Quellcodes auf Wiederholungen und Syntax-Fehler [32]
- `stylelint-config-sass-guidelines`: Anwendung einer Sammlung von Standard-Regeln für die Sass-Syntax [33]
- `stylelint-config-standard`: Anwendung einer Sammlung von Standard-Regeln für die CSS-Syntax [34]

Mit diesen Abhängigkeiten lassen sich in der `scripts` Eigenschaft der `package.json` Skripte definieren. Die durch die Skripte angesprochenen Abhängigkeiten werden bei ihrem Aufruf ausgeführt. [35] Insgesamt wurden neun unterschiedliche Skripte erstellt, die eine oder mehrere Abhängigkeiten gleichzeitig aufrufen.

4.2.1.3 METHODIK & DATEISTRUKTUR

Nicht nur Prozessoren und geladene Abhängigkeiten können dabei helfen, den Quellcode zu optimieren, sondern auch die Anwendung von bestimmten Methodiken. Sie sollen den Entwickelnden dabei unterstützen, beispielsweise große Projekte mit viel Quellcode einfacher warten zu können. Beispiele für bekannte Methodiken wären OOCSS, SMACSS, SUITCSS und Atomic. [36] Die jedoch meistverwendete Methodik ist laut der Statistik von „State of CSS 2020“ BEM. 67% der Befragten gaben an, dass sie auf diese Methodik zurückgreifen. [37] BEM soll dabei helfen, eine Struktur in die CSS-Klassen zu bringen, sodass diese adaptierbar und flexibel gestaltet sind. [38] Zudem gibt

28 Vgl. npmjs: Npm: minify, in: npmjs, 04.05.2021, https://www.npmjs.com/package/minify (abgerufen am 15.08.2021).
29 Vgl. npmjs: Npm: nodemon, in: npmjs, 10.07.2021, https://www.npmjs.com/package/nodemon (abgerufen am 15.08.2021).
30 Vgl. npmjs: Npm: npm-run-all, in: npmjs, 24.11.2018, https://www.npmjs.com/package/npm-run-all (abgerufen am 15.08.2021).
31 Vgl. npmjs: Npm: serve, in: npmjs, 08.06.2021, https://www.npmjs.com/package/serve (abgerufen am 15.08.2021).
32 Vgl. Stylelint: Home | Stylelint, in: Stylelint, o. D., https://stylelint.io/ (abgerufen am 15.08.2021).
33 Vgl. Bjankord: GitHub - bjankord/stylelint-config-sass-guidelines: A stylelint config inspired by https://sass-guidelin.es/, in: GitHub, o. D., https://github.com/bjankord/stylelint-config-sass-guidelines (abgerufen am 15.08.2021).
34 Vgl. npmjs: Npm: stylelint-config-standard, in: npmjs, 24.04.2021, https://www.npmjs.com/package/stylelint-config-standard (abgerufen am 15.08.2021).
35 Vgl. npmjs: Scripts | npm Docs, in: npmjs, o. D., https://docs.npmjs.com/cli/v7/using-npm/scripts/ (abgerufen am 15.08.2021).
36 Vgl. Strukchinsky, Vsevolod Vladimir Starkov And Contributors: BEM — Block Element Modifier, in: getbem, o. D., http://getbem.com/introduction/ (abgerufen am 15.08.2021).
37 Vgl. StateOfCSS: The State of CSS 2020: CSS Methodologies, in: StateOfCSS, o. D., https://2020.stateofcss.com/en-us/technologies/methodologies/ (abgerufen am 15.08.2021).
38 Vgl. Strukchinsky, o. D.

```
<!-- HTML -->
<a class="button button--red">
    <p class="button__text">Text</p>
</a>
```

Abb. 4.1: Klassenbenennung mit BEM,
Quelle: Eigene Darstellung

BEM eine Dateistruktur vor. Der Name der Methodik **BEM** (*Block-Element-Modifier*) beschreibt eine Benennungskonvention nach Block, Element und Modifier. Dabei beschreibt ein Block eine eigenstehende Entität, wie z. B. eine Liste. Ein einzelnes Listenelement ist dagegen ein Beispiel für ein Element, da es nicht eigenständig und im semantischen Kontext an den Block gebunden ist. Modifier sind Ergänzungen und definieren Angaben über das Aussehen oder das Verhalten von Blockelementen. [39]

Abbildung 4.1 zeigt eine mögliche Klassenbenennung eines Buttons mithilfe von BEM.

Damit der zugehörige Quellcode einer Entität leicht zu finden ist, wird dieser in einzelnen Sass-Dateien im Ordner `/src/scss/blocks` gespeichert. Für diese Umsetzung sind insgesamt 16 Sass-Dateien in diesem Ordner abgelegt worden. Im Ordner `/src/scss/configurations` liegt der Quellcode, der keiner Entität zugehörig ist, wie z. B. Definitionen von Gestaltungsangaben, die mithilfe von Elementselektoren umgesetzt werden, Variablendefinitionen oder Quellcode zum Laden von Schriften.

4.2.2 UMSETZUNG DER GESTALTUNGSANGABEN

Damit die aus Abschnitt *3.2 Gestaltung* erstellte Gestaltung exakt umgesetzt werden kann, müssen die vordefinierten Gestaltungsparameter in die Entwicklung überführt und festgehalten werden.

Wie in Abschnitt *4.2.1.1 Prozessoren* bereits erörtert, ist es mit der Verwendung von Sass möglich, wiederverwendbare Variablen zu definieren. Gestaltungsparameter wie Farben, Schriftarten und -größen, Laufweiten, Zeilenabstände, Breitenangaben und Umbrechungspunkte (auch Breakpoints genannt) für die Media-Queries werden demnach in der Datei `_variables.scss` definiert.

Parameter wie Schriftgrößen und Breiten- oder Höhenangaben werden mit rem angegeben. Im Gegensatz zu Pixel bezieht sich eine Größenangabe mit rem auf die Schriftgröße des Wurzel-Elements und ist dadurch eine relative Größenangabe. In den meisten Browsern beträgt die Schriftgröße im Normalfall 16 Pixel. [40] Aufgrund der Barrierefreiheit lässt sich diese Angabe jedoch manipulieren. Würde sich die Schriftgröße im Browser verändern, so würden sich auch alle relativen Größenangaben demnach anpassen.

39 Vgl. Strukchinsky, o. D.
40 Vgl. Hahn, 2020, S. 305.

```scss
/* Sass */
@mixin container {
  display: grid;
  grid-template-columns: $grid-spacing-outside 20% $grid-spacing-outside 18% $grid-spacing-outside 32% $grid-spacing-outside $grid-spacing-outside;
  grid-template-rows: auto;
  max-width: $screen-huger;

  @media (min-width: $screen-huger) {
    padding: $spacing-0 $grid-spacing-outside;
  }
}
```

Abb. 4.2: Rasterdefinition in einem `@mixin`, Quelle: Eigene Darstellung

Alle Angaben in Pixel werden jedoch nicht beeinflusst. Dadurch ist eine Angabe in rem barrierefreier.

Variablen lassen sich anhand des vorangestellten Zeichens `$` erkennen. Mithilfe dieses Zeichens und des Variablennamens lassen sie sich im fortlaufenden Quellcode referenzieren.

Neben den vorgegebenen Designparametern ist ebenfalls ein individuelles Raster vorgegeben. Für die Umsetzung des Rasters aus Abschnitt *3.2.3 Raster*, wird das CSS-Grid-System verwendet (vgl. Abb. 4.2).

4.2.3 REALISIERUNG VON CROSS-BROWSER RENDERING

Unter Cross-Browser Rendering wird die Darstellung des gleichen Quellcodes in unterschiedlichen Browsern verstanden. Entwickelnde müssen die unterschiedlichen User-Agent-Stylesheets und Eigenschaften des jeweiligen Browsers bei der Umsetzung einer Webseite berücksichtigen, damit der erstellte Quellcode in allen Browsern ein gleiches visuelles Ergebnis erzielt. [41]

Der erste Schritt zu einem identischen Ergebnis in allen Browsern, ist der richtige Umgang mit den User-Agent Stylesheets. Es gibt zwei unterschiedliche Wege einen identischen Ausgangspunkt für die Entwicklung zu erhalten, ohne eine Beeinflussung der eigenen Deklarationen durch die User-Agent-Stylesheets. Die erste Möglichkeit wäre eine komplette Überschreibung aller Deklarationen der Browser, sodass diese dadurch zurückgesetzt werden. Die zweite Möglichkeit besteht darin, die Deklarationen der unterschiedlichen Browser aneinander anzupassen. Der Weg der Zurücksetzung gilt als drastischere Maßnahme. [42] Da bei diesem Ansatz die vom Browser erstellten Deklarationen ausgelöscht werden, gibt es für die Entwickelnden keine alternativen Deklarationen, die eingreifen, wenn bestimmte Gestaltungsangaben nicht getätigt werden. Aus diesem Grund wurde für diese Entwicklung der Weg der Anpassung gewählt, die mithilfe der bereits in Abschnitt *4.2.1 Werkzeuge & Tooling* angesprochenen Abhängigkeit `normalize-scss` umgesetzt werden soll.

41 Vgl. MDN Web Docs: Introduction to cross browser testing - Learn web development | MDN, in: MDN Web Docs, 27.04.2021c, https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Cross_browser_testing/Introduction (abgerufen am 16.08.2021).
42 Vgl. Robbins, 2018, S. 554, S.555

Diese Abhängigkeit ist die Sass-Version der Abhängigkeit `normalize.css`. Bei der Erstellung von `normalize.css` wurden die User-Agent Stylesheets jedes modernen Browsers beleuchtet, auf Gemeinsamkeiten und Unterschiede untersucht und aneinander angepasst. Dadurch bleiben die wichtigsten Kerngestaltungselemente der User-Agent Stylesheets erhalten, sodass beispielsweise Paragrafen weiterhin Abstände nach oben und unten aufweisen und Überschriften mit einem dickeren Schriftschnitt dargestellt werden. [43]

Nachdem die Inhalte der Abhängigkeit mithilfe von `npm` installiert wurden, wird die Normalisierungsdatei von `normalize-scss` in die `styles.scss` Datei importiert. Diese Datei stellt den Ausgangspunkt aller Sass-Dateien dar, da in ihr alle weiteren Dateien zusammengeführt werden.

4.2.4 RESPONSIVE WEBDESIGN & MOBILE FIRST

Neben der Berücksichtigung der User-Agent-Stylesheets sind auch die unterschiedlichen Endgeräte zu beachten. Dafür wurde der Mobile First Ansatz aus Abschnitt *4.1.1 Responsive Webdesign* angewendet. Der Ausgangspunkt der Entwicklung aller Herangehensweisen liegt bei dem iPhone 5/SE mit einer Viewportbreite von 320 und einer Viewporthöhe von 568 Pixel. Dieses Gerät ist neben dem Galaxy Fold das kleinste Handy, das in der Entwickleransicht von Chrome simulierbar ist. Von dieser Größe ausgehend wurde die Gestaltung der einzelnen Inhaltsbereiche der Webseite mittels der definierten Breakpoints aus Abbildung 4.3, nach und nach für größere Bildschirme angepasst. Diese sind angelehnt an die üblichen Geräteauflösungen.

```
$screen-small: 640px;
$screen-middle: 768px;
$screen-large: 1024px;
$screen-larger: 1280px;
$screen-huge: 1440px;
$screen-huger: 1650px;
```

Abb. 4.3: Definierte Breakpoints,
Quelle: Eigene Darstellung

4.2.5 PROGRESSIVE ENHANCEMENT

Auch die zweite angesprochene Strategie aus Abschnitt *4.1 Allgemeine Strategien* ist in diese Entwicklung mit einbezogen worden. Da JavaScript laut Progressive Enhancement für die grundlegenden Funktionen nicht verwendet werden sollte, wurde darauf geachtet, dass alle wichtigen Interaktionen auf den grundlegenden Funktionen von HTML und CSS basieren. In der Umsetzung mit handgeschriebenen CSS stellt die mobile Ansicht des Menüs die größte Herausforderung dar. Das mobile Menü beinhaltet alle Navigationspunkte zu den jeweiligen Inhaltselementen des OnePager. Die Sichtbarkeit dieses Menüs könnte mit einer simplen Interaktionsabfrage in JavaScript umgesetzt werden. Dies würde aber bedeuten, dass alle

43 Vgl. Robbins, 2018, S. 555

Anwendenden, die JS nur bedingt oder gar nicht ausführen können, in der mobilen Ansicht der Webseite die Navigation nicht bedienen könnten. Dadurch würde eine große Einschränkung entstehen. Die Sichtbarkeit des Menüs lässt sich allerdings auch mit einer Checkbox abfragen, ohne dass dabei JS eingesetzt werden muss.

Diese Checkbox-Abfrage könnte auch bei anderen Interaktionsmöglichkeiten eingesetzt werden, wie beispielsweise das Ausklappen der Produktauflistung auf mobilen Endgeräten. Da die Checkbox-Abfrage allerdings auf einen ID-Selektor angewiesen ist und dieser nur einmal pro HTML-Dokument referenziert werden kann, müssten für jedes ausklappbare Element ein neuer ID-Selektor und eine neue Abfrage erstellt werden. [44] Dadurch ist der entstehende Quellcode nicht flexibel und verschlechtert die Code-Qualität. Aufgrund dessen wird diese Interaktion doch auf Basis von JavaScript ausgeführt.

Bei der Verwendung von JavaScript für Regulierungen der Sichtbarkeit ist ebenfalls darauf zu achten, dass die Elemente standardisiert eingeblendet sind, sodass bei der Abwesenheit von JS alle Elemente trotzdem sichtbar bleiben und keine Information verloren geht.

Es gibt jedoch zwei weitere JavaScript-Dateien, die eingesetzt wurden, um Probleme in der Umsetzung mit CSS zu lösen. Diese beeinträchtigen jedoch nicht die grundlegenden Funktionen der Webseite und werden in Abschnitt *4.2.9 Schwierigkeiten bei der Umsetzung* näher betrachtet.

4.2.6 CODE-QUALITÄT

Eine gute Code-Qualität bringt viele Vorteile mit sich, da dadurch die Konsistenz, Struktur, Benutzerfreundlichkeit und Wartbarkeit des Codes gewährleistet wird. [45] Zudem wird er dadurch leichter verständlich, wodurch eine potenzielle Zusammenarbeit von mehreren Entwickelnden einfacher gestaltet wird. [46]

4.2.6.1 VERHINDERUNG VON REDUNDANTEM CODE

Zu der Code-Qualität zählt zudem die Sauberkeit des Quellcodes, der so lang wie nötig, aber so kurz wie möglich sein sollte. Demnach sollten Codewiederholungen unbedingt verhindert werden. Um diesem vorzubeugen,

44 Vgl. Laborenz, 2016, S. 50.
45 Vgl. Meiert, Jens Oliver: The Little Book of HTML/CSS Coding Guidelines, Sebastopol, USA: O'Reilly, 2015, S. 5
46 Vgl. Meiert, 2015, S. 6

wird mithilfe der `@mixin`-Angabe eine Sammlung von CSS-Deklarationen bestimmt, die überall im Code wiederholt eingefügt werden können. Dadurch wird verhindert, dass diese Zeilen sich immer und immer wiederholen und dadurch anfälliger für Fehler beim Warten des Codes werden. Abbildung 4.2. zeigt, wie das individuelle Raster in einem `@mixin` definiert ist. Dieses Raster kann dadurch mittels `@include` auf gewünschte BEM-Klassen angewendet werden. Neben dem Raster wurden ebenfalls `@mixin` für Abstandsangaben und Typografie-Deklarationen, je nach Schriftart, erstellt.

4.2.6.2 SYNTAX & STRUKTUR

Für eine Konsistenz innerhalb der Codestruktur und für die korrekte Anwendung der Syntax können bestimmte Hilfsmittel eingesetzt werden. Sie helfen automatisiert bei der Durchleuchtung des Quellcodes und überprüfen ihn auf das Einhalten definierter Standards aus der Webentwicklung. [47]
Bei dieser Herangehensweise wurde dafür das `postcss` Plugin `stylelint` verwendet. Die im Oberverzeichnis liegende Datei `stylelintrc.js` dient als Konfigurationsobjekt. Dort können Regeln aufgestellt oder importiert werden, anhand derer der Quellcode überprüft wird. [48] Die für diese Anwendung importierten Plugins sind `stylelint-config-standard` und `stylelint-config-sass-guidelines`. Sie überprüfen den Quellcode anhand Struktur- und Syntaxregeln für CSS und Sass. [49] [50] Im Folgenden werdend die verwendeten Regelsammlungen konkretisiert.

Stylelint-config-standard:
Diese Regelsammlung besteht aus mehreren CSS-Regelsammlungen wie „The Idiomatic CSS principles“ oder die „CSS Google Style Guide“. [51] Dabei werden Regeln zusammengefasst, die eine korrekte Klammersetzung zur Umschließung von Deklarationen, das Einhalten von Namenskonventionen bei Selektoren, die Überprüfung auf fehlende Semikola, korrekte Leerzeichensetzung und das korrekte Anwenden von leeren Zeilen definieren. [52]

Stylelint-config-sass-guidelines:
Diese Regelsammlung wiederum basiert auf den aufgestellten Sass Guidelines der Frontend-Entwicklerin Kitty Giraudel. Auch hier wird auf die richtige Verwendung von Semikola und Leerzeichen eingegangen, jedoch werden auch Sass-spezifische Konventionen mit eingeschlossen, wie beispielsweise die Überprüfung auf eine einheitliche Angabe von Variablen, korrekte

47 Vgl. Stylelint, o. D.
48 Vgl. Stylelint, o. D.
49 Vgl. npmjs, o. D.
50 Vgl. Bjankord, o. D
51 Vgl. npmjs, o. D.
52 Vgl. Stylelint, o. D.

Namensbenennung von `@mixins`, Überprüfung auf vorhandene Leerzeilen bei `@include`-Angaben, korrekte Verwendung von Verschachtelungen und die alphabetische Sortierung von CSS-Eigenschaften. [53]

4.2.6.3 VALIDATOREN

Valider Quellcode ist nicht nur ein Qualitätsmerkmal, sondern führt auch zu einer besseren Erreichbarkeit der Webseite und zu besseren Ladezeiten im Browser. [54] Sowohl für HTML als auch für CSS gibt es Validatoren, die den Quellcode auf das Einhalten des Webstandards überprüfen.

Nu HTML Checker:
Für die Überprüfung der HTML-Semantik und -Struktur kann der „Nu HTML Checker“ vom W3C hinzugezogen werden. [55] Dieser HTML-Validator überprüft den Quellcode auf Anforderungen in Bereichen wie Barrierefreiheit, Sicherheit, Wartbarkeit und Funktionsfähigkeit. [56] Abbildung 4.4 zeigt das Ergebnis des Validators für die Umsetzung mit CSS. Laut dem Validator können bei diesem Test keine Fehler gefunden werden.

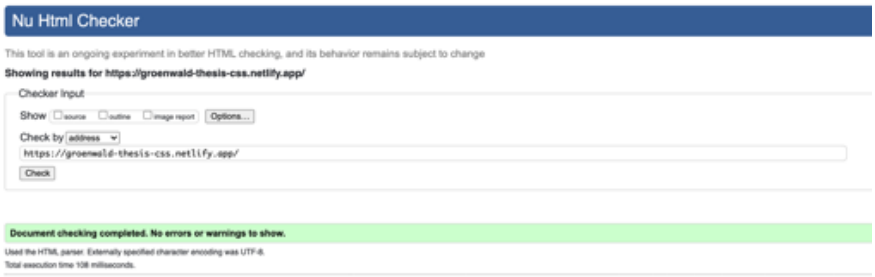


Abb. 4.4: Ergebnis des Nu HTML Checkers, Quelle: Screenshot des Nu HTML Checkers (abgerufen am 16.08.2021)

Jigsaw-Validator:
Der W3C bietet nicht nur Validatoren für HTML an, sondern mit dem „Jigsaw-Validator“ auch einen Test zur Überprüfung des CSS-Quellcodes. [57] Der Validator überprüft den Quellcode auf korrekte Anwendung der CSS3 Standards. Abbildung 4.5 zeigt das Ergebnis des Jigsaw-Validators für das CSS. Auch hier können laut dem Validator keine Fehler gefunden werden.

Abb. 4.5: Ergebnis des Jigsaw-Validators, Quelle: Screenshot des Jigsaw-Validators (abgerufen am 16.08.2021)

53 Vgl. Giraudel, Kitty: Sass Guidelines, in: Sass-Guidelines, o. D., https://sass-guidelin.es/ (abgerufen am 26.08.2021).
54 Vgl. Hahn, 2020, S. 33
55 Vgl. Hahn, 2020, S. 33
56 Vgl. W3C: About the Nu Html Checker, in: Nu Html Checker, o. D., https://validator.w3.org/nu/about.html (abgerufen am 16.08.2021).
57 Vgl. Hahn, 2020, S. 33

4.2.7 PERFORMANCE IM BROWSER

Nicht nur eine ansprechende Gestaltung und guter Quellcode sind Qualitätsmerkmale einer Webseite, sondern auch ihre Performance im Browser. Dabei rückt besonders die Ladezeit in den Fokus, denn je schneller eine Webseite geladen ist, desto höher ist ihre Benutzerfreundlichkeit. Bestimmte Faktoren, die zu der Ladegeschwindigkeit beitragen wie die Datenverbindung oder die Serverleistung, können nicht beeinflusst werden. Ein beeinflussbarer Faktor wiederum ist die Menge an Datenvolumen, die bei einem Aufruf der Webseite geladen wird. [58] Bilder und JS-Code sind Dateien, die beispielsweise mehr Datenvolumen benötigen als CSS- oder HTML-Dateien. Dementsprechend ist dies der erste Anknüpfpunkt, wenn eine Webseite in ihrer Performance optimiert werden soll. [59] Auch bei dieser Umsetzung müssen unter anderem diese Dateien optimiert und angepasst werden, um die Ladezeit verbessern zu können.

4.2.7.1 OPTIMIERUNGSMÖGLICHKEITEN

Zur Optimierung der Bilder werden responsive Bilder eingesetzt. Der entsprechende HTML-Syntax hilft nicht nur dabei, dass die Bilder in mehreren Viewportgrößen responsiv und korrekt angezeigt werden, sondern lässt den Browser auch die korrekte Auflösung selbst auswählen. Dadurch wird verhindert, dass der Browser Bilder in einer viel höheren Auflösung als benötigt lädt, sodass Datenvolumen eingespart werden kann. [60] Für die entsprechende HTML-Syntax wird das eigentliche `<img>`-Element von einem `<picture>`-Tag umschlossen. Innerhalb dieses Tags werden unterschiedliche Bildauflösungen des ursprünglichen Bildes in einem `<source>`–Tag zur Verfügung gestellt.

JavaScript-Dateien:
Um das benötigte JS so gut wie möglich optimieren zu können, wird der Quellcode komprimiert. [61] Bei einer Komprimierung werden beispielsweise alle Leerzeichen und Formatierungen aufgelöst, um den Quellcode so klein

```
<!-- HTML -->
<picture>
  <source sizes="395px" media="(max-width: 400px)" srcset="src/images/flavour-earth-products-cake-w-395.jpg 395w">
  <source sizes="540px" media="(max-width: 540px)" srcset="src/images/flavour-earth-products-cake-w-540.jpg 540w">
  <source sizes="656px" media="(max-width: 700px)" srcset="src/images/flavour-earth-products-cake-w-656.jpg 656w">
  <source sizes="720px" media="(max-width: 1024px)" srcset="src/images/flavour-earth-products-cake-w-720.jpg 720w">
  <source sizes="395px" media="(min-width: 1024px)" srcset="src/images/flavour-earth-products-cake-w-395.jpg 395w">
  <img class="products__image" src="src/images/flavour-earth-products-cake-w-395.jpg" alt="Flavour Eath · Unsere süße Leckereien">
</picture>
```

Abb. 4.6: Responsive Bilder, Quelle: Eigene Darstellung

58 Vgl. Hahn, 2020, S. 152
59 Vgl. Hahn, 2020, S. 153
60 Vgl. MDN Web Docs: Responsive images - Learn web development | MDN, in: MDN Web Docs, 08.07.2021f, https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Responsive_images#resolution_switching_same_size_different_resolutions (abgerufen am 19.08.2021).
61 Vgl. Hahn, 2020, S. 153

wie möglich darstellen zu können. Dafür wird die Abhängigkeit `minify` genutzt. Zudem werden die JS-Skripte asynchron, also parallel mit allen anderen Ressourcen, geladen, damit diese die Ladezeit der Webseite nicht blockieren.

Löschen von nicht verwendetem Quellcode:
Ein weiterer Punkt zur Verringerung des Datenvolumens ist das Beschränken des CSS-Quellcodes auf nur die genutzten CSS-Deklarationen. Dadurch werden die Stylesheets schlanker und sauberer. Dieser Vorgang kann automatisiert geschehen, indem in der `package.json` ein Skript definiert wird, das mithilfe der Abhängigkeit `purgecss` eine Schnittmenge aus dem vorhandenen CSS-Quellcode und den in HTML- und JS-Dateien vorkommenden CSS-Klassen zieht. (vgl. Abb. 4.7).

Es gibt neben der Optimierung von Bildern und JS-Dateien weitere gute Wege die Ladezeit einer Webseite zu verbessern, wie zum Beispiel die Generierung von Dateien, die im Cache des Browsers landen und somit nicht bei jedem Aufruf der Seite neu abgerufen werden müssen. Diese Methode wird aufgrund des Aufwandes nicht mit einbezogen.

```
"css-purge": "purgecss --keyframes --css build/styles.css --content index.html src/js/*.js --output build",
```

Abb. 4.7: Skript mit `purgecss`, Quelle: Eigene Darstellung

4.2.7.2 WERKZEUGE ZUM TESTEN DER PERFORMANCE

Es gibt mehrere Online-Werkzeuge, mit denen sich Webseiten auf ihre Online-Performance testen lassen. Um diese jedoch anwenden zu können, muss die Webseite unter einer Domain bereitgestellt werden. Alle umgesetzten Herangehensweisen innerhalb dieser Arbeit werden mittels des Hosting-Services von Netlify veröffentlicht. Netlify ist ein kostenloser Hosting-Service für statische HTML-Webseiten. [62] Die Domain für die Umsetzung mit handgeschriebenen CSS lautet: https://groenwald-thesis-css.netlify.app/

Pagespeed Insights:
Ein sehr häufig verwendetes Werkzeug zum Testen der Performance ist das „Google Pagespeed Tool“. Es liefert Performance-Ergebnisse in Prozentpunkten und in einem farbigen Ampelsystem. Zusätzlich gibt es Verbesserungsvorschläge und die Ergebnisse werden auf Mobil und Desktop aufgeteilt. Dabei werden Werte zwischen 0% und 49% als schlecht, zwischen 50% und 89% als okay und zwischen 90% und 100% als gut bewertet. [63]

62 Vgl. Netlify: Netlify: Develop & deploy the best web experiences in record time, in: Netlify, o. D., https://www.netlify.com/ (abgerufen am 19.08.2021).
63 Vgl. Hahn, 2020, S. 154

Die Ergebnisse für diese Umsetzung werden in Abbildung 4.8 und Abbildung 4.9 dargestellt. Die Ergebnisse sind jedoch nicht statisch und können schwanken. Bei dem gezeigten Test liegt der Wert für mobile Endgeräte bei 95% und für Desktop bei 99%.

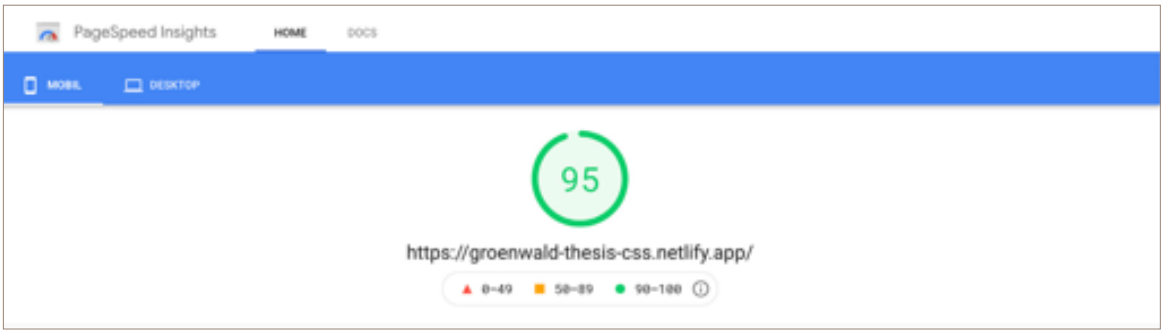


Abb. 4.8: PageSpeed Insights Ergebnis Mobil, Quelle: Screenshot des Pagespeed (abgerufen am 19.08.2021)

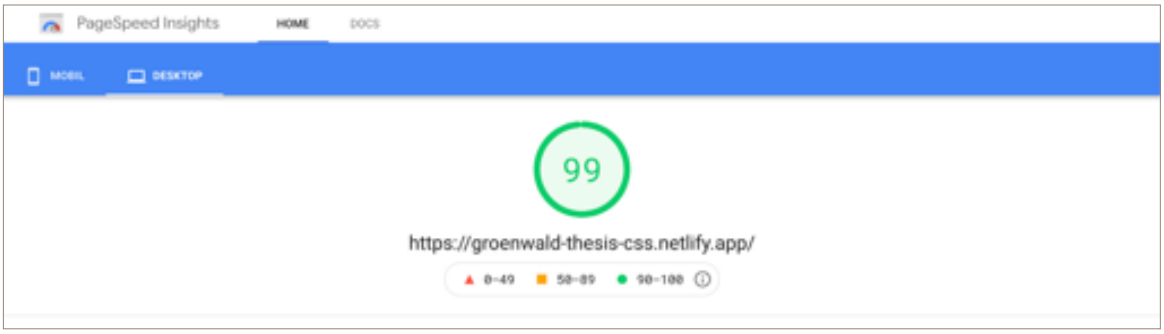


Abb. 4.9: PageSpeed Insights Ergebnis Desktop, Quelle: Screenshot des Pagespeed (abgerufen am 19.08.2021)

Pingdom:
Ein weiteres Werkzeug ist der „Pingdom Website Speed Test“. Im Gegensatz zum Werkzeug von Google werden hier Daten zur Ladezeit, Datenmenge und Serveranfragen bereitgestellt. Die Ergebnisse dieses Werkzeuges werden in Abbildung 4.10 gezeigt. Laut diesem Test beträgt die Datenmenge der Webseite 644.8 Kilobyte, die Ladezeit beträgt 282 Millisekunden und der Server muss 21 Abfragen tätigen.

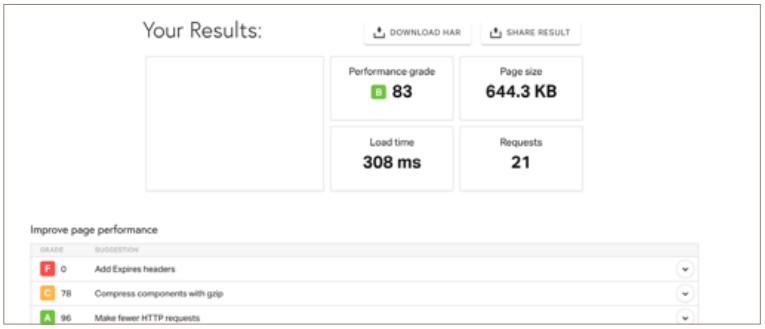


Abb. 4.10: Pingdom Website Speed Ergebnis
Quelle: https://tools.pingdom.com/#5f06f1fa0cc00000
(abgerufen am 25.09.2021)

Chrome Entwickler-Ansicht: Network

In der Entwickleransicht im Chrome-Browser gibt es ein Bereich mit dem Namen „Network“. Mittels dieser Ansicht kann überprüft werden, welche Abfragen getätigt werden. Gleichzeitig können für jede einzelne Ressource der Webseite ihre Größe, ihr Inhalt und ihre benötigte Zeit zum Laden angezeigt werden. [64] Diese werden in einem Wasserfall-Diagramm angezeigt (vgl. Abb. 4.11). Anhand dieser Abbildung kann geschlussfolgert werden, dass als Erstes die `styles.css` und die verwendeten Bilder geladen werden. Etwas weiter hinten wird die JS-Datei `site.js`, die Schriften und das Favicon der Webseite geladen. Die Schriften weisen eine längere Ladezeit auf, da sie extern mittels Typekit importiert werden.

Das CSS-Stylesheet hat eine geladene Dateigröße von 3,6 Kilobyte und der JS-Quellcode eine Dateigröße von 516 Byte. Diese Werte werden im nächsten Abschnitt *4.2.8. Dateigrößen* genauer betrachtet.

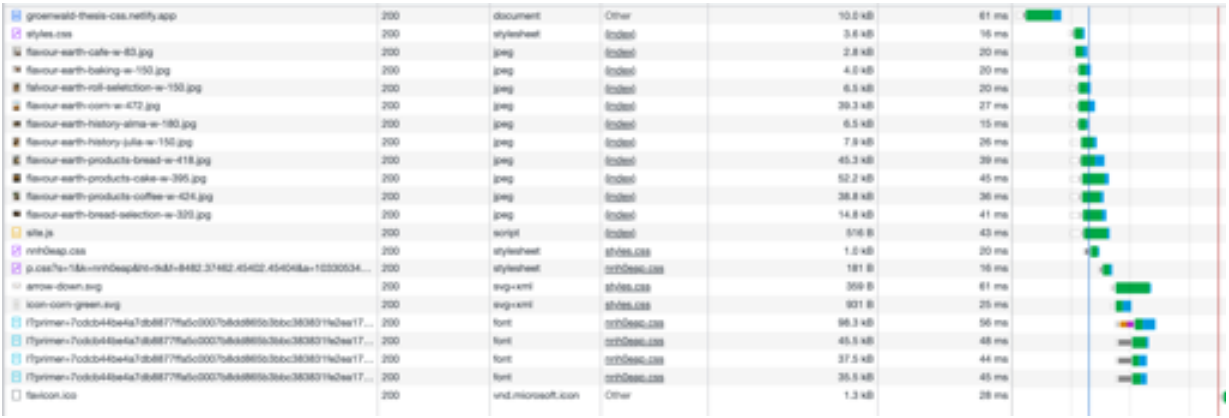


Abb. 4.11: Wasserfalldiagramm im Network-Bereich
Quelle: https://6141eca2a896ee0008de70e5--groenwald-thesis-css.netlify.app/ (abgerufen am 27.09.2021)

4.2.8 DATEIGRÖSSEN

In diesem Abschnitt sollen die optimierten und nicht optimierten Dateigrößen miteinander verglichen werden. Unter optimierter Dateigröße werden die Dateien bezeichnet, die komprimiert und mit `purgecss` optimiert wurden. Dieser Vergleich soll mit den Dateigrößen aus dem zuvor beschriebenen Wasserfalldiagramm hergestellt werden. Abbildung 4.11 zeigt dabei die optimierten Dateigrößen auf und Abbildung 4.12 die nicht optimierten Dateigrößen. Anhand Abbildung 4.12 kann abgelesen werden, dass die nicht optimierte `styles.css` Datei eine Dateigröße von 4,3 Kilobyte aufzeigt und die nicht optimierte `site.js` Datei eine Dateigröße von 715 Byte.

64 Vgl. Chrome Developers: Inspect network activity, in: Chrome Developers, o. D., https://developer.chrome.com/docs/devtools/network/ (abgerufen am 27.08.2021).

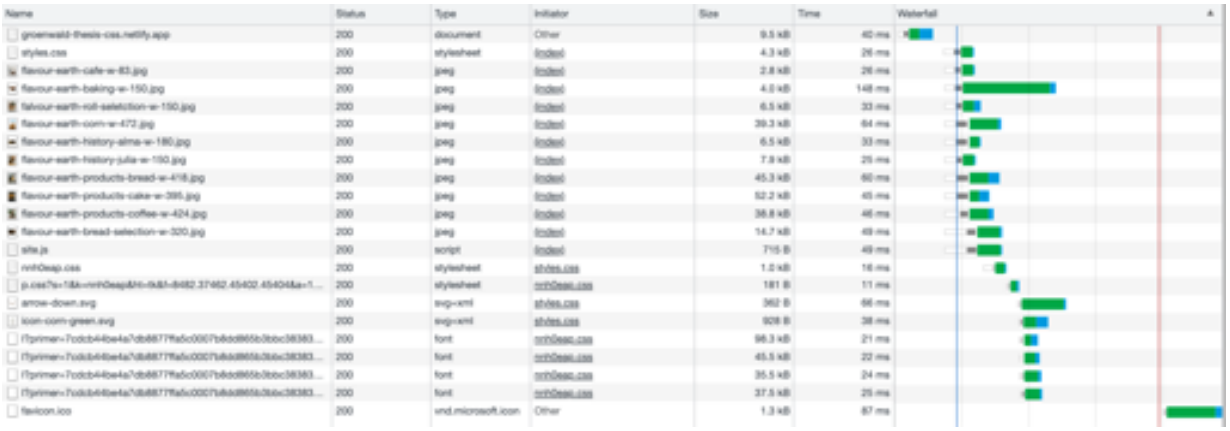


Abb. 4.12: Wasserfalldiagramm mit nicht optimierten Dateien
Quelle: https://612869205069b80007939083--groenwald-thesis-css.netlify.app/ (abgerufen am 27.09.2021)

In der folgenden Tabelle 4.1 werden nun die entsprechenden Dateigrößen gegenübergestellt.

	optimiert	nicht optimiert	Differenz
styles.css	4.300 Byte	3.600 Byte	700 Byte
site.js	715 Byte	516 Byte	199 Byte

Tabelle 4.1: Dateigrößen Vergleich CSS, Quelle: Eigene Darstellung

Schlussendlich kann also gesagt werden, dass durch die Optimierung des CSS-Stylesheets 700 Byte in der Dateigröße eingespart werden konnte. Innerhalb des JS-Quellcodes verlief sich die Einsparung auf 199 Byte.

4.2.9 SCHWIERIGKEITEN BEI DER UMSETZUNG

Während der Entwicklung sind zwei Schwierigkeiten im Bereich der Benutzerfreundlichkeit entstanden, die nicht mit reinem HTML und CSS gelöst werden konnten.

1. Problem: Links scrollen zu weit
Die Links in der Navigation sind mit den jeweiligen Inhaltsbereichen des OnePagers durch passende ID-Attribute verknüpft. Beim Aufruf dieser Links scrollt der Inhaltsbereich jedoch direkt an den oberen Bildschirmrand, sodass der erste Teil des Inhalts vom fixierten Header überdeckt wird. Es wurde ausprobiert, mithilfe der Pseudoklasse `:target` dem zu verlinkenden Inhaltsblock einen Versatz hinzuzufügen. Da die Verlinkung aber nicht nur aus einem einzelnen HTML-Element besteht, wurde dieser bei der Verwendung der Pseudoklasse nicht mehr angezeigt. Weitere Ansätze wie das Hinzufügen von positiven und negativen `padding` und `margin` oder die Positionierung von absoluten Elementen wurden aufgrund schlechter Funktionalität oder schlechter Codequalität verworfen. Als schlussendliche Lösung wurde auf JavaScript zurückgegriffen.

2. Problem: Menü schließt sich nicht bei Klick auf Navigationspunkt
Das zweite Problem liegt in der Bedingung des mobilen Menüs. Entgegengesetzt der Erwartung des Nutzers wird das mobile Menü nicht geschlossen, sobald ein Navigationspunkt betätigt wird. Dieses Problem wird normalerweise umgangen, indem neue Unterseiten geladen werden. Da in dieser Umsetzung jedoch Inhaltsblöcke auf der gleichen Seite und kein neues HTML-Dokument verlinkt sind, wurde auch hier auf JS zurückgegriffen.

4.2.10 VORKENNTNISSE

Dieser Abschnitt beschreibt, welche und wie viele Vorkenntnisse innerhalb der Frontend-Entwicklung nötig gewesen sind, um das individuelle Design korrekt überführen zu können.

4.2.10.1 BENÖTIGTE VORKENNTNISSE

Für die komplette Entwicklung der Webseite mit handgeschriebenen CSS sind ausgebaute Vorkenntnisse über CSS und HTML nötig gewesen. Es wurden viele verschachtelte HTML-Elemente angewendet, um komplexe Systeme nachbilden zu können. Als Beispiel dafür kann die Produktauflistung genannt werden, da dort eine Liste innerhalb einer Liste erstellt wurde. Damit weiterhin eine korrekte HTML-Struktur erhalten bleibt, wurde Vorwissen über die Semantik von HTML benötigt. Gleichzeitig wurde auf eine sparsame, gezielte Anwendung von HTML-Elementen mit wenig semantischer Bedeutung wie z. B. `<div>` geachtet.

Die benötigten Vorkenntnisse in CSS sind unter anderem bei der Umsetzung des individuellen Rasters eingesetzt worden. Ein fundiertes Grundwissen über das CSS-System Grid hat dabei geholfen, den Wirkungsraum von `display:grid` abschätzen zu können und die jeweiligen Inhaltselemente mit Befehlen wie `grid-column`/`grid-row` oder `align-self` am Raster korrekt auszurichten. Gleichzeitig wurde viel mit Pseudoelementen wie `::after` und `::before` gearbeitet, um inhaltslose Designelemente wie die Kreise oder die Hintergründe umsetzen zu können, ohne dafür weitere HTML-Elemente erstellen zu müssen. Auch dafür wurde tiefgreifendes Verständnis über die Funktionsweise von Positionierungsangaben benötigt.

Weiteres Vorwissen über JavaScript, die Verwendung von Sass zur Optimierung des Workflows oder die generelle Optimierung von CSS-Dateien mithilfe von bestimmten Abhängigkeiten haben zur Codequalität beigetragen, jedoch waren diese nicht notwendig, um die individuelle Gestaltung auf eine Webseite überführen zu können.

4.2.10.2 BENÖTIGTE ZEIT ZUR UMSETZUNG

Um die jeweils benötigte Zeit für eine Umsetzung ermitteln zu können, wurde das Werkzeug TMetric genutzt. TMetric ist eine Zeittracking-Software. Dort kann anhand von Projekten und erstellten Aufgaben die jeweils aufgewendete Zeit gezählt und angezeigt werden. [65] Somit lässt sich herausfinden, welches Projekt wie viel Zeit in Anspruch genommen hat. Laut der gezählten Zeit bei TMetric wurde für die Entwicklung der individuellen Webseite mit der Herangehensweise des reinen, handgeschrieben CSS insgesamt 25 Stunden und 20 Minuten aufgewendet.

65 Vgl. TMetric: Free Time Tracking Software & App - TMetric, in: TMetric, o. D., https://tmetric.com/ (abgerufen am 23.08.2021).

4.3 TAILWINDCSS

Die zweite Herangehensweise an CSS, die beleuchtet und verglichen werden soll, basiert auf dem Utility First Ansatz. Ryan Lanciaux schreibt in seinem Buch „Modern Front-end Architecture“ diesem Ansatz immer mehr Bedeutung zu: „In recent years, utility-first styling libraries have risen to the forefront of app styling options, popularizes by libraries such as Tailwind and Tachyons.“ [1]

4.3.1 TAILWINDCSS ALS UTILITY-KLASSEN BASIERTES CSS-FRAMEWORK

Die Entwickler Adam Wathan und Steve Schoger haben im November 2017 die erste Version des CSS-Frameworks TailwindCSS veröffentlicht. [2] Dieses CSS-Framework wird mit folgendem Satz vermarktet: „Rapidly build modern websites without ever leaving your HTML“. [3] Was sich genau hinter diesem Satz und hinter dem Utility First Ansatz verbirgt, wird in diesem Kapitel konkreter untersucht.

Auch die Statistik von „State of CSS 2020“, die bereits mehrfach hinzugezogen wurde, zeigt deutlich, dass dieses CSS-Framework im Vergleich zum Vorjahr immer mehr Aufmerksamkeit erlangt, in der Beliebtheit deutlich angestiegen ist und immer mehr Anwendende findet. Zudem zeigt die Statistik, dass TailwindCSS das meistverbreitete und -verwendete Utility-Klassen basierte CSS-Framework ist. [4] Dementsprechend wird es in dieser Arbeit als Repräsentant für den Utility First Ansatz gehandelt.

4.3.2 FUNKTIONSWEISE

Die allgemeine Funktionsweise von Utility-Klassen wurde bereits im Abschnitt *2.4.2.1 Definition von Utility-Klassen* konkretisiert. Dementsprechend soll dieser Abschnitt tiefergehende Funktionen thematisieren, wie die Umsetzung von Media-Queries und die Erstellung von Komponenten.

4.3.2.1 FUNKTIONSWEISE DER UTILITY-KLASSEN

Der Aufbau einer Utility-Klasse wird von Noel Rappin in seinem Buch “Modern CSS with Tailwind” treffend formuliert: „Nearly all of the basic Tailwind classes are thin wrappers around a single CSS style setting [...]“ [5]

1 Lanciaux, 2021, S. 67.
2 tailwindlabs, o. D.
3 Vgl. TailwindCSS: Tailwind CSS - Rapidly build modern websites without ever leaving your HTML., in: TailwindCSS, 15.11.2020, https://tailwindcss.com/ (abgerufen am 19.08.2021).
4 The State of CSS 2020: CSS Frameworks, o. D.
5 Rappin, 2021, S. IX

Demnach sind Utility-Klassen einfache Klassenselektoren, die einer gewissen Benennungskonvention folgen und nur eine CSS-Eigenschaft mit einem expliziten Wert repräsentieren (vgl. Abb. 4.13). [6]

```
/* CSS */
.mb-0 {
  margin-bottom: 0;
}
```

Abb. 4.13: Utility-Klasse, Quelle: Eigene Darstellung

Die Benennungskonvention der Utility-Klassen folgt einem eigenen Schema und entspricht nicht, wie bei reinem CSS, einer Funktionsbeschreibung. Die automatisch generierten Utility-Bezeichnungen spiegeln die CSS-Eigenschaft und deren zugewiesenen Wert wider. [7] Beispielsweise entsteht aus der CSS-Eigenschaft `justify-content: center` die Utility-Klasse `justify-center` und aus `padding-top: 1rem` entsteht `pt-4`. Die Hintergründe des Werte-Systems werden in Abschnitt *4.3.4 Vordefiniertes Gestaltungssystem* genauer betrachtet.

4.3.2.2 UMSETZUNG VON RESPONSIVITÄT

Wie in der Herangehensweise mit handgeschriebenen CSS wird auch in TailwindCSS Responsivität mithilfe von Media-Queries umgesetzt. Die verwendeten Media-Queries werden dazu eingesetzt, den Wirkungsrahmen einer Utility-Klasse zu beeinflussen und zu beschränken. Dazu werden Voranstellungen bestimmter Breitenangaben wie `sm`, `md` oder `lg` verwendet (z. B. `sm:mb-0`), die jeweils einen vordefinierten Breakpoint reflektieren. Die fünf standardisiert mitgelieferten Breakpoints sind laut der Dokumentation von häufig verwendeten Bildschirmauflösungen inspiriert (vgl. Abb. 4.14). [8]

Neben den bereits vordefinierten Media-Queries mit `min-width` lassen sich innerhalb der Konfigurationsdatei von TailwindCSS weitere Media-Query-Typen erstellen wie `max-width`, `print` und `orientation`. Diese werden allerdings nicht standardisiert mitgeliefert. [9]

Breakpoint prefix	Minimum width	CSS
`sm`	640px	`@media (min-width: 640px) { ... }`
`md`	768px	`@media (min-width: 768px) { ... }`
`lg`	1024px	`@media (min-width: 1024px) { ... }`
`xl`	1280px	`@media (min-width: 1280px) { ... }`
`2xl`	1536px	`@media (min-width: 1536px) { ... }`

Abb. 4.14: Vordefinierte Media-Queries von TailwindCSS, Quelle: https://tailwindcss.com/docs/responsive-design (abgerufen am 16.09.2021)

6 Vgl. Rappin, 2021, S.IX
7 Vgl. Rappin, 2021, S.IX
8 Vgl. TailwindCSS: Responsive Design - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/responsive-design#overview (abgerufen am 19.08.2021).
9 Vgl. TailwindCSS: Breakpoints - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/breakpoints (abgerufen am 16.09.2021).

4.3.2.3 UMSETZUNG VON PSEUDOKLASSEN & -ELEMENTEN

Auch Pseudoklassen und -elemente werden durch Voranstellungen definiert (z. B. `hover:bg-primary`). Insgesamt werden 16 Pseudoklassen mitgeliefert. [10] Mit ihnen lassen sich jedoch längst nicht alle Möglichkeiten abbilden, wie es mit normalen CSS-Pseudoklassen der Fall wäre. Beispielsweise lässt sich `:nth-child(3n+1)` nicht abbilden.

Auch in dieser Umsetzung wurden die Pseudoklassen von TailwindCSS verwendet, um Interaktionsmöglichkeiten mit Buttons und Verlinkungen visuell zu kennzeichnen.

Seit der Version 2.1 von TailwindCSS können neben den Pseudoklassen auch wichtige Pseudoelemente wie `::before`, `::after` und `::selection` angewendet werden. [11] Alle Designelemente, wie die farblich abgesetzten Hintergründe oder die Kreise, wurden damit umgesetzt.

4.3.2.4 ERSTELLUNG VON KOMPONENTEN

Wenn TailwindCSS in Zusammenhang mit einem *Content-Management-System* (**CMS**), also einem System zur „Verwaltung und Aufarbeitung von Dokumenten und Daten“ [12], verwendet wird, lassen sich wiederverwendbare Codefragmente auslagern, die aus einer oder sogar mehreren HTML-Elementen bestehen können (vgl. Abb. 4.15). Sie können auch unter der Verwendung von anderen Programmiersprachen wie PHP oder JS-Frameworks, wie Vue.js oder React.js, erstellt werden. [13]

```
<!-- button.antlers.html -->
<button class="bg-cyan-dark text-base text-white uppercase px-6 py-3 hover:rounded-3xl">
    <p class="mt-1 block">
        {{ slot }}
    </p>
</button>
```

Abb. 4.15: Partial (wiederverwendbares Codefragment) im CMS Statamic, Quelle: Eigene Darstellung

Da in dieser Anwendung weder ein CMS noch ein JavaScript-Framework verwendet wird, ist diese Art zur Erstellung von Partials nicht möglich. Es gibt jedoch auch eine von TailwindCSS mitgelieferte Möglichkeit, um wiederkehrende Utility-Klassen auszulagern. Diese Utility-Klassen werden zusammen mit einem Klassennamen in der `components`-Ebene definiert.

10 Vgl. TailwindCSS: Hover, Focus & Other States - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/hover-focus-and-other-states (abgerufen am 23.08.2021).
11 Vgl. TailwindCSS: Just-in-Time Mode - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/just-in-time-mode#pseudo-element-variants (abgerufen am 23.08.2021).
12 Vgl. Dudenredaktion: Content Management, in: Duden, o. D., https://www.duden.de/rechtschreibung/Content_Management (abgerufen am 24.09.2021).
13 Vgl. TailwindCSS: Extracting Components - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/extracting-components#extracting-template-components (abgerufen am 24.08.2021).

Dabei funktioniert die Zuweisung der Sammlung an Utility-Klassen wie ein CSS-Klassenselektor. [14] Die Abbildung 4.16 zeigt ein Beispiel für die Definition einer solchen Sammlung.

```
/* tailwind.css */
@layer components {
  .custom-grid {
    @apply grid grid-cols-[6%,20%,5.5%,18.5%,6%,32%,6%,6%];
  }
}
```

Abb. 4.16: Individuelles Raster, Quelle: Eigene Darstellung

4.3.2.5 EINBINDUNGSMÖGLICHKEITEN

Da nun die grundlegenden Funktionen beleuchtet wurden, stellt sich die Frage, wie TailwindCSS in ein Projekt eingebunden werden kann. Laut der Dokumentation von TailwindCSS gibt es drei Möglichkeiten:

1. Einbindung als `postcss` Plugin
2. Verwendung von TailwindCSS in der *Kommandozeile* (**CLI**)
3. Einbindung mithilfe eines *Content-Delivery-Networks* (**CDN**)

1. Einbindung als postcss Plugin
„The Tailwind developers recommend that Tailwind be installed in most projects as a plugin for PostCSS, which is a general CSS processing tool.“ [15] Demnach empfiehlt Noel Rappin die Einbindung von TailwindCSS als `postcss`-Plugin. Im Gegensatz zu der Herangehensweise mit handgeschriebenen CSS, wird `postcss` dabei nicht als Post- sondern als Präprozessor eingebunden, sodass die generierten Utilities in CSS-Quellcode umgewandelt werden können.

Dieser Installationsweg bietet die meisten Möglichkeiten, TailwindCSS nach den Wünschen des Entwickelnden anzupassen. Daraus schließend wurde dieser Einbindungsweg für diese Umsetzung ausgewählt. Allerdings hat diese Einbindungsmöglichkeit einen Nachteil, da bestimmte Abhängigkeiten vorhanden sein müssen, um alles korrekt ausführen zu können. Dadurch wird der Tooling-Aufwand gesteigert. Welche Abhängigkeiten genau vorhanden sein müssen, wird in Abschnitt *4.3.3.1 Werkzeuge von TailwindCSS* konkreter untersucht.

2. Verwendung von TailwindCSS in der Kommandozeile (CLI)
Die zweite Einbindungsmöglichkeit verwendet TailwindCSS nur innerhalb des *Command-Line-Interfaces*. Indem bestimmte Befehle in der Kommandozeile ausgeführt werden, wird mithilfe der standardisierten Konfigurationsdatei von

14 Vgl. TailwindCSS: Extracting Components - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/extracting-components#extracting-component-classes-with-apply (abgerufen am 24.08.2021).
15 Rappin, 2021, S. 2

TailwindCSS eine CSS-Datei mit allen benötigten Utilities erstellt. Dadurch müssen das CSS-Framework und die ansonsten benötigten Abhängigkeiten nicht eingebunden werden, sodass der ein großer Teil des Tooling-Aufwands erspart bleibt. [16]

Da bei dieser Einbindungsweise jedoch auf die standardisierte Konfigurationsdatei zurückgegriffen wird, können die Utilities nicht individualisiert und angepasst werden. [17] Dadurch ist der/die Entwickelnde in der Umsetzung des Designs deutlich eingeschränkt.

3. Einbindung mithilfe eines Content-Delivery-Networks (CDN)
Mit einem Content-Delivery-Network werden bestimmte Inhalte nicht nur auf einem zentralen Ursprungsserver, sondern auch auf weitere Replica-Webserver gespeichert, sodass der aufzurufende Inhalt schneller erreichbar ist. [18] Auch der komplette CSS-Quellcode von TailwindCSS kann von einem CDN geliefert und mit einem Link eingebunden werden. Dabei geht, wie bei der zweiten Einbindungsmöglichkeit, die Funktion der Anpassbarkeit verloren. Gleichzeitig können weitere Funktionen wie `@apply`, `@variants`, das Hinzufügen von weiteren Varianten für die Pseudoklassen, Installieren von Plugins und das Löschen von nicht verwendeter Utility-Klassen nicht genutzt werden. [19]

4.3.2.6 INHALTE

Laut der Webseite von `npm` werden mit der Installation des CSS-Frameworks TailwindCSS insgesamt 264 Dateien zur Verfügung gestellt. Dabei beläuft sich die Dateigröße der Abhängigkeit auf 34,6 Megabyte. [20]

Um auf die Utility-Klassen des CSS-Frameworks zugreifen zu können, müssen die mitgelieferten Inhalte von TailwindCSS in einer CSS-Datei geladen werden. Diese Datei heißt in diesem Fall `tailwind.css`. Mit der Angabe `@tailwind` werden bestimmte Sammlungen an Inhalte angesprochen und importiert. In der Dokumentation sind die mitgelieferten Inhalte in vier Sammlungen bzw. Ebenen eingeteilt. [21]

16 Vgl. TailwindCSS: Installation - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/installation#using-tailwind-cli (abgerufen am 24.08.2021).
17 Vgl. TailwindCSS, o. D.
18 Vgl. Badach, Anatol: Content Delivery Network, in: researchgate.net, o. D., https://www.researchgate.net/profile/Anatol-Badach/publication/282008087_CDN_-_Content_Delivery_Network/links/560147d708ae07629e52bf24/CDN-Content-Delivery-Network.pdf (abgerufen am 24.08.2021), Abschn. S.1-2.
19 Vgl. TailwindCSS, o. D.
20 Vgl. npmjs: Npm: tailwindcss, in: npmjs, 30.08.2021, https://www.npmjs.com/package/tailwindcss (abgerufen am 05.09.2021).
21 Vgl. TailwindCSS: Functions & Directives - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/functions-and-directives#tailwind (abgerufen am 24.08.2021).

1. @tailwind base:
In dieser Ebene befinden sich die Basis-Gestaltungsangaben. Hauptsächlich befindet sich hier das „Preflight“ von TailwindCSS. [22] Welche Angaben darunter zu verstehen sind, wird in Abschnitt *4.3.6.1 Preflight* verdeutlicht.

2. @tailwind components:
In dieser Ebene befinden sich die Komponenten von TailwindCSS, wie beispielsweise die `container` Komponente. [23]

3. @tailwind utilities:
In dieser Ebene befinden sich die Utility-Klassen von TailwindCSS. [24]

4. @tailwind screens:
Diese Ebene kann optional mit angegeben werden, wenn die responsiven Varianten der Utility-Klassen gesondert geladen werden sollen. Ansonsten werden sie automatisch am Ende des Stylesheets importiert. [25]

4.3.3 WERKZEUGE & TOOLING

Wie zuvor in Abschnitt *4.3.2.5 Einbindungsmöglichkeiten* beschrieben, wird für die Entwicklung mit der Herangehensweise des Utility-Klassen basierten Ansatzes die Installation über `postcss` vorgenommen. Innerhalb des zuvor erwähnten Abschnitts wurde bereits angesprochen, dass explizite Abhängigkeiten notwendig sind, um das CSS-Framework korrekt ausführen zu können. Diese und weitere Abhängigkeiten werden in diesem Abschnitt untersucht.

4.3.3.1 WERKZEUGE VON TAILWINDCSS

Das CSS-Framework selbst wird mit der Abhängigkeit `tailwindcss` eingebunden. Dadurch wird sichergestellt, dass alle Inhalte aus dem Abschnitt *4.3.2.6 Inhalte* auch geladen werden können. Demnach sind die benötigten Abhängigkeiten zur Gewährleistung der korrekten Ausführung die folgenden:

- `tailwindcss`: Installation der Inhalte für das CSS-Framework TailwindCSS [26]
- `postcss`: Manipulierung von CSS-Dateien mithilfe von JS-Plugins [27]

22 Vgl. TailwindCSS, o. D.
23 Vgl. TailwindCSS, o. D.
24 Vgl. TailwindCSS, o. D.
25 Vgl. TailwindCSS, o. D.
26 Vgl. npmjs: Npm: tailwindcss, in: npmja, 26.09.2021, https://www.npmjs.com/package/tailwindcss (abgerufen am 28.09.2021).
27 Vgl. npmjs, 2021.

- `autoprefixer`: Überprüfung des CSS-Codes auf CSS-Eigenschaften, die sogenannte Herstellerpräfixe benötigen, um eine fehlerfreie Anzeige im Browser zu ermöglichen [28]

Ohne diese Abhängigkeiten kann das CSS-Framework TailwindCSS nicht genutzt werden.

4.3.3.2 WEITERE ABHÄNGIGKEITEN

Neben `postcss` und `tailwindcss` werden folgende Abhängigkeiten ebenfalls als `dependencies` eingebunden. Sie ermöglichen es, dass der Quellcode optimiert und TailwindCSS mit der Kommandozeile ausgeführt werden kann.

- `minify`: Minimieren von JS-Dateien [29]
- `postcss-cli`: Ausführung von `postcss` über die Kommandozeile [30]

Die folgenden Abhängigkeiten sind als `devDependencies` gelistet und werden somit nur während der Entwicklung angewendet.

- `nodemon`: Automatisiertes Neustarten der Anwendung bei Fehler [31]
- `npm-rull-all`: Ausführung mehrerer `npm`-Skripte gleichzeitig [32]
- `serve`: Erstellen eines lokalen Servers zum Laden statischer Dateien [33]

Diese Abhängigkeiten werden in insgesamt sieben unterschiedlichen Skripten festgehalten und aufgerufen.

4.3.4 VORDEFINIERTES GESTALTUNGSSYSTEM

Mithilfe der Utility-Klassen von TailwindCSS lassen sich eine Vielzahl an CSS-Eigenschaften abbilden. Die Dokumentation von TailwindCSS teilt die Utilities in 15 Bereiche ein: Layout, Flexbox und Grid, Abstände, Größenangaben, Typografie, Hintergründe, Ränder, Effekte, Filter, Tabellen, Übergange und Animationen, Transformationen, Interaktivität, SVG und Barrierefreiheit. [34] Trotzdem werden dadurch noch nicht alle CSS-Eigenschaften abgebildet und die Entwickelnden sind demnach in ihrem Handeln eingeschränkt. Diese Einschränkung ist von TailwindCSS bewusst gewählt worden, da so auch Vorteile entstehen können. Somit werden die mitgelieferten Utilities nur auf die Wesentlichen beschränkt. Gleichzeitig werden dazu gezwungen, auf

28 Vgl. Robbins, 2018, S. 577
29 Vgl. npmjs, 2021
30 Vgl. npmjs, 2021
31 Vgl. npmjs, 2021.
32 Vgl. npmjs, 2021.
33 Vgl. npmjs, 2021.
34 Vgl. TailwindCSS, o. D.

saubere Werte aus einem bestehenden Größensystem zurückzugreifen. Dies sorgt für eine Vereinheitlichung innerhalb der Utility-Klassen und zu einer Konsistenz innerhalb des Designsystems.

Das mitgelieferte Größensystem für Breiten- und Höhenangaben (z. B. `w-20` und `h-24`), Angaben zu Positionen (z. B. `top-20` und `left-24`), Definitionen zu Verschiebungen (z. B. `-translate-x-12`) oder Utility-Klassen zur Definition von Abständen basieren alle auf dem gleichen „Spacing-System“ von TailwindCSS. Sie folgen alle einer bestimmten Namenskonvention. Das CSS-Framework liefert Angaben für Abstände und Größen in einem Bereich von 0 bis 24rem, dabei sind alle Angaben proportional zueinander und deren kleinste Einheit ist 0.25rem/4px. Somit entspricht eine Abstandsangabe mit der Zahl 1 einem entsprechend Wert von 0.25rem/4px. Die Utility-Klasse `mb-20` ist dementsprechend das Äquivalent zu der CSS-Deklaration `margin-bottom: 5rem` bzw. zu der Angabe `margin-bottom: 80px`. [35]

Das Designsystem basiert jedoch nicht nur auf einem durchziehenden Größensystem, sondern auch auf einer aufeinander abgestimmten Farbpalette mit ca. 90 Farb-Utilities. [36] Dadurch soll sichergestellt werden, dass alle Bereiche der Webseite eine identische Atmosphäre ausstrahlen. [37] Dabei muss allerdings berücksichtigt werden, dass bei einer häufigen und gleichen Anwendung des Designsystems ohne die Anpassung bestimmter Parameter die Persönlichkeit und Individualität einer Webseite verloren gehen kann.

4.3.5 UMSETZUNG DER GESTALTUNGSANGABEN

Um die Persönlichkeit und Individualität einer Gestaltung nicht aus den Augen zu verlieren, müssen bestimmte Parameter im Designsystem von TailwindCSS angepasst werden. Dafür müssen unter anderem die Gestaltungselemente der Marke, wie die Corporate Farben und die ausgewählten Schriftarten in der Entwicklung referenziert werden können. Die Designangaben werden innerhalb der Konfigurationsdatei von TailwindCSS individualisiert und angepasst. Diese Datei wird im Folgenden konkretisiert.

```
// tailwind.config.js
module.exports = {
  purge: [],
  darkMode: false, // or 'media' or 'class'
  theme: {
    extend: {},
  },
  variants: {
    extend: {},
  },
  plugins: [],
}
```

Abb. 4.17: Aufbau der Konfigurationsdatei, Quelle: https://tailwindcss.com/docs/configuration#creating-your-configuration-file (abgerufen am 25.08.2021)

Die Konfigurationsdatei von TailwindCSS trägt die Benennung `tailwind.config.js` und muss im Oberverzeichnis des Projektes liegen, um erkannt zu werden. Diese enthält nach ihrer

35 Vgl. TailwindCSS: Customizing Spacing - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/customizing-spacing#default-spacing-scale (abgerufen am 03.09.2021).
36 Vgl. Rappin, 2021, S. 19.
37 Vgl. Lanciaux, 2021, S. 68.

```
theme: {
  colors: {
    beige: {
      lightest: '#faf9f7',
      light: '#e7e4db',
    },
    red: '#9d887a',
    brown: '#b3a898',
    green: '#8e8b7c',
    dark: '#5f6053',
  },
  fontFamily: {
    'ivypresto-display': ['ivypresto-display', 'serif'],
    'adobe-caslon-pro': ['adobe-caslon-pro', 'serif'],
    'montserrat-light': ['montserrat-light', 'serif'],
    'montserrat-medium': ['montserrat-medium', 'serif'],
  },
  fontSize: {
    'smallest': '0.625rem',
    'smaller': '0.75rem',
    'small': '0.875rem',
    'base': '1rem',
    'large': '1.125rem',
    'larger': '1.25rem',
    'largest': '1.875rem',
    'huge': '2.188rem',
    'huger': '3.125rem',
    'hugest': '4.375rem',
  },
  letterSpacing: {
    'wide': '0.15em',
    'standard': '0.12em',
    'tight': '0.05em',
  },
  lineHeight: {
    'standard': '1.5',
    'tight': '1.2',
    'none': '0',
  },
```

Abb. 4.18: Werteüberschreibungen, Quelle: Eigene Darstellung

Generierung bereits eine Grundstruktur (vgl. Abb. 4.17). [38] Im weiteren Verlauf des Textes wird näher auf die einzelnen Bereiche der Grundstruktur eingegangen.

Innerhalb des `theme` Bereichs der Konfigurationsdatei können alle visuellen Gestaltungsparameter angepasst werden. Dabei wird unterschieden, ob die selbst erstellten Definitionen die vordefinierten, bereits mitgelieferten Definitionen überschreiben oder erweitern sollen. Für eine Erweiterung der Daten müssen die Parameter innerhalb des Bereiches `extend` definiert werden und für eine Überschreibung außerhalb. Dabei können bestimmte Utility-Klassen mithilfe sogenannter `keys` angesprochen werden. [39] Exemplarisch wäre der `key` für die Erweiterung oder Überschreibung von Breakpoints der Begriff `screen`.

Alle vorgenommenen Definitionen zur Überschreibung von vordefinierten Utilities werden in Abbildung 4.18 dargestellt. Wie in dieser Abbildung zu erkennen ist, werden alle Farbwerte mit den Corporate Farben der Marke ersetzt, die individuellen Schriftarten werden hinzugefügt, es werden neue Schriftgrößen definiert und weitere typografische Attribute, wie Zeilenabstand und Laufweite, werden neu gesetzt. Da die vordefinierten Werte für diese Utilities nicht verwendet werden, ist eine Überschreibung dieser Werte sinnvoll.

```
  extend: {
    spacing: {
      '16.6/100': '6%',
      '8.3/100': '12%',
      '3/10': '30%',
    },
    outline: {
      green: '2px solid #8e8b7c'
    },
    height: {
      '112': '28rem',
    },
    maxHeight: {
      '152': '38rem',
    },
    minHeight: {
      '384': '24rem',
    },
    zIndex: {
      '-1': '-1',
    },
    inset: {
      '3/20': '15%',
      '1/5': '20%',
      '16.6/100': '6%',
    },
```

Abb. 4.19: Werteerweiterungen, Quelle: Eigene Darstellung

Alle Definitionen, die die vordefinierten Utility-Bereiche nur erweitern, werden in Abbildung 4.19 angezeigt. Dabei handelt es sich hauptsächlich um Größenangaben, Abstände oder Positionierungsangaben mit Prozentwerten. Bei der Erweiterung von schon bestehenden Utility-Sammlungen wird weiterhin auf die Namenskonvention geachtet (vgl. *4.3.4 Vordefiniertes Gestaltungssystem*). Dabei entspricht die linksstehende Angabe die Benennung der Utility und die rechtsstehende Angabe den zugewiesenen Wert. Aus Abbildung 4.19 geht somit `h-112` als mögliche Utility hervor.

38 Vgl. TailwindCSS: Configuration - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/configuration (abgerufen am 25.08.2021).
39 Vgl. TailwindCSS, o. D.

4.3.6 REALISIERUNG VON CROSS-BROWSER-RENDERING

Im Folgenden soll untersucht werden, wie TailwindCSS Cross-Browser-Rendering realisiert und wie es um ihre Browserkompatibilität steht.

4.3.6.1 PREFLIGHT

Für die Anpassung der verschiedenen User-Agent-Stylesheets untereinander bringt das zu untersuchende CSS-Framework eine bereits eingebaute Lösung mit. Diese eingebaute Lösung wird als „Preflight“ bezeichnet. „Preflight is a set of base styles for Tailwind projects that are designed to smooth over cross-browser inconsistencies and make it easier for you to work within the constraints of your design system.“ [40] Die Anpassungen befinden sich innerhalb der `@base`-Ebene und werden nicht angewendet, sobald diese Ebene nicht importiert wird oder das „Preflight“ in der Konfigurationsdatei deaktiviert worden ist. [41]

Laut der Dokumentation basieren die Anpassungen von „Preflight“ auf der Abhängigkeit `modern-normalize`. In dieser Abhängigkeit werden nur bestimmte User-Agent-Stylesheets von modernen Browsern, wie Safari, Chrome und Firefox, mit einbezogen. [42] Um einen tieferen Einblick in das „Preflight“ zu geben, liefert TailwindCSS auf deren Webseite eine kleine Auflistung mit den wichtigsten Anpassungen. Es werden beispielsweise die standardisierten `margins` von mehreren HTML-Elementen gelöscht, Überschriften werden in ihrer Schriftgröße und in ihrer Schriftdicke zurückgesetzt, Listen haben keine Aufzählungspunkte oder Abstände mehr, Medien wie Bilder und Videos sind im normalen Dokumentenfluss mit inbegriffen, Ränder sind zurückgesetzt und Buttons erhalten beim Fokussieren des Elements eine Außenlinie. [43] Obwohl „Preflight“ als Sammlung von Deklarationen zur Anpassung von Browser-Deklarationen vermarktet wird, werden hier viele Definitionen zurückgesetzt bzw. überschrieben, sodass es keine alternative visuelle HTML-Auszeichnung mehr gibt.

4.3.6.2 BROWSERKOMPATIBILITÄT

Da auch einzelne Abhängigkeiten unterschiedliche Browserkompatibilitäten aufweisen können, soll auch die Kompatibilität der Abhängigkeit `tailwindcss`

40 TailwindCSS: Preflight - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/preflight (abgerufen am 30.08.2021).
41 Vgl. TailwindCSS: Preflight - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/preflight#disabling-preflight (abgerufen am 30.08.2021).
42 Vgl. Sindresorhus: GitHub - sindresorhus/modern-normalize: Normalize browsers' default style, in: GitHub, o. D., https://github.com/sindresorhus/modern-normalize#browser-support (abgerufen am 30.08.2021).
43 Vgl. TailwindCSS, o. D.

genauer ergründet werden. Seitdem TailwindCSS auf Version 2.0 aktualisiert wurde, werden alle Versionen des Browsers *Internet Explorer* (**IE**) nicht mehr unterstützt. Zudem gibt es eine Anmerkung in der Dokumentation von TailwindCSS, dass bestimmte Funktionen, wie `focus–visible`, nicht in allen modernen Browsern abgebildet werden können. [44]

Nicht nur bei Abhängigkeiten kann die Kompatibilität der Browser unterschiedlich sein, sondern auch bei CSS-Eigenschaften. Bestimmte CSS-Eigenschaften können von bestimmten Browsern nur mit vorangestellten Herstellerpräfixen korrekt interpretiert werden. [45]

Diese Herstellerpräfixe können mit dem `postcss` Plugin `autoprefixer` automatisch hinzugefügt werden. Dies spart den Entwickelnden viel Zeit zu erörtern, welche CSS-Eigenschaften ein Herstellerpräfix benötigen und welche nicht. Sobald TailwindCSS innerhalb der CLI aufgerufen wird, wird auch `autoprefixer` verwendet. Somit muss bei einer Verwendung von TailwindCSS innerhalb der CLI auch `autoprefixer` installiert sein. [46]

4.3.7 RESPONSIVE WEBDESIGN & MOBILE FIRST

In Abschnitt *4.3.2.2 Umsetzung von Responsivität* wurden bereits die standardisierten Breakpoints aufgelistet. Dabei ist die Funktionsweise der vordefinierten Breakpoints und der Media-Queries an der Mobile First Strategie orientiert. [47] Es wurde jedoch darauf verzichtet, die vordefinierten Breakpoints aus Abbildung 4.14 zu erweitern oder zu überschreiben, da alle Ansichten mit den mitgelieferten Mitteln umgesetzt werden konnten. Ergänzend dazu soll ermittelt werden, wie sich die vordefinierten Breakpoints auf die individuelle Gestaltung auswirken.

Wie in der Umsetzung mit handgeschriebenen CSS wurde als Ausgangspunkt der Entwicklung ein Endgerät mit einer Mindestbildschirmbreite von 320 Pixel gewählt.

4.3.8 PROGRESSIVE ENHANCEMENT & JAVASCRIPT

Im Gegensatz zu der Umsetzung mit CSS ist es in dieser Herangehensweise deutlich schwieriger, alle Funktionen nur mithilfe des CSS-Frameworks umzusetzen und nicht auf JS zurückzugreifen. Der Grund liegt darin, dass

44 Vgl. TailwindCSS: Browser Support - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/browser-support (abgerufen am 30.08.2021).
45 Vgl. MDN Web Docs: Herstellerpräfix - Glossar | MDN, in: MDN Web Docs, 29.08.2021, https://developer.mozilla.org/de/docs/Glossary/Vendor_Prefix (abgerufen am 30.08.2021).
46 Vgl. TailwindCSS, o. D.
47 Vgl. TailwindCSS: Responsive Design - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/responsive-design#overview (abgerufen am 19.08.2021).

Verknüpfungen von Selektoren mittels Kombinatoren in TailwindCSS nicht abgebildet werden können. Der Wirkungsraum der Utility-Klassen beschränkt sich nur auf das jeweilige zugewiesene HTML-Element. [48]

Wie bereits in Abschnitt *4.2.5 Progressive Enhancement* untersucht, lässt sich bei der Ansicht des mobilen Menüs JavaScript umgehen, indem Checkboxen eingesetzt werden, die mithilfe von CSS abgefragt werden können. Diese Abfrage basiert allerdings auf einer Verbindung unterschiedlicher Selektoren, zusammen mit einem Kombinator (vgl. Abb. 4.20). Da diese Abfrage in TailwindCSS nicht abgebildet werden kann, muss dafür JavaScript hinzugezogen werden.

```
/* CSS */
#navigation-open:not(:checked) {
  ~ .navigation {
    display: none;
  }

  @media (min-width: $screen-large) {
    ~ .navigation {
      display: block;
    }
  }
}
```

Abb. 4.20: Abfrage zur Sichtbarkeit des mobilen Menüs, Quelle: Eigene Darstellung

Dementsprechend wird in einer JS-Datei die Interaktion des Anwendenden mit dem Menü-Icon abgefragt und die entsprechende Utility-Klasse `hidden` (in CSS: `display:none`) wird dem Menü-Element hinzugefügt oder weggenommen. Dies kann zu einem Problem führen, sobald Anwendende der Webseite JavaScript nur bedingt oder gar nicht ausführen können, da dadurch das Menü in der mobilen Ansicht nicht erreichbar ist (vgl. *4.2.5 Progressive Enhancement*).

Neben der JS-Datei für das Menü werden zwei weitere JS-Dateien verwendet, die auch in der Umsetzung mit CSS eingesetzt wurden. Die eine Datei reguliert die Sichtbarkeit der Produktauflistung, währenddessen die andere Datei das Scrolling zu den verknüpften Inhaltselementen aus der Navigation optimiert (vgl. *4.2.. Progressive Enhancement*).

4.3.9 CODE-QUALITÄT

Auch in CSS-Frameworks, wie TailwindCSS, hat die Code-Qualität einen hohen Stellenwert, da die Vorteile, die zuvor im Abschnitt zu handgeschriebenen CSS erwähnt wurden, auch hier zutreffen. (vgl. *4.2.6 Code-Qualität*).

4.3.9.1 VERHINDERUNG VON REDUNDANTEM CODE

Wie zuvor in Abschnitt *4.3.2.4. Erstellung von Komponenten* veranschaulicht, können ohne den Einsatz von einem CMS oder JS-Frameworks keine

48 Vgl. Rappin, 2021, S.IX

wiederverwendbaren Codefragmente bzw. Partials genutzt werden. Aus diesem Grund ist es in dieser Anwendung besonders schwer, redundanten Code vorzubeugen und zu verhindern.

Um bereits vor der Entwicklung Wiederholungen zu verhindern und den Code wartbarer zu gestalten, gibt es die Möglichkeit, allgemeinen HTML-Elementen bestimmte Utility-Klassen zuzuweisen. Dies ist vergleichbar mit den CSS-Elementselektoren. Diese Definitionen werden in der `tailwind.css` Datei vorgenommen, indem diese innerhalb der `base`-Ebene definiert werden (vgl. Abb. 4.21).

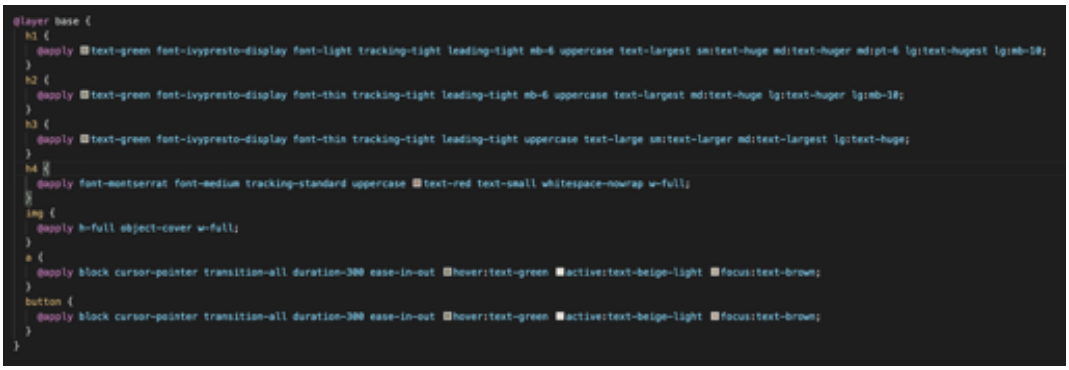


Abb. 4.21: Sammlungen an Utility-Klassen für bestimmte HTML-Elemente, Quelle: Eigene Darstellung

Zusätzlich zu den Elementselektoren können auch Komponenten bzw. wiederverwendbare Sammlungen an Utility-Klassen eingesetzt werden (vgl. *4.3.2.4 Erstellung von Komponenten*). Abbildung 4.22 zeigt alle Utility-Sammlungen, die während dieser Umsetzung in der `components`-Ebene hinzugefügt wurden. Mittels dieser Sammlungen sollen redundante Utility-Klassen umgangen und das hin und her kopieren von Utilities verhindert werden, da dadurch leicht Fehlerquellen und schlecht wartbarer Quellcode entstehen können.

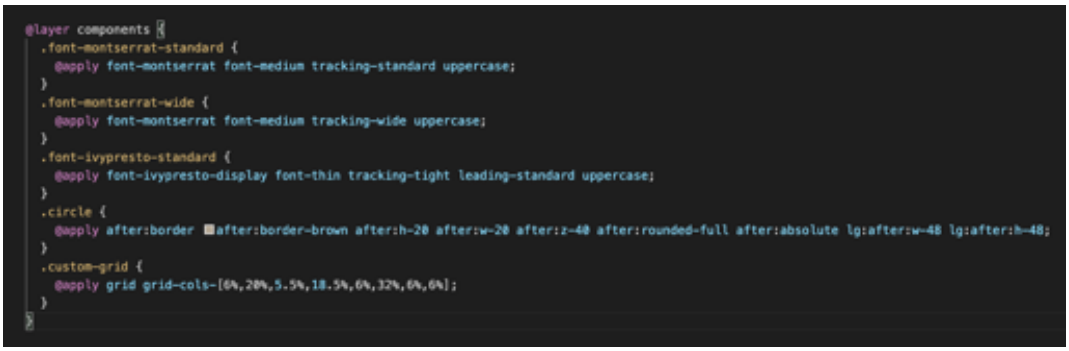


Abb. 4.22: Sammlung an Utility-Klassen mit Klassennamen, Quelle: Eigene Darstellung

Durch die Verwendung von Komponenten wird zudem die Länge der Zeichen innerhalb einer HTML-Klasse deutlich verringert. Zur Verdeutlichung wird dafür ein `<picture>`-Element aus der Anwendung verwendet, welches

einmal mit und einmal ohne Komponente gezeigt wird. Durch die langen Verkettungen ist der Code sehr unübersichtlich.

Beispiel mit circle-Komponente:

```
<picture class="relative col-start-1 col-end-7 row-start-1 row-end-2
h-40 mb-6 w-full sm:h-60 lg:col-start-6 lg:col-end-7 lg:h-4/5 lg:mb-0
circle after:-top-10 after:-right-10 lg:after:hidden">
```

Beispiel ohne circle-Komponente:

```
<picture class="relative col-start-1 col-end-7 row-start-1 row-end-2
h-40 mb-6 w-full sm:h-60 lg:col-start-6 lg:col-end-7 lg:h-4/5 lg:mb-0
after:border after:border-brown after:h-20 after:w-20 after:z-40
after:rounded-full after:absolute lg:after:w-48 lg:after:h-48 after:-
top-10 after:-right-10 lg:after:hidden">
```

4.3.9.2 SYNTAX & STRUKTUR

Leider gibt es noch keine Hilfsmittel, die die Entwickelnden dabei unterstützen, eine Ordnung und Struktur innerhalb der verwendeten Utility-Klassen zu bewahren. Das einzig verwendete Hilfsmittel für die Umsetzung mit TailwindCSS, ist die im Editor Visual Studio Code vorhandene Erweiterung „Tailwind CSS IntelliSense". Diese Erweiterung gibt Vorschläge zu den eingetippten Utility-Klassen und gibt visuelle Hinweise bei vorkommenden Dopplungen. [49]

4.3.9.3 VALIDATOREN

Die in Abschnitt *4.2.6.3 Validatoren* vorgestellten Validatoren für HTML und CSS sollen auch für diese Herangehensweise angewendet werden.

Nu HTML Checker:
Als Validator für HTML wird wieder der „Nu HTML Checker" hinzugezogen. Abbildung 4.23 zeigt das Ergebnis, welches besagt, dass keine Fehler in der HTML-Struktur gefunden werden konnten.

49 Vgl. TailwindCSS: Editor Support - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/editor-support#intelli-sense-for-vs-code (abgerufen am 31.08.2021).

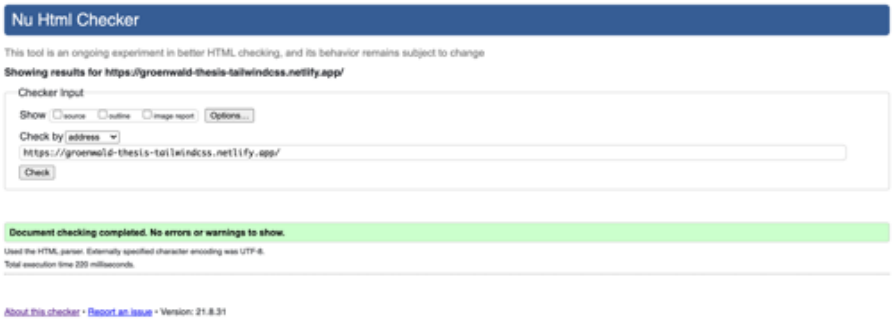


Abb. 4.23: Ergebnis des Nu HTML Checkers, Quelle: Screenshot des Nu HTML Checkers (abgerufen am 31.08.2021)

Jigsaw-Validator:
Das Ergebnis dieses Validators ist in dieser Umsetzung nicht allzu aussagekräftig, da nur die von `postcss` generierte CSS-Datei und die selbst erstellte CSS-Datei `site.css` überprüft werden. Das Ergebnis des Validators für CSS besagt, dass keine Fehler in der CSS-Syntax gefunden werden konnten (vgl. Abb. 4.24).

Abb. 4.24: Ergebnis des Jigsaw-Validators
Quelle: Screenshot des Jigsaw-Validators (abgerufen am 31.08.2021)

4.3.10 PERFORMANCE IM BROWSER

Um nachher einen guten Vergleich zwischen den unterschiedlichen Herangehensweisen an CSS ziehen zu können, wird auch für diese Umsetzung deren Performance im Browser untersucht.

4.3.10.1 OPTIMIERUNGSMÖGLICHKEITEN

Optimierung der Bilder:
In dieser Umsetzung wurde wie in der vorherigen Umsetzung, die gleiche Strategie zur Optimierung der Bilder verfolgt. Aufgrund dessen ist die HTML-Struktur in diesem Bereich bei beiden Herangehensweisen identisch (vgl. *4.2.7.1 Optimierungsmöglichkeiten*).

JavaScript-Dateien:
Auch in der Optimierung der JS-Dateien wurde sich auf die gleiche Abhängigkeit wie in der Umsetzung mit CSS berufen. Bei dieser Herangehensweise werden jedoch nicht nur die JavaScript-Dateien, sondern

auch die CSS-Dateien komprimiert. Die JavaScript-Dateien werden ebenfalls asynchron und parallel geladen (vgl. *4.2.7.1 Optimierungsmöglichkeiten*).

Löschen von nicht verwendetem Quellcode:
Das Löschen von nicht genutztem Quellcode ist besonders bei CSS-Frameworks wie TailwindCSS wichtig, da dieser bereits viele vordefinierte Utility-Klassen mitliefert, die nicht unbedingt alle einen Anwendungsfall liefen. Ohne eine Beschränkung der Utilities wäre der schlussendliche Quellcode überladen mit nicht genutzten und dadurch unnötigen Inhalten. [50]

Abb. 4.25: Dateien für `purgecss`,
Quelle: Eigene Darstellung

TailwindCSS hat für dieses Problem eine eingebaute Lösung. Mit der bereits bekannten Abhängigkeit `purgecss` können alle nicht verwendeten Utilities aus der CSS-Datei gelöscht werden, die von TailwindCSS generiert wurden. Dafür wird eine Schnittmenge gebildet, indem ein Vergleich zwischen den verwendeten und allen generierten Utility-Klassen stattfindet. Um diese Funktion zu aktivieren, müssen die Dateien, die Utility-Klassen beinhalten, in der Konfigurationsdatei von TailwindCSS angegeben werden (vgl. Abb. 4.25). [51]

4.3.10.2 WERKZEUGE ZUM TESTEN DER PERFORMANCE

Wie die vorherige Umsetzung, wird auch diese Umsetzung online mit unterschiedlichen Werkzeugen auf ihre Performance getestet. Dafür wird das Ergebnis wieder mit dem Hosting-Service Netlify auf einer Domain bereitgestellt. Die Domain für die Umsetzung mit TailwindCSS lautet: https://groenwald-thesis-tailwindcss.netlify.app/

Pagespeed Insights:
Die Ergebnisse des „Google Pagespeed Tools“ sind in Abbildung 4.26 für mobile Endgeräte und in Abbildung 4.27 für Desktop festgehalten. Auch hier kann das Ergebnis zwischen unterschiedlichen Tests stark schwanken. Der Wert für Mobil liegt bei 91% und für Desktop bei 99%.

50 Vgl. Hahn, 2020, S. 31.
51 Vgl. Rappin, 2021, S. 71.

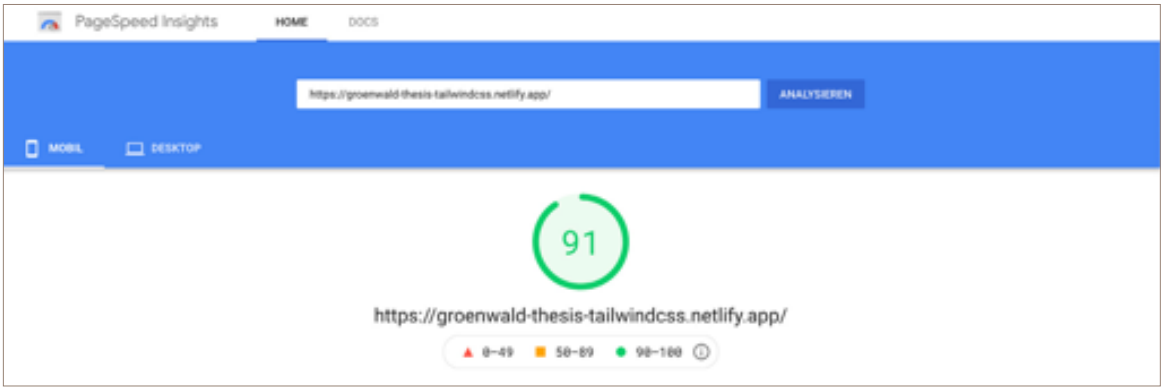


Abb. 4.26: PageSpeed Insights Ergebnis Mobil, Quelle: Screenshot des Pagespeed Insights (abgerufen am 01.09.2021)

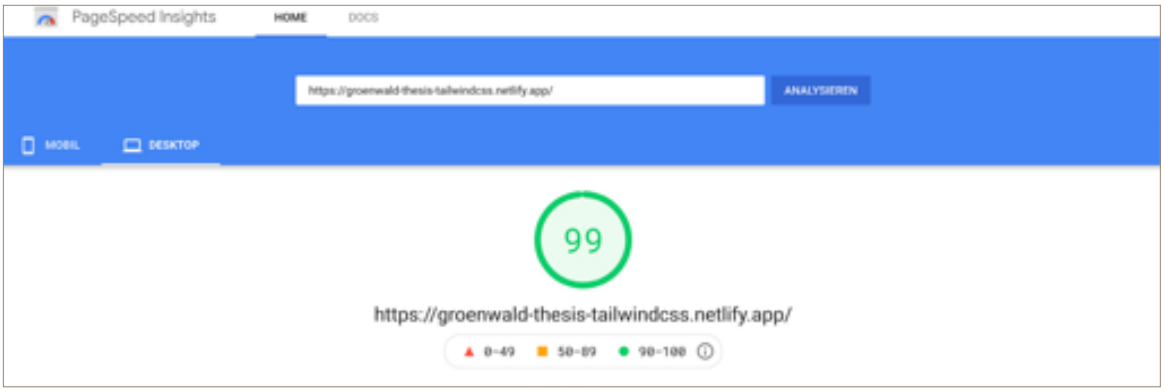


Abb. 4.27: PageSpeed Insightes Ergebnis Desktop, Quelle: Screenshot des Pagespeed Insights (abgerufen am 01.09.2021)

Pingdom:
Auch das Werkzeug „Pingdom Website Speed Test“ wird wieder angewendet. Das Ergebnis wird in Abbildung 4.28 festgehalten. Laut diesem Test beträgt die Datenmenge für die Umsetzung mit TailwindCSS 647.2 Kilobyte, die Ladezeit beträgt 300 Millisekunden und der Server muss 21 Abfragen verarbeiten.

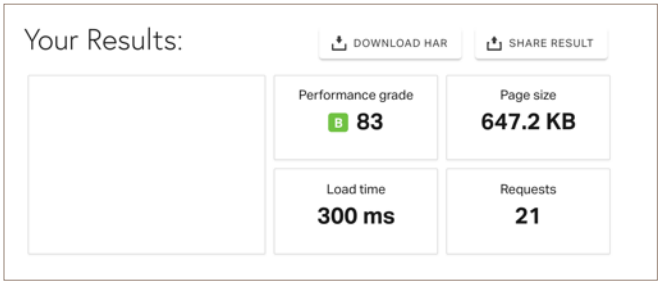


Abb. 4.28: Ergebnis des Pingdom Speed Tests
Quelle: https://tools.pingdom.com/#5f06f2159e000000
(abgerufen am 27.09.2021)

Chrome Entwickler-Ansicht: Network
Um mehr Informationen aus den Datenmengen zu erhalten, wird wieder das Wasserfalldiagramm aus dem Network-Bereich der Entwickler-Ansicht des Browsers Chrome hinzugezogen (vgl. Abb. 4.29). Die Reihenfolge, welche Dateien als Erstes und welche als Letztes geladen werden, hat sich im Vergleich zur CSS-Umsetzung nicht verändert.

Das CSS-Stylesheet weist hier eine Dateigröße von 4,8 Kilobyte und das gebündelte JavaScript eine Dateigröße von 607 Byte auf.

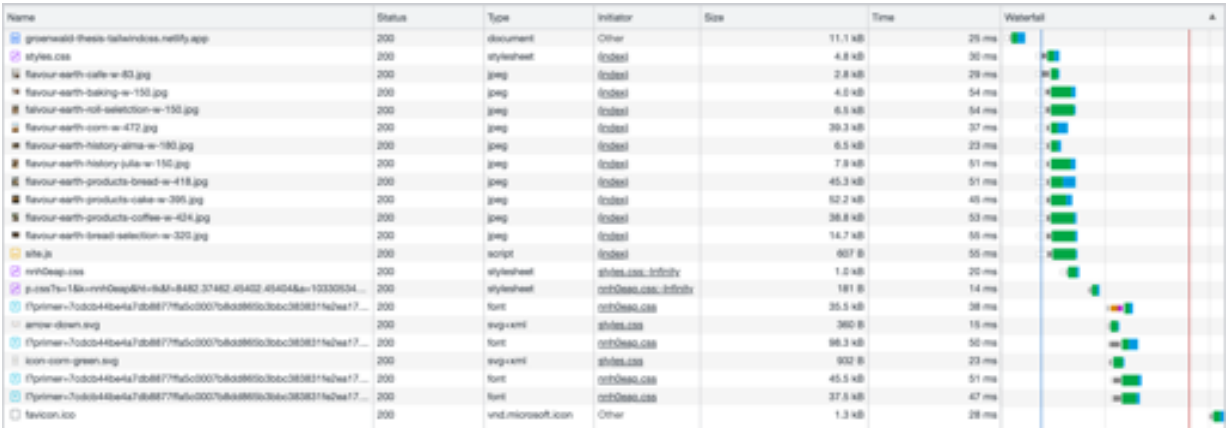


Abb. 4.29: Wasserfalldiagramm im Network-Bereich,
Quelle: https://6145c1a3b509460007050e87--groenwald-thesis-tailwindcss.netlify.app/ (abgerufen am 27.09.2021)

4.3.11. DATEIGRÖSSEN

In diesem Abschnitt sollen wieder die Dateigrößen optimierter und nicht optimierter CSS und JS-Dateien miteinander verglichen werden. Im vorherigen Abschnitt *4.3.10.2 Werkzeuge zum Testen der Performance* wurden bereits die Dateigrößen der optimierten Dateien vorgestellt. Zur Erhebung der nicht optimierten Dateigrößen wird ebenfalls ein Wasserfalldiagramm hinzugezogen (vgl. Abb. 4.30). Diese Abbildung beinhaltet eine Version der Umsetzung ohne eine Optimierung der Dateien. Dementsprechend kann an dieser Abbildung abgelesen werden, dass das nicht optimierte Stylesheet eine Dateigröße von 105 Kilobyte und die nicht optimierte JavaScript-Datei eine Dateigröße von 936 Byte aufweist.

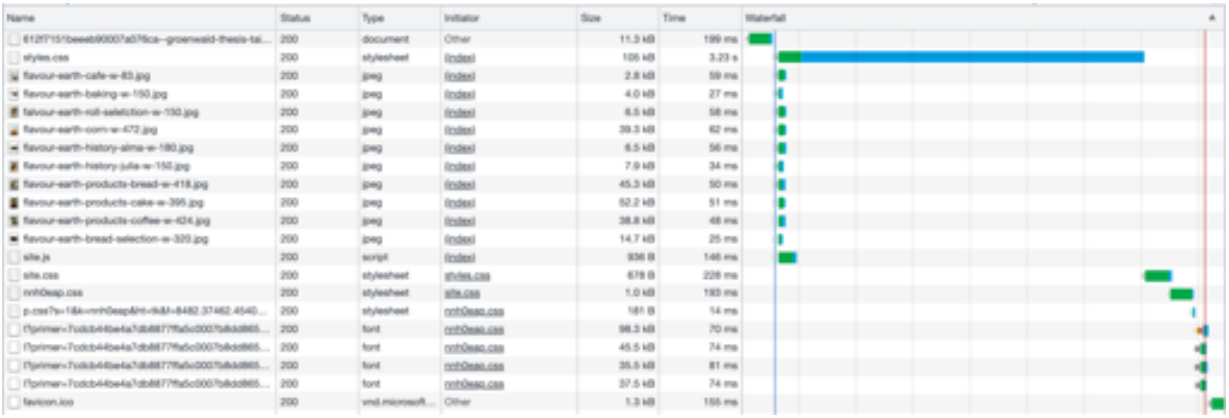


Abb. 4.30: Wasserfalldiagramm mit nicht optimierten Dateien
Quelle: https://612f7151beeeb90007a076ca--groenwald-thesis-tailwindcss.netlify.app/ (abgerufen am 01.09.2021)

Zur Gegenüberstellung der nicht optimierten und optimierten Dateigrößen wurde die folgende Tabelle 4.2 erstellt.

	optimiert	nicht optimiert	Differenz
styles.css	105.000 Byte	4.800 Byte	100.200 Byte
site.js	938 Byte	607 Byte	331 Byte

Tabelle 4.2: Vergleich Dateigrößen TailwindCSS, Quelle: Eigene Darstellung

Diese Tabelle zeigt, dass durch die Optimierung mit `purgecss` und die Komprimierung mit `minify`, die Dateigröße des CSS-Quellcodes um 100,2 Kilobyte verringert werden konnte. Die Komprimierung zeigt auch im JavaScript-Quellcode eine Auswirkung, da dieser um 331 Byte minimiert werden konnte.

4.3.12 SCHWIERIGKEITEN BEI DER UMSETZUNG

Die Schwierigkeiten bei der Umsetzung mit TailwindCSS sind durch fehlende Funktionen oder CSS-Eigenschaften entstanden, die mithilfe des Utility-Klassen basierten CSS-Frameworks nicht abgebildet werden konnten. Es gibt eine Sammlung an Deklarationen, die mithilfe einer weiteren unabhängigen CSS-Datei erstellt werden mussten, damit die individuelle Gestaltung wie gewünscht umgesetzt werden konnte. Diese CSS-Datei hat die Bezeichnung `site.css` und wird vor der `base`-Ebene in die `tailwind.css` Datei importiert. Die folgende Aufzählung beschreibt den Inhalt dieser Datei.

1.Problem: Import der Schriften
Sobald eine individuelle Schrift mit TailwindCSS verwendet werden soll, muss diese mithilfe der CSS-Eigenschaft `@font-face` in einer weiteren CSS-Datei definiert und geladen werden. Falls die Dateien der verwendeten Schriftschnitte nicht lokal vorhanden sind, sondern online zur Verfügung gestellt werden (wie in dieser Umsetzung mit Typekit von Adobe Fonts), müssen diese mit einem Link referenziert werden. Erst dann wird in der Konfigurationsdatei von TailwindCSS eine Referenz zur korrekten `fontFamily` hergestellt. [52]

2. Problem: Weiches Scrolling
Um das Scroll-Verhalten zu verbessern, kann dem `<html>`-Element eine CSS-Eigenschaft für ein weiches und glattes Scrolling zugewiesen werden. Dafür gibt es keine entsprechende Utility-Klasse.

3. Problem: Änderung der Laufrichtung von Text
Mit der CSS-Eigenschaft `writing-mode` lässt sich die Laufrichtung eines Textes ändern. Dafür gibt es in TailwindCSS keine entsprechende Utility-Klasse. Diese CSS-Eigenschaft wurde für die seitliche Darstellung des Entstehungsjahres auf mobilen Endgeräten benötigt.

52 Vgl. tailwindlabs: How to add Custom Fonts to a Website using TailwindCss? · Discussion #2060 · tailwindlabs/tailwindcss, in: GitHub, o. D., https://github.com/tailwindlabs/tailwindcss/discussions/2060 (abgerufen am 01.09.2021).

4.Problem: Ausschneiden von Bildern, Elementen oder Illustrationen
Auch für die folgende Funktion gibt es keine Abbildung in TailwindCSS. Mithilfe der CSS-Eigenschaft `clip-path` lassen sich Bilder, Elemente oder Illustrationen nach einem angegebenen Pfad entsprechend beschneiden. [53] In dem Design wurde dies für eine individuelle Darstellung von Bildern (z. B. zweites Produktbild in der Produktauflistung) verwendet. Um das umsetzen zu können, musste ebenfalls die CSS-Datei hinzugezogen werden.

5.Problem: Hintergrundbilder
Wenn ein Pfeil oder andere Icons hinter einem HTML-Element eingefügt werden sollen, kann dies entweder mit einem weiteren HTML-Element umgesetzt werden oder der/die Entwickelnde erstellt ein Pseudoelement. Falls ein Pseudoelement genutzt werden soll, wird das Icon als Hintergrundbild in das Pseudoelement mithilfe der CSS-Eigenschaft `background-image` gesetzt. [54] Dies wurde in der Entwicklung mehrfach angewendet, wie z. B. beim `<footer>` als hinterlegtes Muster oder beim Pfeil der Produktauflistung. Allerdings kann auch dies nicht nur mit TailwindCSS umgesetzt werden, da es dafür keine entsprechende Utility-Klasse gibt.

4.3.13 VORKENNTNISSE

In diesem Abschnitt wird untersucht, ob und welche Vorkenntnisse innerhalb des Bereiches der Frontend-Entwicklung benötigt wurden, um die individuelle Gestaltung mit TailwindCSS umsetzen zu können. Zudem soll die Frage beantwortet werden, wie viel Zeit diese Umsetzung in Anspruch genommen hat.

4.3.13.1 BENÖTIGTE VORKENNTNISSE

Da die HTML-Struktur ähnlich bis identisch wie zur Umsetzung mit reinem CSS ist, wurden auch hier fundierte Kenntnisse über HTML benötigt, um eine semantisch korrekte Auszeichnung der Inhalte zu erhalten. Darüber hinaus wurde ein gewisses Vorwissen an CSS-Eigenschaften vorausgesetzt, da deren Nutzen und Auswirkungen bei der Entwicklung mit TailwindCSS bekannt sein müssen. Sobald die Entwickelnden wissen, mit welcher CSS-Eigenschaft welches Ergebnis erreicht wird, kann mithilfe der Dokumentation über TailwindCSS die entsprechende Utility-Klasse herausgefunden werden, die daraufhin nur auf das passende HTML-Element angewendet werden muss. Alle weiteren Funktionsweisen von CSS, wie Verschachtelungen mithilfe

53 Vgl. MDN Web Docs: Clip-path - CSS | MDN, in: MDN Web Docs, 01.09.2021, https://developer.mozilla.org/de/docs/Web/CSS/clip-path (abgerufen am 01.09.2021).
54 Vgl. MDN Web Docs: Background-image - CSS | MDN, in: MDN Web Docs, 01.09.2021, https://developer.mozilla.org/de/docs/Web/CSS/background-image (abgerufen am 01.09.2021).

von Selektoren und Kombinatoren oder explizite Berechnungen und Programmierschleifen, die mit Sass erreicht werden können, können mit TailwindCSS nicht abgebildet werden. Dementsprechend muss darüber auch kein Vorwissen vorhanden sein.

Während der Entwicklung einer Webseite mit TailwindCSS wird JavaScript öfter verwendet, um Lösungen für Entwicklungsschwierigkeiten zu entwickeln, die mit TailwindCSS nicht gelöst werden können. Dadurch ist hier ein gewisses Maß an Vorkenntnissen über die Funktionsweise von JavaScript nicht unbedingt nötig, aber hilfreich.

4.3.13.2 BENÖTIGTE ZEIT ZUR UMSETZUNG

Für die Ermittlung der aufgewendeten Zeit der Entwicklung wurde ebenfalls TMetric hinzugezogen. Die gezählte Zeit in TMetric beträgt 18 Stunden und 44 Minuten.

4.4 BOOTSTRAP

Die dritte Umsetzung der individuellen Gestaltung der Webseite des veganen Cafés soll mit dem Komponenten basierten CSS-Framework Bootstrap umgesetzt werden. Es wird vermarktet mit dem folgenden Satz: „Quickly design and customize responsive mobile-first sites with Bootstrap, the world's most popular front-end open source toolkit, featuring Sass variables and mixins, responsive grid system, expensive prebuilt components, and powerful JavaScript plugins." [1]

4.4.1 BOOTSTRAP ALS KOMPONENTEN BASIERTES CSS-FRAMEWORK

Bootstrap ist ein CSS-Framework, das von den beiden Entwicklern Mark Otto und Jacob Thornton ursprünglich für den Mikrobloggingdienst Twitter entwickelt wurde. Im Jahr 2011 wurde Bootstrap als Open-Source Projekt für die Öffentlichkeit zur Verfügung gestellt. [2] Bei der Vermarktung des CSS-Frameworks wurde und wird weiterhin stark betont, dass es sich hierbei um ein responsives, Mobile First vertretendes Framework handelt. [3] Da im Jahr 2007 das erste mobile Endgerät mit einem Webbrowser und Touchscreen vorgestellt wurde, fand das Thema des responsiven Webdesigns immer mehr Zuwachs und so kam die Veröffentlichung von Bootstrap mit seinen neuartigen, responsiven Komponenten sehr gelegen. [4] Dies spiegelt sich auch in der Statistik von „State of CSS 2020" wieder. Insgesamt haben 86% der 11.492 Befragten angegeben, dass sie Bootstrap verwenden, womit dies das meistverwendete CSS-Framework ist. Gleichzeitig ist dies auch das bekannteste CSS-Framework, da alle Teilnehmer der Befragung angeführt haben, über die Existenz dieses Frameworks in Kenntnis zu sein. [5] Aus diesen Gründen wurde Bootstrap als passender Repräsentant des Komponenten basierten Ansatzes gewählt.

4.4.2 FUNKTIONSWEISE

Die allgemeine Funktionsweise von Komponenten wurde bereits im Abschnitt *2.4.1.2 Nutzen von Komponenten* angesprochen. In diesem Abschnitt wird untersucht, welche Komponenten genau von Bootstrap mitgeliefert werden und wie sie verwendet werden können.

1 Otto, Mark/Jacob Thornton: Bootstrap, in: GetBootstrap, o. D., https://getbootstrap.com/ (abgerufen am 04.09.2021).
2 Vgl. Barker, 2014, S. 131.
3 Vgl. Otto/Thornton, o. D.
4 Vgl. Hahn, 2020, S. 31.
5 Vgl. State Of CSS, o. D.

4.4.2.1 FUNKTIONSWEISE VON BOOTSTRAP

Tom Barker fasst in seinem Buch „High Performance Responsive Design“ die Funktionsweise von Bootstrap wie folgt zusammen: „Bootstrap’s base installation comes with predefined CSS and JavaScript to implement a set of frontend components that have responsiveness built in to them. These components include buttons, tabs, progress bars, gird systems, patterns for alerts, and even specific page layouts.“ [6]

Die Komponenten basieren laut der Dokumentation von Bootstrap auf einer sogenannten „base-modifier“-Nomenklatur. Dabei werden die grundlegenden Deklarationen einer Komponente in einer bezeichnenden Klasse zusammengefasst, wie z. B. `btn`. Explizite Varianten einer Komponente werden in einer zusätzlichen modifizierten Klasse angegeben, wie beispielsweise `btn–dark`. Diese zusätzlichen Varianten werden automatisch generiert, indem Bootstrap im Hintergrund mittels Sass auf Sass-Maps, also Sammlungen an definierten Variablen, wiederholt zugreift und daraus bestimmte Komponentenvarianten generiert. [7]

4.4.2.2 LISTE AN KOMPONENTEN

Innerhalb der aktuellen Version von Bootstrap (Version 5.1.) sind insgesamt 24 Komponenten aufgelistet. Die Komponenten sind die Folgenden: ausklappbares Akkordeon, Push-Benachrichtigungen, Anzeige von Label, Breadcrumbs, Buttons, Button-Gruppe, Cards, Karussell, Schließen-Button, ausklappbare Inhalte, Dropdowns, Listen, Popup-Overlays, Navigationsleiste, ausklappbare Navigation, Sidebars, Paginations, Platzhalter, Popover, Ladebalken und Ladeanimationen, automatisch aktualisierende Sidebar und Popup-Beschreibungen. [8]

Anhand dieser Liste lässt sich erkennen, mit welchem Hintergrund Bootstrap entworfen wurde. Entweder sind die Komponenten darauf ausgelegt, zusätzliche Inhalte zu verstecken bzw. anzuzeigen (wie z. B. Popup-Fenster, ausklappbare Elemente und Dropdowns) oder sie dienen zur Verbesserung der Benutzerfreundlichkeit bei Anwendungen mit viel Inhalt und vielen Unterseiten (wie z. B. Breadcrumb, Pagination und Ladeanimation). Diese Aspekte sind besonders für große Anwendungen, die viel Interaktivität der Nutzer voraussetzen, sinnvoll. Ein Beispiel einer solchen Anwendung ist dementsprechend Twitter.

6 Barker, 2014, S. 131.
7 Vgl. Otto, Mark/Jacob Thornton: Components, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.1/customize/components/ (abgerufen am 04.09.2021).
8 Vgl. Otto/Thornton, o. D.

4.4.2.3 LISTE AN UTILITIES

Nicht nur Komponenten werden mitgeliefert, sondern auch eine bestimmte Anzahl an Utility-Klassen. Diese sollen den Entwickelnden hauptsächlich dabei helfen, schneller responsive und mobile Ansichten zu entwickeln. Dementsprechend werden auch hauptsächlich Utility-Klassen mitgeliefert, die die Sichtbarkeit, die Farben und die Abstände von Elementen regeln. [9]

Die mitgelieferten Utilities von Bootstrap werden in 16 unterschiedliche Bereiche aufgeteilt: Hintergründe, Ränder, Farben, Display, Flex, Float, Interaktionen, Deckkraft, Overflow, Positionen, Schatten, Größen, Abstände, Text, vertikale Ausrichtung und Sichtbarkeit.

4.4.2.4 WEITERE ELEMENTE

In der Dokumentation von Bootstrap werden neben den Komponenten und Utilities weitere Reiter aufgeführt, wie Forms, Layouts und Helpers. Unter Forms und Layouts wird aufgeführt, wie Formulare und Gestaltungsraster in Bootstrap umsetzbar sind. Unter Helpers hingegen werden sonstige Funktionen aufgeführt.

4.4.2.5 UMSETZUNG VON RESPONSIVITÄT

Um Responsivität sicherzustellen, liefert Bootstrap sechs standardisierte Breakpoints (vgl. Abb. 4.31). Laut der Dokumentation von Bootstrap sind diese an häufig vorkommende Bildschirmbreiten unterschiedlicher Endgeräte orientiert und sollen ein Vielfaches von 12 entsprechen. [10] Warum dies bei den Breakpoints von 992 und 1400 Pixel nicht der Fall ist, ist aus der Dokumentation nicht herauszulesen.

Breakpoint	Class infix	Dimensions
X-Small	*None*	<576px
Small	`sm`	≥576px
Medium	`md`	≥768px
Large	`lg`	≥992px
Extra large	`xl`	≥1200px
Extra extra large	`xxl`	≥1400px

Die vordefinierten Breakpoints können angepasst werden, indem die Sass-Map `$grid-breakpoints` mit eigenen Werten überschrieben wird.

Abb. 4.31: Mitgelieferte Breakpoints von Bootstrap, Quelle: Quelle: https://getbootstrap.com/docs/5.1/layout/breakpoints/ (abgerufen am 04.09.2021)

9 Vgl. Otto, Mark/Jacob Thornton: Utilities for layout, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.1/layout/utilities/ (abgerufen am 04.09.2021).
10 Vgl. Otto, Mark/Jacob Thornton: Breakpoints, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.1/layout/breakpoints/ (abgerufen am 04.09.2021).

4.4.2.6 EINBINDUNGSMÖGLICHKEITEN

Bootstrap bietet drei unterschiedliche Möglichkeiten zur Einbindung des CSS-Frameworks an. Diese lauten wie folgt:

1. Einbindung mit Zugriff auf den Quellcode
2. Einbindung von bereits vorkompilierten CSS- und JS-Dateien
3. Einbindung mithilfe eines CDN

1. Einbindung mit Zugriff auf den Quellcode
Bei dieser Möglichkeit wird Bootstrap mittels `npm` als Abhängigkeit installiert, sodass der Zugriff auf den Quellcode des Frameworks ermöglicht wird. Dieser Quellcode kann mit selbst erstellten Dateien angepasst werden. Die korrekte Funktionsweise von Bootstrap kann allerdings nur dann sichergestellt werden, wenn weitere Abhängigkeiten ebenfalls installiert werden. [11] Diese werden in Abschnitt *4.4.3.1 Werkzeuge von Bootstrap* näher ergründet.

Für die Umsetzung der individuellen Gestaltung wird auf diese Einbindungsmöglichkeit zurückgegriffen, da mit diesem Ansatz die meisten Anpassungen der Gestaltungsparameter und des Quellcodes möglich sind. [12]

2. Einbindung von bereits vorkompilierten CSS- und JS-Dateien
Die zweite Möglichkeit besteht darin, bereits kompilierten Quellcode von Bootstrap einzubinden. Mit dieser Einbindung ist es nicht mehr möglich, auf den bereits bestehenden Quellcode von Bootstrap zuzugreifen. [13]

3. Einbindung mithilfe eines (CDN)
Bootstrap verwendet `jsDelivr` als Content-Delivery-Network. Der bereits bestehende Link zu den kompilierten Dateien wird in das HTML-Dokument eingebunden. Dadurch entsteht der Vorteil, dass keine Abhängigkeiten geladen werden müssen, sodass der Tooling-Aufwand verringert wird. Jedoch haben die Entwickelnden auch hier keinen Zugriff auf den Quellcode von Bootstrap. [14]

4.4.2.7 INHALTE

Laut der Detailansicht von `npm` zu Bootstrap enthält die Abhängigkeit insgesamt 189 Dateien, die zusammengefasst und nicht komprimiert

11 Vgl. Otto, Mark/Jacob Thornton: Download, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.1/getting-started/download/#source-files (abgerufen am 04.09.2021).
12 Vgl. Otto, Mark/Jacob Thornton: Customize, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.1/customize/overview/#overview (abgerufen am 04.09.2021).
13 Vgl. Otto/Thornton, o. D.
14 Vgl. Otto/Thornton, o. D.

eine Dateigröße von 8,57 Megabyte aufweisen. [15] Welche Dateien genau installiert werden, ist in der Dokumentation von Bootstrap zu finden. Die heruntergeladene Abhängigkeit beinhaltet sowohl bereits kompilierte CSS- und JavaScript-Dateien als auch alle Dateien des ursprünglichen Sass- und JavaScript-Quellcodes. Zudem wird die Dokumentation mitgeliefert. [16]

4.4.3 WERKZEUGE & TOOLING

In diesem Abschnitt werden die benötigten Werkzeuge erläutert, die für eine fehlerfreie Nutzung von Bootstrap verwendet und installiert werden müssen.

4.4.3.1 WERKZEUGE VON BOOTSTRAP

Neben der eigentlichen Abhängigkeit des CSS-Frameworks, werden zwei weitere Abhängigkeiten für eine fehlerfreie Nutzung vorausgesetzt. Die drei benötigten Abhängigkeiten sind die folgenden:

- `bootstrap`: Installation der Inhalte für das CSS-Framework Bootstrap [17]
- `node-sass`: Kompilierung von Sass- in CSS-Quellcode [18]
- `autoprefixer`: Überprüfung des CSS-Codes auf CSS-Eigenschaften, die sogenannte Herstellerpräfixe benötigen, um eine fehlerfreie Anzeige im Browser zu ermöglichen [19]

Ohne diese Abhängigkeiten funktioniert Bootstrap nicht.

4.4.3.2 WEITERE ABHÄNGIGKEITEN

Neben den von Bootstrap unbedingt benötigten Abhängigkeiten, gibt es eine Liste an weiteren Werkzeugen, die entweder als `dependencies` oder als `devDependencies` eingebunden werden. Die folgenden Abhängigkeiten zählen neben den aus Abschnitt *4.4.3.1 Werkzeuge von Bootstrap* aufgelisteten Abhängigkeiten ebenfalls zu den `dependencies`:

- `purgecss`: Überprüfung des CSS-Codes auf nicht genutzte Deklarationen [20]
- `postcss`: Manipulierung von CSS-Dateien mithilfe von JS-Plugins [21]

15 Vgl. npmjs: Npm: bootstrap, in: npmjs, 04.08.2021, https://www.npmjs.com/package/bootstrap (abgerufen am 05.09.2021).
16 Vgl. Otto, Mark/Jacob Thornton: Contents, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.0/getting-started/contents/ (abgerufen am 05.09.2021).
17 Vgl. npmjs, 2021.
18 Vgl. npmjs, 2021.
19 Vgl. Robbins, 2018, S. 577
20 Vgl. npmjs, 2021.
21 Vgl. npmjs, 2021.

`postcss-cli`: Ausführung von `postcss` über die Kommandozeile [22]

- `webpack`: Bündeln und Zusammenführen von JavaScript-Dateien [23]
- `webpack-cli`: Ausführen von Webpack über die Kommandozeile [24]
- `babel-loader`: Loader für Webpack, um JS auch für alte Browser verständlich zu gestalten [25]

Zu den `devDependencies` zählen wiederum diese Abhängigkeiten:

- `nodemon`: Automatisiertes Neustarten der Anwendung bei Fehler [26]
- `npm-run-all`: Ausführung mehrerer `npm`-Skripte gleichzeitig [27]
- `serve`: Erstellen eines lokalen Servers zum Laden statischer Dateien [28]
- `stylelint`: Überprüfung des Quellcodes auf Wiederholungen und Syntax-Fehler [29]
- `stylelint-config-twbs-bootstrap`: Anwendung einer Sammlung von Regeln für die Syntax der CSS- und Sass-Dateien von Bootstrap [30]

Diese Abhängigkeiten werden in insgesamt neun unterschiedlichen Skripten festgehalten und aufgerufen.

4.4.4 VORDEFINIERTES GESTALTUNGSSYSTEM

Das Gestaltungssystem von Bootstrap basiert auf zwei unterschiedlichen Aspekten: die definierten Variablen und die Auswahl an Komponenten.

Alle grundlegenden Gestaltungsparameter werden bei Bootstrap in einer Sass-Datei festgehalten. Diese Datei trägt den Namen `_variables.scss`. Sie dient als Ausgangspunkt des vordefinierten Gestaltungssystems und liefert bereits zu Beginn eine Vielzahl an aneinander angepassten Farbpaletten. [31] Mittels dieser Farbpaletten wurde von Bootstrap bereits ein Farbschema festgelegt, das auf alle Komponenten automatisch angewendet wird. Bei der Ersteinbindung haben also alle Komponenten bei allen Entwickelnden immer einen identischen Farbkanon.

22 Vgl. npmjs, 2021.
23 Vgl. npmjs: Npm: webpack, in: npmjs, 03.09.2021, https://www.npmjs.com/package/webpack (abgerufen am 09.09.2021).
24 Vgl. npmjs: Npm: webpack-cli, in: npmjs, 15.08.2021, https://www.npmjs.com/package/webpack-cli (abgerufen am 09.09.2021).
25 Vgl. npmjs: Npm: babel-loader, in: npmjs, 26.11.2020, https://www.npmjs.com/package/babel-loader (abgerufen am 09.09.2021).
26 Vgl. npmjs, 2021.
27 Vgl. npmjs, 2021.
28 Vgl. npmjs, 2021.
29 Vgl. npmjs, 2021.
30 Vgl. npmjs: Npm: stylelint-config-twbs-bootstrap, in: npmjs, 19.07.2021, https://www.npmjs.com/package/stylelint-config-twbs-bootstrap (abgerufen am 05.09.2021).
31 Vgl. Otto, Mark/Jacob Thornton: Sass, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.1/customize/sass/#variable-defaults (abgerufen am 06.09.2021).

Auch bei den Abständen sind bereits vordefinierte rem-Werte vorhanden, die die Maße bestimmter Komponenten beeinflussen. Sie entsprechen standardisiert einem Vielfachen von 0.5rem. [32]

Nicht nur Farben und Abstände sind bereits definiert, sondern auch Schriftgrößen, welche Schriftgröße für welche Überschrift genutzt wird, wie alle Felder eines Formulars aussehen und vieles mehr.

Schlussendlich bietet das CSS-Framework den Entwickelnden nicht nur Komponenten und Utilities an, sondern liefert eine bereits komplett durchdachte und durchgestylte Gestaltung einer Webseite. Durch die bereits vordefinierten Parameter wird von vornerein eine Konsistenz innerhalb des Designs gewährleistet.

Durch die eingeschränkte Auswahl der mitgelieferten Komponenten geben die Entwickler von Bootstrap bestimmte Inhaltstypen vor, die ohne großen Entwicklungsaufwand umgesetzt werden können. Gleichzeitig entsteht dadurch eine Einschränkung, welche Inhalte sich rein mit Bootstrap umsetzen lassen und welche nicht.

4.4.5 UMSETZUNG VON GESTALTUNGSANGABEN

Damit das Endprodukt der Gestaltung des Screendesigns entspricht, müssen die vordefinierten Parameter von Bootstrap innerhalb der `_variables.scss` Datei angepasst werden. Dafür müssen die vorgeschriebenen Bootstrap-Variablen überschrieben werden. [33] Diese Überschreibung findet in einer selbst erstellten Sass-Datei statt.

4.4.5.1 DEFINITION VON GESTALTUNGSPARAMETERN

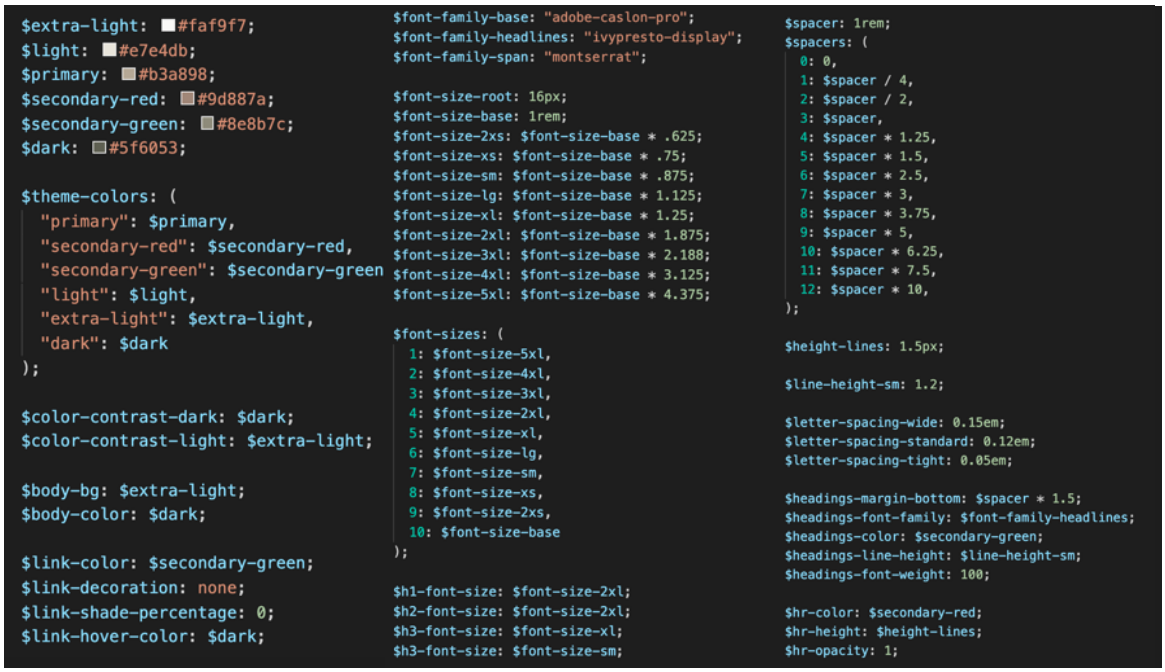
Die Datei der Bootstrap-Variablen weist insgesamt eine Zeilenlänge von 1639 Zeilen auf. Innerhalb dieses Dokumentes befinden sich eine große Auswahl an Variablen, die nach dem Belieben des Entwickelnden angepasst und verändert werden können. Bestimmte Variablen davon sind globale Einstellungen, andere wiederum komponentenspezifisch.

Die globalen Einstellungsvariablen können das Verhalten des CSS-Frameworks beeinflussen. [34] Auch in dieser Anwendung wurden bestimmte globale Einstellungen überschrieben. Dazu zählt die Aktivierung des

32 Vgl. Otto, Mark/Jacob Thornton: Spacing, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.1/utilities/spacing/#maps (abgerufen am 06.09.2021).
33 Vgl. Otto/Thornton, o. D.
34 Vgl. Otto, Mark/Jacob Thornton: Options, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.0/customize/options/ (abgerufen am 08.09.2021).

Grid-Systems von CSS und die Deaktivierung der abgerundeten Ecken, da diese innerhalb der individuellen Gestaltung keine Anwendung finden.

Die wichtigsten Gestaltungsparameter sind jedoch die grundlegenden Designelemente der individuellen Gestaltung, sprich Farbe, Typografie und Abstände. Dementsprechend sind dies Variablen, die auf jeden Fall überschrieben werden müssen, um die Atmosphäre des Screendesigns widerspiegeln zu können (vgl. Abb. 4.32, Abb. 4.33, Abb. 4.34).



```scss
$extra-light: #faf9f7;
$light: #e7e4db;
$primary: #b3a898;
$secondary-red: #9d887a;
$secondary-green: #8e8b7c;
$dark: #5f6053;

$theme-colors: (
  "primary": $primary,
  "secondary-red": $secondary-red,
  "secondary-green": $secondary-green,
  "light": $light,
  "extra-light": $extra-light,
  "dark": $dark
);

$color-contrast-dark: $dark;
$color-contrast-light: $extra-light;

$body-bg: $extra-light;
$body-color: $dark;

$link-color: $secondary-green;
$link-decoration: none;
$link-shade-percentage: 0;
$link-hover-color: $dark;
```

```scss
$font-family-base: "adobe-caslon-pro";
$font-family-headlines: "ivypresto-display";
$font-family-span: "montserrat";

$font-size-root: 16px;
$font-size-base: 1rem;
$font-size-2xs: $font-size-base * .625;
$font-size-xs: $font-size-base * .75;
$font-size-sm: $font-size-base * .875;
$font-size-lg: $font-size-base * 1.125;
$font-size-xl: $font-size-base * 1.25;
$font-size-2xl: $font-size-base * 1.875;
$font-size-3xl: $font-size-base * 2.188;
$font-size-4xl: $font-size-base * 3.125;
$font-size-5xl: $font-size-base * 4.375;

$font-sizes: (
  1: $font-size-5xl,
  2: $font-size-4xl,
  3: $font-size-3xl,
  4: $font-size-2xl,
  5: $font-size-xl,
  6: $font-size-lg,
  7: $font-size-sm,
  8: $font-size-xs,
  9: $font-size-2xs,
  10: $font-size-base
);

$h1-font-size: $font-size-2xl;
$h2-font-size: $font-size-2xl;
$h3-font-size: $font-size-xl;
$h3-font-size: $font-size-sm;
```

```scss
$spacer: 1rem;
$spacers: (
  0: 0,
  1: $spacer / 4,
  2: $spacer / 2,
  3: $spacer,
  4: $spacer * 1.25,
  5: $spacer * 1.5,
  6: $spacer * 2.5,
  7: $spacer * 3,
  8: $spacer * 3.75,
  9: $spacer * 5,
  10: $spacer * 6.25,
  11: $spacer * 7.5,
  12: $spacer * 10,
);

$height-lines: 1.5px;

$line-height-sm: 1.2;

$letter-spacing-wide: 0.15em;
$letter-spacing-standard: 0.12em;
$letter-spacing-tight: 0.05em;

$headings-margin-bottom: $spacer * 1.5;
$headings-font-family: $font-family-headlines;
$headings-color: $secondary-green;
$headings-line-height: $line-height-sm;
$headings-font-weight: 100;

$hr-color: $secondary-red;
$hr-height: $height-lines;
$hr-opacity: 1;
```

Abb. 4.32 (links): Überschreiben der Farb-Variablen,
Abb. 4.33 (mittig): Überschreiben der Schrift-Variablen,
Abb. 4.34 (rechts): Überschreiben der Abstand-Variablen,
Quelle: Eigene Darstellung

Zudem werden auch komponentenspezifische Variablen überschrieben. Dazu zählen beispielsweise `$btn-padding-y` und `$btn-padding-x` für das `padding` eines Buttons oder `$navbar-light-color`, um die Farbe der Komponentenvariante `navbar-light` zu ändern. Zudem gibt es eine Auswahl an selbst definierten Variablen, die die Entwicklung und die Wartbarkeit des Codes vereinfachen sollen.

4.4.5.2 SCHWIERIGKEITEN WÄHREND DER UMSETZUNG VON GESTALTUNGSANGABEN

Während der Entwicklung sind Probleme im Workflow zur Variablenüberschreibung aufgefallen, die auf einem Sass-Problem basieren. Obwohl die Überschreibung der Variablen ein unglaublich wichtiger Faktor für Bootstrap ist, wird dieses Problem nicht umgangen, sondern lediglich in einem Satz in der Dokumentation angesprochen.

Das Problem besteht darin, dass die Variablenüberschreibung entgegengesetzt der eigentlichen Kaskade von CSS und somit entgegengesetzt der Intuition des Entwickelnden erfolgen muss, um korrekt greifen zu können.

Aufgrund der in Abschnitt *2.2.3 Funktionsweise von CSS* beschriebenen Spezifikation von CSS besteht der erste Impuls des Entwickelnden darin, die selbst erstellte Sass-Datei mit den eigenen Gestaltungsparametern nach dem Import der Bootstrap-Quelldateien vorzunehmen. Dadurch werden jedoch nicht alle Variablen in der Bootstrap-Quelldatei überschrieben, sondern nur die direkt angesprochenen. Sobald die überschriebene Variable einer weiteren Variablen zugewiesen wird, wird der überschriebene Wert nicht mehr übernommen. [35]

Dementsprechend muss die selbst erstellte Sass-Datei zur Variablenüberschreibung vor den mitgelieferten Bootstrap-Dateien importiert werden.

4.4.5.3 UTILITIES ANPASSEN

Wie bei dem Utility-Klassen basierten CSS-Framework TailwindCSS, wird mit Bootstrap nur eine Auswahl an Utility-Klassen mitgeliefert. Hier ist die Auswahl jedoch um einiges geringer, sodass viele CSS-Eigenschaften überhaupt nicht abgebildet werden können. Die mitgelieferten Utilities reichen nicht für die Umsetzung des individuellen Screendesigns aus. Mittels der Utility API von Bootstrap und der Funktion `map-merge` von Sass lassen sich die mitgelieferten Utilities in einer selbst erstellten Sass-Datei erweitern oder anpassen. [36] Diese Anpassungen finden in der 107 Zeilen langen `/components/_utilities.sass` Datei statt. Dabei wurde die Utility für folgende CSS-Eigenschaften komplett neu hinzugefügt: `justify-self`.

Nicht nur die geringe Auswahl an Utility-Klassen ist ein Problem, sondern auch die geringe Werteauswahl. Beispielsweise werden für die Utility-Klasse `height` insgesamt nur 5 Werte mitgeliefert: 25%, 50%, 75%, 100% und auto. [37] Dieses Problem kann ebenfalls mittels der Utility API behoben werden, indem den Sass-Maps der bereits bestehenden Utilities neue Werte zugewiesen werden. [38] Somit werden Utilities der folgenden CSS-Eigenschaften erweitert: `font-family`, `padding-x`, `margin-x`, `max-width` und `max-height`.

35 Vgl. Twbs: Issue with variable overrides · Issue #43 · twbs/bootstrap-npm-starter, in: GitHub, o. D., https://github.com/twbs/bootstrap-npm-starter/issues/43 (abgerufen am 08.09.2021).
36 Vgl. Otto, Mark/Jacob Thornton: Utility API, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.0/utilities/api/ (abgerufen am 23.09.2021).
37 Vgl. Otto, Mark/Jacob Thornton: Sizing, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.0/utilities/sizing/ (abgerufen am 23.09.2021).
38 Vgl. Otto/Thornton, o. D.

In Abschnitt *4.4.8 Responsive Webdesign & Mobile First* wird genauer beleuchtet, wie Utilities responsiv angewendet werden können. Auch von den responsiven Utilities werden nur eine bestimmte Stückzahl generiert und mitgeliefert. Um Utilities, wie beispielsweise Schriftgröße, -farbe oder Position responsiv verändern zu können, muss in der selbst erstellten `_utilities.sass` Datei der Wert für `responsive` aktiviert werden. [39] Dies wurde für die folgenden Utilities bzw. CSS-Eigenschaften ermöglicht: `font-size`, `height`, `width`, `position`, `max-width`, `max-height` und `justify-self`.

4.4.6 VERWENDETE KOMPONENTEN

In dieser Anwendung wurden insgesamt vier der mitgelieferten Komponenten von Bootstrap verwendet.

1. Komponente: `navbar`
Diese Komponente wurde für die Umsetzung des Headers verwendet. Mit der Komponentenvariante `navbar-collapse` wird die Sichtbarkeit der Navigation durch ein Menü-Icon geregelt. Mit der Angabe `navbar-expand-lg` ist definiert, ab welchem Breakpoint die Navigation nicht mehr versteckt wird und mit `navbar-light` wird die Farbvariante des Headers bestimmt. [40]

2. Komponente: `blockquote`
Die zweite Komponente wurde insgesamt zwei Mal verwendet: einmal für ein Zitat im Bereich über die Philosophie und einmal im Kontakt-Bereich.

3. Komponente: `btn`
Für den Button „Teile deine Geschichte mit uns!“ wurde die Button-Komponente verwendet. Hier konnte die Komponentenvariante `btn-extra-light` angewendet werden, die die Farbwelt des Buttons bestimmt. [41]

4. Komponente: `collapse`
Als vierte Komponente wurde `collapse` verwendet, um in der mobilen Ansicht der Webseite die Produktauflistung ein- und ausklappbar zu gestalten.

Grundsätzlich wurde bei der Umsetzung der individuellen Gestaltung mit Bootstrap eher auf die mitgelieferten Utilities anstatt auf die Komponenten zurückgegriffen. In Abschnitt *4.4.2.2 Liste an Komponenten* wurde bereits erörtert, dass viele der Komponenten von Bootstrap auf größere Webseiten mit viel Inhalt ausgelegt sind. Diese Komponenten finden jedoch kaum eine

39 Vgl. Otto/Thornton, o. D.
40 Vgl. Otto/Thornton, o. D.
41 Vgl. Otto, Mark/Jacob Thornton: Buttons, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.0/components/buttons/ (abgerufen am 08.09.2021).

Verwendung bei einer Umsetzung wie dieser, da beispielsweise zu wenig Inhalt für eine Pagination und zu wenig Unterseiten für ein Breadcrumb vorhanden sind und keine Push-Benachrichtigungen oder andere Interaktionsmöglichkeiten benötigt werden.

4.4.7 REALISIERUNG VON CROSS-BROWSER-RENDERING

Innerhalb dieses Abschnitts wird untersucht, wie Bootstrap Cross-Browser-Rendering ermöglicht und wie es um die Browserkompatibilität der Abhängigkeit steht.

4.4.7.1 REBOOT

Die Anpassung der User-Agent Stylesheets aneinander wird automatisch durch Bootstrap vorgenommen. Der Quellcode dieser Anpassungen befindet sich in der Datei `_reboot.scss`. Der Grundbaustein des Bootstrap-Reboots ist die bereits bekannte Abhängigkeit `normalize.css`. Innerhalb der Dokumentation von Bootstrap werden die wichtigsten Anpassungen beschrieben und begründet. Zum Beispiel sorgt das Reboot dafür, dass Standard-Werte für die Vereinheitlichung von Einheiten nicht in em, sondern in rem angegeben werden und dass die CSS-Eigenschaft `margin–top` vermieden wird, um ineinander fallende `margins` zu verhindern. [42]

4.4.7.2 BROWSERKOMPATIBILITÄT

Bootstrap weist eine sogenannte `.browserlistrc` Datei auf, in der alle kompatiblen Browser mit Versionsnummern auflistet werden (vgl. Abb. 4.35). Seit der Bootstrap-Version 5.0 wird der Browser Internet Explorer von dem CSS-Framework nicht mehr unterstützt. Laut der Dokumentation sollten die Entwickelnden, die auf eine Kompatibilität mit IE angewiesen sind, auf die Bootstrap-Version 4 zurückgreifen. [43]

```
# https://github.com/browserslist/browserslist#readme

>= 0.5%
last 2 major versions
not dead
Chrome >= 60
Firefox >= 60
Firefox ESR
iOS >= 12
Safari >= 12
not Explorer <= 11
```

Abb. 4.35: Browserkompatibilität, Quelle: https://getbootstrap.com/docs/5.0/getting-started/browsers-devices/ (abgerufen am 08.09.2021)

42 Vgl. Otto/Thornton, o. D.
43 Vgl. Otto, Mark/Jacob Thornton: Browsers and devices, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.0/getting-started/browsers-devices/#internet-explorer (abgerufen am 08.09.2021).

In Abschnitt *4.3.6.2 Browserkompatibilität* wurde bereits der Nutzen der Herstellerpräfixe thematisiert. Auch Bootstrap greift bei der Kompilierung des eigenen Quellcodes auf die Verwendung von `autoprefixer` zurück, um die Unterstützung aller CSS-Eigenschaften in unterschiedlichen Browsern zu gewährleisten. [44]

4.4.8 RESPONSIVE WEBDESIGN & MOBILE FIRST

In Abschnitt *4.4.2.5 Umsetzung von Responsivität* wurden die von Bootstrap mitgelieferten Breakpoints angesprochen und untersucht. Die Nutzung von Breakpoints innerhalb der Komponenten bzw. Utilities und die dadurch entstehende Responsivität ist eines der Kernkonzepte von Bootstrap. Das CSS-Framework setzt zudem auf die Frontend-Strategie Mobile First. [45]

Anhand der definierten Breakpoints aus der Sass-Map `$grid-breakpoints` werden nicht nur Klassen für Komponenten wie z. B. `navbar-expand-{breakpoint}` generiert, sondern auch Utility-Klassen. Um beispielsweise ein Textelement nur für große Viewportbreiten sichtbar zu gestalten, könnten folgende Utilities angewendet werden:

```
<p class="d-none d-lg-block">Das ist ein Beispiel</p>
```

Die Utility `d-none` ist das Äquivalent für die CSS-Eigenschaft `display: none` und `d-lg-block` für die CSS-Eigenschaft `display: block`, wobei diese ab der definierten Viewportbreite `lg` angewendet wird.

Wie bei den beiden vorherigen Umsetzungen wurde auch hier die Webseite mit einer Mindestbildschirmbreite von 320 Pixel entwickelt. Die von Bootstrap vordefinierten Breakpoints wurden nicht überschrieben, da mit dieser Entwicklung getestet werden soll, wie sich diese verhalten und welche Auswirkungen sie auf das Endergebnis haben.

4.4.9 PROGRESSIVE ENHANCEMENT & JAVASCRIPT

Aus Abschnitt *4.4.2.7 Inhalte* lässt sich bereits herauslesen, dass Bootstrap neben dem eigentlichen Sass-Quellcode auch JavaScript-Dateien mitliefert. Nun stellt sich die Frage, wofür die JavaScript Dateien eingesetzt werden.

Die Dokumentation von Bootstrap beschreibt die Nutzung von JS wie folgt: „Many of our components require the use of JavaScript to function.

44 Vgl. Otto/Thornton, o. D.
45 Vgl. Otto/Thornton, o. D.

Specifically, the require our own JavaScript plugins and Popper.“ [46] Die Dokumentation listet alle Komponenten auf, deren Funktionsweise von JS abhängig sind (vgl. Abb. 4.36).

Insgesamt basiert die Funktionsweise von 11 Komponenten auf JS. Dies hat besonders hinsichtlich Progressive Enhancement einen großen Nachteil, da die korrekte Ausführbarkeit von JS und somit auch die korrekte Funktionsweise der Komponenten bei den Anwendenden nicht garantiert ist. Um dies in Relation zu setzen, wären bei einer fehlerhaften Ausführung von JS fast die Hälfte aller mitgelieferten Komponenten von Bootstrap eingeschränkt.

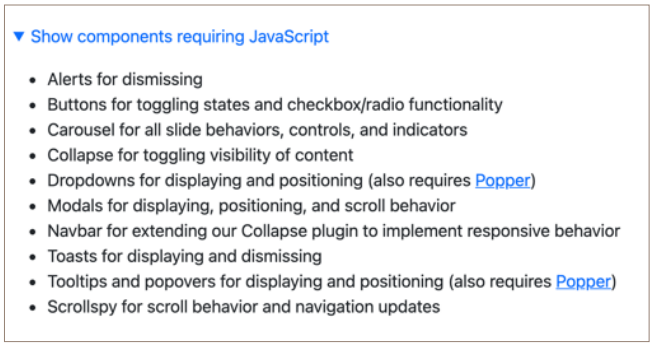


Abb. 4.36: Komponenten mit JS, Quelle: https://getbootstrap.com/docs/5.0/getting-started/introduction/#js (abgerufen am 08.09.2021)

Auch in der Umsetzung der Webseite für das vegane Café werden Komponenten genutzt, deren Funktionsweisen auf JavaScript basieren. Die erste Komponente `navbar` regelt die Sichtbarkeit des Menüs. Sobald JavaScript bei dem Endgerät des Anwendenden nicht korrekt oder gar nicht ausführbar ist, kann in der mobilen Ansicht der Webseite das Menü nicht mehr angezeigt werden. Die zweite Komponente `collapse` reguliert die Sichtbarkeit der Produktauflistung. Bei der Abwesenheit von JavaScript wird diese Auflistung ebenfalls nicht mehr angezeigt. Dies ist laut der Strategie von Progressive Enhancement nicht das optimale Ergebnis einer Entwicklung.

Neben den mitgelieferten JS-Dateien von Bootstrap werden ebenfalls zwei weitere selbst erstellte JS-Dateien eingebunden. Diese Dateien sind identisch zu den JS-Dateien, die in der Umsetzung mit handgeschriebenen CSS angewendet wurden (vgl. *4.2.9 Schwierigkeiten bei der Umsetzung*).

4.4.10 CODE-QUALITÄT

Auch bei der Entwicklung mit Bootstrap hat die Code-Qualität einen hohen Stellenwert, damit Wartbarkeit, Konsistenz und Benutzerfreundlichkeit auch in dieser Umsetzung sichergestellt werden können.

46 Otto, Mark/Jacob Thornton: Introduction, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.0/getting-started/introduction/#js (abgerufen am 09.09.2021).

4.4.10.1 VERHINDERUNG VON REDUNDANTEM CODE

Die Komponenten des Frameworks sind bereits darauf ausgelegt, dass durch ihre Nutzung redundanter Code verhindert wird. Da bei dieser Herangehensweise nicht alle Inhaltselemente mit den vordefinierten Komponenten umgesetzt werden können, muss eigener Quellcode generiert werden. Demnach muss der eigene Quellcode ebenfalls optimiert werden.

Zum einen wurde die von Bootstrap genutzte „base-modifier“-Nomenklatur genutzt, um eigene Komponenten zu erstellen. Eine selbst erstellte Komponente ist beispielsweise die Klasse `circle`. Mit ihren Modifier-Klassen `circle-bottom-right` bzw. `circle-top-left` können die Kreise als Designelemente umgesetzt werden.

Zum anderen wurden für wiederkehrende Deklarationen `@mixins` genutzt, sodass damit beispielsweise das individuelle Raster umgesetzt werden konnte.

4.4.10.2 SYNTAX & STRUKTUR

Wie in der Entwicklung mit handgeschriebenen CSS wurde auch hier das `postcss` Plugin `stylelint` genutzt. Für die Umsetzung mit Bootstrap wurde in dem Konfigurationsobjekt `stylelintrc.js` ebenfalls eine Sammlung an bereits bestehenden Regeln importiert, anhand derer der Quellcode untersucht werden kann. Das importierte Plugin in dieser Anwendung trägt die Bezeichnung `stylelint-config-twbs-bootstrap`.

Stylelint-config-twbs-bootstrap:
Diese Regelsammlung wird auch im eigenen Quellcode von Bootstrap genutzt. [47] Hierbei wird der Quellcode auf grundlegende Konventionen überprüft, wie die korrekte Anwendung von Semikola und das Verwenden von Leerzeichen. Gleichzeitig wird aber auch eine spezifische, individuelle Reihenfolge der CSS-Eigenschaften festgelegt. Zudem wird die maximale Anzahl der Attribute, der Klassen, der Kombinatoren und der ID’s innerhalb der Selektoren festgelegt. [48]

Innerhalb der Dokumentation von Bootstrap werden zudem Empfehlungen zur HTML-Struktur ausgesprochen. Sobald Komponenten oder Utilities vorgestellt werden, liefert die Dokumentation Anwendungsbeispiele. Sie dienen zwar als Veranschaulichung, jedoch verleitet es dazu, diese HTML-Struktur einfach zu übernehmen.

47 Vgl. npmjs, 2021.
48 Vgl. Twbs: Stylelint-config-twbs-bootstrap/index.js at main · twbs/stylelint-config-twbs-bootstrap, in: GitHub, o. D., https://github.com/twbs/stylelint-config-twbs-bootstrap/blob/main/css/index.js (abgerufen am 10.09.2021).

```
<div class="card">
  <div class="card-header">
    Featured
  </div>
  <div class="card-body">
    <h5 class="card-title">Special title treatment</h5>
    <p class="card-text">With supporting text below as a natural lead-in to additional co
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>
```

Abb. 4.37: HTML-Struktur einer `card`-Komponente, Quelle: https://getbootstrap.com/docs/5.1/components/card/#header-and-footer (abgerufen am 10.09.2021)

Abbildung 4.37 zeigt eine solche empfohlene HTML-Struktur. Nicht nur in diesem Beispiel, sondern auch bei vielen anderen beispielhaften Anwendungen von Komponenten oder Utilities, wird auffallend häufig das HTML-Element `<div>` verwendet. Grundsätzlich ist nichts verkehrt an der Nutzung eines `<div>`-Elements, jedoch muss dem Entwickelnden bewusst sein, dass dieses Element keine semantische Bedeutung aufweist. Bei deren häufigen Anwendung leidet folglich die Semantik des HTML-Dokuments und demnach sollte dieses Element sparsam und gezielt eingesetzt werden. [49] Dieser Aspekt wird in der Dokumentation leider kaum berücksichtigt.

4.4.10.3 VALIDATOREN

Wie bei den beiden vorherigen Umsetzungen wird auch für diese Herangehensweise der „Nu HTML Checker“ und der „Jigsaw-Validator“ zur Überprüfung von validem HTML und validem Quellcode herangezogen.

Nu HTML Checker:
Für die Überprüfung der HTML-Semantik und -Struktur wird der „Nu HTML Checker“ von W3C verwendet. Im folgenden Absatz wird das Ergebnis dieses Validators ergründet.

Insgesamt gibt der „Nu HTML Checker“ 10 Fehler aus. Die ersten drei Fehler, die in Abbildung 4.38 gezeigt werden, entstehen aufgrund der fehlerhaften HTML-Struktur des Menü-Buttons. Der Validator bemängelt, dass innerhalb des `<button>`-Elements insgesamt drei weitere `<div>`-Elemente vorhanden sind. Diese werden für die Gestaltung des typischen Menü-Icons eingesetzt. Bei den beiden vorherigen Herangehensweisen wurde innerhalb des `<button>`-Elements ein SVG eingebunden, sodass diese Fehlermeldung dort umgangen wurde.

Nun stellt sich jedoch die Frage, warum in der Umsetzung mit Bootstrap nicht auch ein SVG eingebunden wurde. Sowohl bei der Herangehensweise mit handgeschriebenen CSS als auch mit TailwindCSS sind die Buttons zum Öffnen und zum Schließen zwei unterschiedliche `<button>`-Elemente. Dementsprechend agieren sie getrennt voneinander und beeinflussen sich nicht gegenseitig. Somit können die beiden Buttons auch zwei unterschiedliche Icons beinhalten. Der Schließen-Button weist ein Kreuz auf, wohingegen das Menü-Icon aus drei Strichen besteht. In Bootstrap wurde

49 Vgl. MDN Web Docs: - HTML: HyperText Markup Language | MDN, in: MDN Web Docs, 09.09.2021, https://developer.mozilla.org/de/docs/Web/HTML/Element/div (abgerufen am 10.09.2021).

dafür die Komponente navbar eingesetzt, die jedoch ein und denselben Button zum Öffnen bzw. Schließen des Menüs verwendet. Dadurch ist es nicht möglich, zwei unterschiedliche SVGs einzubinden. Damit der Button aber weiterhin beim Schließen und Öffnen das korrekte Icon anzeigt, müssen die Icons mit HTML-Elementen nachgebildet und animiert werden. Aus diesem Grund befinden sich die -Elemente innerhalb des Buttons, da diese die Icons nachstellen.

Abb. 4.38: Fehler in Button-Struktur, Quelle: Screenshot des Nu HTML Checkers für https://groenwald-thesis-bootstrap.netlify.app/ (abgerufen am 10.09.2021)

Die weiteren sieben Fehler entstehen durch den mehrfachen Einsatz der collapse Komponente innerhalb der Produktauflistung (vgl. Abb. 4.39). Diese Fehlermeldung besagt, dass das verwendete -Element kein type Attribut beinhalten darf und mit einem Button ersetzt werden sollte. Hierbei wurde absichtlich kein <button>-Element verwendet, da die Funktion der Komponente nur auf mobile Endgeräte angewendet werden soll. Ab einer Bildschirmbreite von 992 Pixel wird die collapse Komponente zu einer normalen Auflistung, sodass in diesem Fall die Semantik des Buttons nicht mehr passend gewesen wäre.

Abb. 4.39: Fehler in collapse-Komponente, Quelle: Screenshot des Nu HTML Checkers für https://groenwald-thesis-bootstrap.netlify.app/ (abgerufen am 10.09.2021)

Jigsaw-Validator:
Der „Jigsaw-Validator“ ist ein Überprüfungswerkzeug des W3C für CSS-Dateien. Abbildung 4.40 zeigt das fehlerfreie Ergebnis des „Jigsaw-Validators“.

Abb. 4.40: Ergebnis des „Jigsaw-Validators“, Quelle: Screenshot des Jigsaw-Validators für https://groenwald-thesis-bootstrap.netlify.app/ (abgerufen am 10.09.2021)

4.4.11 PERFORMANCE IM BROWSER

Auch das Endresultat der Entwicklung mit Bootstrap wird optimiert und mithilfe unterschiedlicher Werkzeuge im Browser untersucht.

4.4.11.1 OPTIMIERUNGSMÖGLICHKEITEN

Optimierung der Bilder:
In dieser Umsetzung wurde die gleiche Strategie zur Optimierung der Bilder verfolgt wie in der Umsetzung mit handgeschriebenen CSS und TailwindCSS. Aufgrund dessen ist die HTML-Struktur in diesem Bereich bei allen Herangehensweisen identisch (*4.2.7.1 Optimierungsmöglichkeiten, 4.3.10.1 Optimierungsmöglichkeiten*).

JavaScript-Dateien:
Im Gegensatz zu den anderen Herangehensweisen ist die Optimierung und die Verwendung der JavaScript-Quelldateien von Bootstrap etwas schwieriger. Während der lokalen Entwicklung können die JS-Dateien von Bootstrap wie folgt eingebunden werden:

```
import „/node_modules/bootstrap/js/dist/*“;
```

Durch diese Angabe werden alle JS-Quelldateien von Bootstrap in das Projekt importiert. Sobald die Webseite jedoch online über eine Domain aufrufbar ist, kann das Projekt nicht mehr auf den `node_modules`-Ordner zugreifen und es folgt eine Fehlermeldung, dass die JavaScript-Dateien nicht mehr geladen werden können (vgl. Abb. 4.41).

Abb. 4.41: Fehlermeldung bei Zugriff auf `node_modules`-Ordner, Quelle: https://6139f2042e67d70007823815--groenwald-thesis-bootstrap.netlify.app/ (abgerufen am 10.09.2021)

In der Dokumentation sind zwei Werkzeuge aufgeführt, die für das Bündeln und Zusammenführen von JS-Dateien zuständig sind: Webpack und Parcel. Diese beiden externen Werkzeuge sorgen dafür, dass die JS-Dateien von Bootstrap und die eigenen JS-Dateien neu interpretiert und in einer neuen Datei zusammengesetzt werden. Diese neu generierte Datei kann auch vom Browser ausgelesen werden, sodass JavaScript ausgeführt werden kann. Dadurch wird die Fehlermeldung vermieden. Aus diesem Grund wurde in dieser Umsetzung auf Webpack zurückgegriffen. Für die Verwendung von Webpack müssen allerdings weitere Abhängigkeiten geladen und eine Konfigurationsdatei erstellt werden, sodass sich dadurch der Tooling-Aufwand weiterhin erhöht. [50]

50 Vgl. Otto, Mark/Jacob Thornton: Webpack and bundlers, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.1/getting-started/webpack/ (abgerufen am 10.09.2021).

Löschen von nicht verwendetem Quellcode:
In der vorherigen Umsetzung wurde bereits angerissen, warum das Löschen von nicht verwendeten Codeabschnitten eine große Bedeutung für CSS-Frameworks hat (vgl. *4.3.10.1 Optimierungsmöglichkeiten*). Im Gegensatz zu TailwindCSS liefert Bootstrap keine bereits eingebaute Lösung zum Löschen von nicht genutzten CSS-Deklarationen. Innerhalb der Dokumentation von Bootstrap wird jedoch auf die Abhängigkeit `purgecss` aufmerksam gemacht. [51] Mittels dieser Abhängigkeit und einem selbst erstellten Skript können die verwendeten Selektoren innerhalb des CSS-Quellcodes mit den verwendeten Klassen in der `index.html` und in den JS-Dateien verglichen werden. Daraus entsteht eine optimierte Schnittmenge (vgl. Abb. 4.42).

```
"css-purge": "purgecss --keyframes --css build/styles.css --content index.html src/js/*.js \"node_modules/bootstrap/js/dist/{util,modal}.js\" --output build/",
```

Abb. 4.42: Skript mit `purgecss`, Quelle: Eigene Darstellung

4.4.11.2 WERKZEUGE ZUM TESTEN DER PERFORMANCE

Auch das Endergebnis mit Bootstrap wird durch den Hosting-Service Netlify zur Verfügung gestellt.
Die Domain lautet: https://groenwald-thesis-bootstrap.netlify.app/

Pagespeed Insights:
Die folgenden Abbildungen 4.43 und 4.44 zeigen die Ergebnisse des „Google Pagespeed Tools“ einmal für mobile Endgeräte und einmal für Desktop. Für Mobil liegt das Ergebnis des Tests bei 87% und für Desktop bei 99%. Auch hier können die Ergebnisse schwanken.

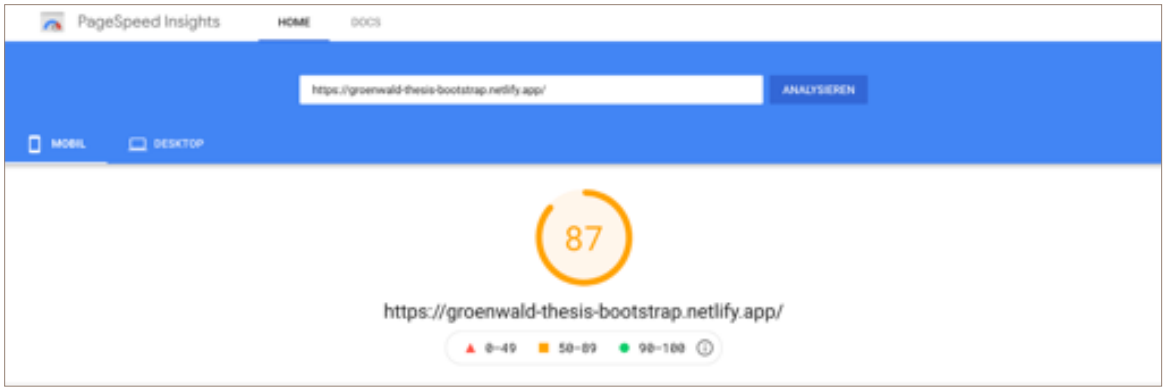


Abb. 4.43: PageSpeed Insights Ergebnis Mobil, Quelle: Screenshot des Pagespeed Insights (abgerufen am 11.09.2021)

51 Vgl. Otto, Mark/Jacob Thornton: Optimize, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.1/customize/optimize/#unused-css (abgerufen am 10.09.2021).

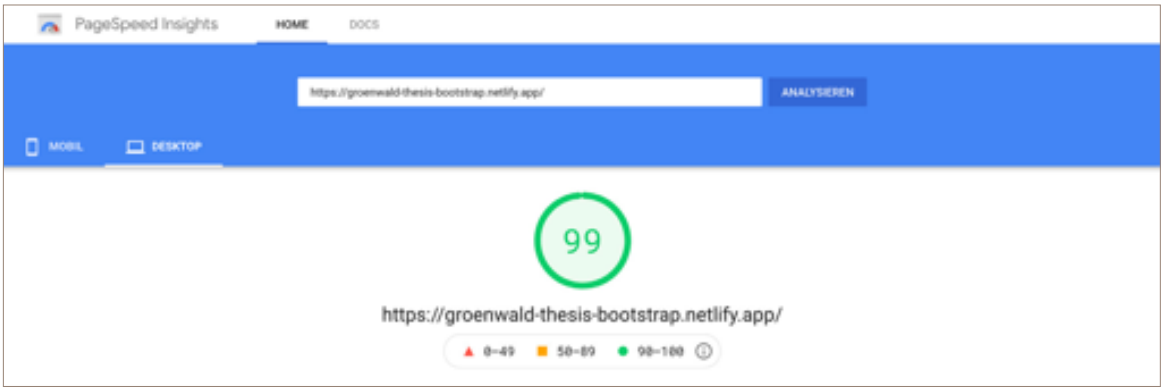


Abb. 4.44: PageSpeed Insights Ergebnis Desktop, Quelle: Screenshot des Pagespeed Insights (abgerufen am 11.09.2021)

Pingdom:
Um tiefergreifende Einblicke zu erhalten, wird auch hier das Testwerkzeug „Pingdom Website Speed Test“ hinzugezogen. Abbildung 4.45 zeigt das Ergebnis. Laut diesem Test beträgt die Datenmenge für die Umsetzung mit Bootstrap 664.5 Kilobyte, die Ladezeit beträgt 483 Millisekunden und der Server muss 21 Abfragen verarbeiten.

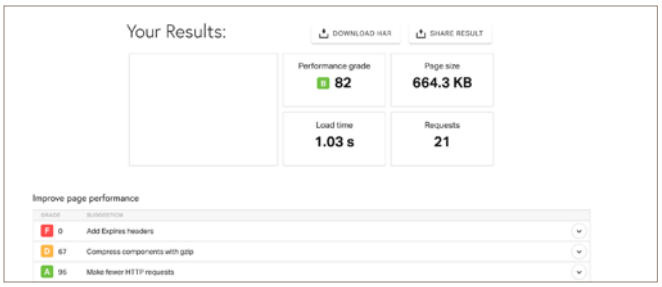


Abb. 4.45: Pingdom Speed Test Ergebnis, Quelle: https://tools.pingdom.com/#5f06f275d5400000 (abgerufen am 25.09.2021)

Chrome Entwickler-Ansicht: Network
Das Ergebnis mit Bootstrap wird ebenfalls mithilfe des Wasserfalldiagramms aus dem Network-Bereich der Entwickler-Ansicht ausgewertet (vgl. Abb. 4.46). Die Reihenfolge, welche Dateien als Erstes und welche als Letztes geladen werden, hat sich im Vergleich zu den beiden vorherigen Umsetzungen nicht verändert.

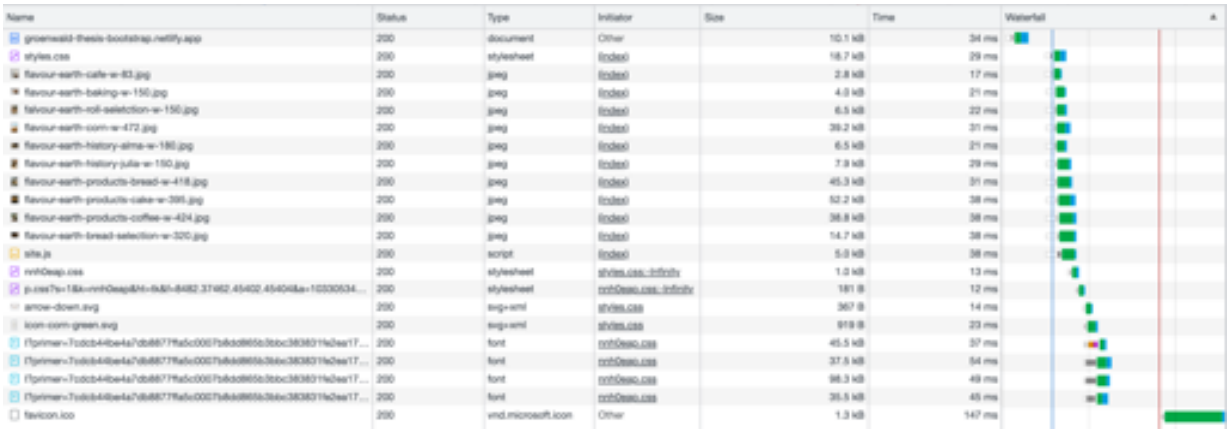


Abb. 4.46: Wasserfalldiagramm im Network-Bereich,
Quelle: https://613ca71473ad54659fc6b665--groenwald-thesis-bootstrap.netlify.app/ (abgerufen am 11.09.2021)

Das CSS-Stylesheet weist hier eine Dateigröße von 18,7 Kilobyte und das gebündelte JavaScript eine Dateigröße von 5,0 Kilobyte auf. Diese Daten dienen als Vergleichspunkt für den folgenden Abschnitt.

4.4.12 DATEIGRÖSSEN

Hier wird der Vergleich zwischen den optimierten und nicht optimierten CSS- und JavaScript-Dateien gezogen. Im vorherigen Abschnitt *4.4.11.2 Werkzeuge zum Testen der Performance* wurden bereits die optimierten Dateigrößen vorgestellt. Um herausfinden zu können, wie groß die Dateigrößen der nicht optimierten Dateien sind, wurde das Wasserfalldiagramm aus dem Network-Bereich ebenfalls für die Webseitenversion ohne Optimierung und Komprimierung ausgewertet. Das Wasserfalldiagramm wird in Abbildung 4.47 dargestellt und dort kann abgelesen werden, dass die nicht optimierte Version von `styles.css` 90,3 Kilobyte und die nicht optimierte Version von `site.js` 6,1 Kilobyte an Dateigröße umfasst.

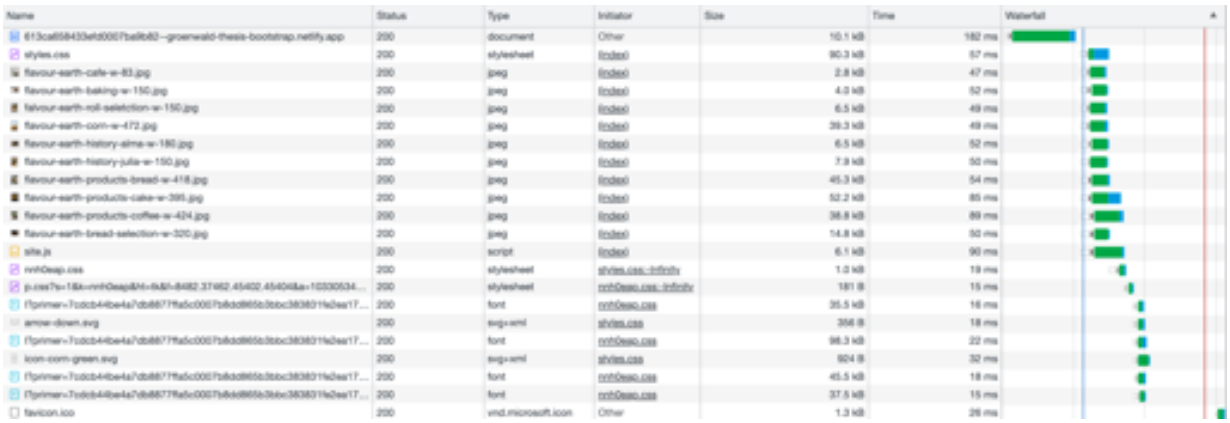


Abb. 4.47: Wasserfalldiagramm mit nicht optimierten Dateien,
Quelle: https://613ca658433efd0007ba9b82--groenwald-thesis-bootstrap.netlify.app/ (abgerufen am 11.09.2021)

Zur Gegenüberstellung der nicht optimieren und optimierten Dateigrößen wurde die Tabelle 4.3 erstellt.

	optimiert	nicht optimiert	Differenz
styles.css	90.300 Byte	18.700 Byte	71.600 Byte
site.js	6.100 Byte	5.000 Byte	1.100 Byte

Tabelle 4.3: Vergleich Dateigrößen Bootstrap, Quelle: Eigene Darstellung

Schlussendlich konnten mithilfe der Komprimierung und dem Löschen von nicht verwendeten Codeangaben im Stylesheet 71,6 Kilobyte und innerhalb des JS-Codes 1,1 Kilobyte eingespart werden.

4.4.13 SCHWIERIGKEITEN BEI DER UMSETZUNG

Während der Umsetzung der individuellen Gestaltung sind mehrere unterschiedliche Schwierigkeiten entstanden, die nicht allein mit Bootstrap gelöst werden konnten. In vielen verschiedenen Bereichen musste somit auf eigenen Quellcode zurückgegriffen werden, sodass insgesamt 18 selbst erstellte Sass-Dateien benötigt wurden. Innerhalb dieser Dateien ist entweder der bereits von Bootstrap mitgelieferter Quellcode überschrieben worden oder es mussten neue Komponenten erstellt werden.

4.4.13.1 ALLGEMEINE, GRUNDLEGENDE SCHWIERIGKEITEN

1. Problem: Breakpoint-Variablen können nicht referenziert werden
Dieses Problem tritt dann auf, wenn im selbst erstellten Quellcode Media-Queries eingesetzt werden sollen. Wie bereits in Abschnitt *4.4.2.5 Umsetzung von Responsivität* thematisiert, werden die vordefinierten Breakpoints in der mitgelieferten Variablendatei von Bootstrap definiert.

Da sich diese Breakpoints allerdings in einer Sass-Map befinden, können die Entwickelnden auf die Werte nicht zugreifen. Demnach können in selbst erstellten Media-Queries die vordefinierten Breakpoints von Bootstrap nicht referenziert werden (vgl. Abb. 4.48, Abb. 4.49, Abb. 4.50). Daraus können Probleme für die Wartbarkeit des Codes entstehen, da bei einer Veränderung innerhalb der vordefinierten Breakpoints alle selbst erstellten Media-Queries manuell angepasst werden müssen.

Um die Entwicklung zu vereinfachen, wurden die vordefinierten Breakpoints in einer eigenen Datei, wo auch die anderen Gestaltungsparameter definiert wurden (vgl. *4.4.5.1 Definition von Gestaltungsparametern*), identisch überführt.

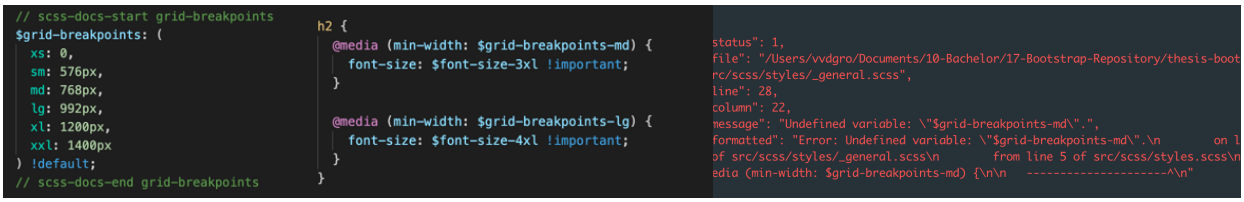


Abb. 4.48 (links): Vordefinierte Breakpoints von Bootstrap
Abb. 4.49 (mittig): Beispielhafter Sass-Quellcode für Nutzung eines Breakpoints
Abb. 4.50 (rechts): Fehlermeldung bei Versuch auf Variablenzugriff der Breakpoints
Quelle: Eigene Darstellung

2. Problem: Überschreibungen von vordefinierten Deklarationen sehr mühsam
Das nächste Problem bezieht sich auf Überschreibungen von vordefinierten Deklarationen. Um Deklarationen überschreiben zu können, muss die Spezifikation mindestens gleich oder höherwertiger als die zu

überschreibende Deklaration sein (vgl. *2.2.3 Funktionsweise von CSS*). Bootstrap verwendet innerhalb seines Quellcodes allerdings viele Verschachtelungen, sodass bestimmte Komponentendeklarationen bereits eine hohe Spezifikation aufweisen und somit schwerer zu überschreiben sind (vgl. Abb. 4.51).

Aus diesem Grund wird bei einer Überschreibung oft zu der `!important`-Anweisung gegriffen. Warum dies eigentlich verhindert werden sollte, kann ebenfalls in Abschnitt *2.2.3 Funktionsweise von CSS* nachgelesen werden.

```
.card {
  position: relative;
  display: flex;
  flex-direction: column;
  min-width: 0; // See https://github.com/twbs/bootstrap/pull/22740#i
  height: $card-height;
  word-wrap: break-word;
  background-color: $card-bg;
  background-clip: border-box;
  border: $card-border-width solid $card-border-color;
  @include border-radius($card-border-radius);
  @include box-shadow($card-box-shadow);

  > hr {
    margin-right: 0;
    margin-left: 0;
  }

  > .list-group {
    border-top: inherit;
    border-bottom: inherit;

    &:first-child {
      border-top-width: 0;
      @include border-top-radius($card-inner-border-radius);
    }

    &:last-child  {
      border-bottom-width: 0;
      @include border-bottom-radius($card-inner-border-radius);
    }
  }

  // Due to specificity of the above selector (`.card > .list-group`),
  // use a child selector here to prevent double borders.
  > .card-header + .list-group,
  > .list-group + .card-footer {
    border-top: 0;
  }
}
```

Abbildung 4.51: Verschachtelte Anweisungen, Quelle: Eigene Darstellung

4.4.13.2 SCHWIERIGKEITEN, DIE SPEZIELL FÜR DIE UMSETZUNG DIESER GESTALTUNG AUFTRETEN

In diesem Abschnitt werden Probleme aufgelistet, die explizit bei der Umsetzung des individuellen Screendesigns aufgetreten sind.

1. Problem: Individuelles Raster nicht umsetzbar
Bootstrap liefert ein vordefiniertes, 12-spaltiges Raster, wobei die Spalten alle eine identische Breite aufweisen. Für eine responsive Anwendung wird nicht die Spaltenanzahl angepasst, sondern die daran ausgerichteten Elemente müssen sich dementsprechend verhalten, indem sie sich je nach Bildschirmbreite über mehrere Spalten erstrecken. [52] Die Individualisierung des Rasters ist allerdings stark limitiert, da nur die allgemeine Spaltenanzahl, aber nicht die einzelnen Spaltenbreiten angepasst werden können. [53] Aufgrund dessen musste das in Abschnitt *3.2.3. Raster* erstellte Raster mittels eines selbst erstellten `@mixin` umgesetzt werden.

Die folgenden Abbildungen sollen zudem veranschaulichen, wie das individuelle Design des veganen Cafés umgesetzt aussieht, wenn anstatt des individuellen Rasters das 12-spaltige Raster von Bootstrap genutzt werden würde.

52 Vgl. Otto, Mark/Jacob Thornton: CSS Grid, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.1/layout/css-grid/ (abgerufen am 12.09.2021).
53 Vgl. Otto/Thornton, o. D.

Abb. 4.52 (links): Hero mit 12-spaltigen Raster
Abb. 4.53 (rechts oben): Geschichte mit 12-spaltigen Raster
Abb. 4.54 (rechts unten): Philosophie mit 12-spaltigen Raster
Quelle: Eigene Darstellung

2.Problem: Für Pseudoelemente, wie ::before und ::after, gibt es keine Komponenten oder Utilities

In Bootstrap gibt es weder eine Komponente noch bestimmte Utilities, die die Umsetzung von Pseudoelementen ermöglichen. Da in der Webseitengestaltung des vergangen Cafés Hintergründe und Kreiselemente vorgesehen sind, die mittels der Pseudoelemente umgesetzt werden sollten, muss deren Entwicklung in selbst erstelltem Quellcode passieren.

3. Problem: Positionierung der Elemente per Utilities funktionieren nicht bei der Nutzung eines individuellen Grids

Wie die Abbildung 4.55 aus der Bootstrap-Quelldatei `_grid.scss` zeigt, wirken Utilities, wie `g-col-4` und `g-col-start-1` nur, wenn sie von einem HTML-Element mit der zugewiesenen Klasse `grid` umschlossen werden. Demnach können die von Bootstrap mitgelieferten Utilities zur Positionierung von Elementen in einem Grid nicht bei der Anwendung des individuellen Rasters genutzt werden. Dies hat zur Folge, dass alle Angaben zur Spaltenanordnung, wie `grid-column`, im eigenen Quellcode definiert werden müssen.

```scss
@if $enable-cssgrid {
  .grid {
    display: grid;
    grid-template-rows: repeat(var(--#{$variable-prefix}rows, 1), 1fr);
    grid-template-columns: repeat(var(--#{$variable-prefix}columns, #{$grid-columns}), 1fr);
    gap: var(--#{$variable-prefix}gap, #{$grid-gutter-width});

    @include make-cssgrid();
  }
}
```

Abb. 4.55: Wirkungsbereich Utilities für Grid-Positionierung, Quelle: Eigene Darstellung

4.4.14 VORKENNTNISSE

In diesem Abschnitt wird untersucht, wieviel Vorwissen über die Frontend-Werkzeuge HTML, CSS und JavaScript benötigt wurden, um die individuelle Gestaltung mittels Bootstrap zu entwickeln.

4.4.14.1 BENÖTIGTE VORKENNTNISSE

Für eine Webseitenumsetzung mittels Bootstrap wird grundsätzlich wenig Vorwissen über HTML, CSS oder JavaScript benötigt. Alle benötigten HTML-Elemente können aus den Anwendungsbeispielen der Komponenten und Utilities entnommen werden und das HTML-Grundgerüst wird ebenfalls in der Dokumentation von Bootstrap angesprochen und aufgezeigt, sodass auch dieses von den Nutzenden nur kopiert und eingefügt werden muss. Für Webseiten, deren Inhalte und Gestaltung mit den vordefinierten Komponenten, Utilities und Gestaltungsparametern von Bootstrap komplett umgesetzt werden können, muss zudem kein eigenes CSS bzw. Sass geschrieben werden, sodass für deren Umsetzung weder Vorwissen über JavaScript noch CSS benötigt wird.

Sobald eine Webseite aber nicht dem standardisierten Aufbau von Bootstrap entspricht und Komponenten überschrieben, erweitert oder neu hinzugefügt werden müssen, wird das entsprechende Vorwissen über die Frontend-Werkzeuge vorausgesetzt. Dies zeigt sich besonders bei Codeabschnitten, die aufgrund der Kaskade und Spezifität von CSS nur mithilfe von expliziten Anweisungen oder `!important`-Angaben überschrieben werden können.

4.4.14.2 BENÖTIGTE ZEIT ZUR UMSETZUNG

Um den Zeitaufwand dieser Umsetzung einschätzen zu können, wurde wieder die Anwendung TMetric hinzugezogen. Demnach wurde für die Entwicklung mit Bootstrap insgesamt 28 Stunden und 21 Minuten aufgewendet.

5 VERGLEICH

Nachdem nun alle Herangehensweisen anhand eines individuellen Screendesigns umgesetzt wurden und jeder Entwicklungsansatz tiefgreifend erörtert wurde, können diese miteinander verglichen werden. Dieser Vergleich findet anhand verschiedener Vergleichspunkte statt. Jeder dieser Aspekte wurde zuvor einzeln im vollen Umfang betrachtet, sodass die daraus geschlossenen Erkenntnisse und Kernaussagen nun gegenübergestellt und in einem Zusammenhang gebracht werden können. Diese Gegenüberstellung dient als Basis des Fazits und zur Beantwortung der Leitfrage, **welche Vor- und Nachteile die unterschiedlichen Herangehensweisen an CSS anhand einer individuell gestalteten Webseite bieten**.

5.1 STÄRKEN & SCHWÄCHEN DER FUNKTIONSWEISEN

Der erste Vergleichspunkt bezieht sich auf die unterschiedlichen Funktionsweisen der unterschiedlichen Herangehensweisen.

Die Herangehensweise mit handgeschriebenen CSS repräsentiert die ursprüngliche Art und Weise, Webseiten zu entwickeln. Während der Entwicklung konnten auf alle grundlegenden CSS-Funktionen und -Eigenschaften zurückgegriffen werden, sodass keine Einschränkungen vorhanden waren. Mittels der Einbindung von Sass konnten die grundlegenden Funktionen sogar noch erweitert werden (vgl. *4.2.1.1 Prozessoren*). Aufgrund der nicht vorhandenen Einschränkungen entstand die volle Entscheidungsmacht über bestimmte Entwicklungsentscheidungen, welches beispielsweise der beste Umsetzungsweg ist oder welche Methodik die passende für diese Anwendung ist.

Innerhalb der Umsetzung mit dem CSS-Framework TailwindCSS wird diese volle Entscheidungsmacht des Entwickelnden bereits durch die mitgelieferten Utility-Klassen eingeschränkt, da nicht alle CSS-Eigenschaften und Werte vom Framework selbst zur Verfügung gestellt werden, wie beispielsweise die fehlende Möglichkeit zur Einbindung von Hintergrundbildern (vgl. *4.3.12 Schwierigkeiten bei der Umsetzung*). Die größte Einschränkung entstand jedoch dadurch, dass mit TailwindCSS nicht alle CSS-Funktionen abgebildet werden konnten, sodass Verschachtelungen von Selektoren und Kombinatoren nicht möglich waren und dadurch bestimmte Abfragen nicht getätigt werden konnten (vgl. *4.3.8 Progressive Enhancement & JavaScript*).

Allerdings bietet TailwindCSS aufgrund ihrer speziellen Funktionsweise auch viele Vorteile gegenüber der ursprünglichen Herangehensweise. In Abschnitt *2.4.2.2 Nutzen von Utility-Klassen* wurden bereits einige allgemeine

Vorteile vorgegriffen, wie die Vereinfachung des Debuggens oder der geringe Quellcodezuwachs während der Entwicklung von neuen Inhaltselementen. Zusätzlich dazu kann als weiterer Vorteil genannt werden, dass das HTML-Dokument während der Entwicklung nicht verlassen werden muss, sodass ein hin und her wechseln zwischen unterschiedlichen Dateien verhindert wird. Zudem wird dadurch ermöglicht, dass sowohl die Struktur als auch die Gestaltung der Webseite auf einen Blick erkennbar und ablesbar ist. Da durch die Utility-Klassen auch kaum bis gar kein eigener Quellcode benötigt wird, werden große, tief verschachtelte Dateistrukturen vermieden (vgl. *4.2.1.3 Methodik & Dateistruktur*).

Auch in der Entwicklung mit dem Komponenten basierten CSS-Framework Bootstrap konnte eine Limitierung der CSS-Funktionen festgestellt werden, da bereits alle Komponenten, Utilities und Gestaltungsangaben fertig und abgerundet mitgeliefert werden. Daraus entsteht aber auch ein unglaublicher Vorteil von Bootstrap gegenüber den beiden anderen Herangehensweisen, da mit der Nutzung von komplett fertig ausgebauten, bereits entwickelten, auf Browser-Bugs getesteten und optimierten Komponenten eine große Zeiteinsparung in der Entwicklung entstehen kann. Dieser Vorteil wird wiederum durch die Unflexibilität des Frameworks geschmälert, da die Entscheidungsmacht des Entwickelnden durch die Vordefinierungen eingeschränkt wird. Dadurch, dass innerhalb der Umsetzung der individuellen Gestaltung auch nur vier der eigentlich 24 mitgelieferten Komponenten von Bootstrap einen Anwendungszweck gefunden haben, konnten die Vorteile zudem nicht ausgereizt werden (vgl. *4.4.6 Verwendete Komponenten*). Aufgrund der Unflexibilität, der hohen Einschränkung und durch die Überladung des CSS-Frameworks mit unbrauchbaren Elementen, überwiegen die Nachteile von Bootstrap im Vergleich zu den beiden anderen Ansätzen, sodass die Funktionsweise des Komponenten basierten Ansatzes am wenigsten überzeugen konnte.

5.2 TOOLING-AUFWAND

Da der erste Vergleichspunkt etwas allgemeiner gehalten wurde, soll mit den folgenden Aspekten tieferliegende, technischere Punkte angesprochen und gegenübergestellt werden.

In der Entwicklung mit reinem, handgeschriebenen CSS sind die Entwickelnden nicht auf bestimmte Werkzeuge oder Abhängigkeiten angewiesen und sind demnach komplett frei in der Entscheidung, wie viel Zeit und Aufwand ins Tooling einfließen soll. Dadurch entsteht ein großer Vorteil gegenüber den CSS-Frameworks, da diese Entscheidung auch von Anwendung zu Anwendung variieren und anders gestaltet werden kann. Allerdings wurde für diese Umsetzung die explizite Entscheidung gefällt, einen gewissen

Aufwand zu betreiben, um den bestmöglichen Ausgangspunkt für die Entwicklungen erreichen zu können. Aus diesem Grund wurde nicht nur der Präprozessor Sass, sondern auch `postcss` als Postprozessor hinzugezogen. Neben diesen beiden Optimierungsmöglichkeiten wurden zudem weitere Werkzeuge eingesetzt, sodass insgesamt zehn Abhängigkeiten und zwei Konfigurationsobjekte [1] genutzt wurden. Diese können in neun Skripten aufgerufen und angewendet werden (vgl. *4.2.1 Werkzeuge & Tooling*).

Im Gegensatz zu reinem CSS benötigt eine Umsetzung mit TailwindCSS bestimmte Abhängigkeiten, um eine korrekte Funktionsweise sicherstellen zu können. Dadurch entsteht ein Nachteil, da die Entwickelnden keine Entscheidungsmacht mehr darüber haben, ob sie einen entsprechenden Tooling-Aufwand betreiben wollen oder nicht (vgl. *4.3.3.1 Werkzeuge von TailwindCSS*).

Neben den drei unbedingt notwendigen Abhängigkeiten werden in der Umsetzung mit dem Utility basierten CSS-Framework sechs weitere geladen, die wiederum in sieben Skripte aufgerufen werden können (vgl. *4.3.3.2 Weitere Abhängigkeiten*). Dementsprechend wurden im Gegensatz zur vorherigen Umsetzung zwei Abhängigkeiten und zwei Konfigurationsobjekte [2] weniger geladen, sowie zwei Skripte weniger erstellt. In diesem Vergleich ist der Tooling-Aufwand von TailwindCSS zwar erzwungen, aber es werden grundsätzlich weniger Importe und Skripte geladen als mit CSS. Dies hat auch einen bestimmten Grund, da für TailwindCSS keine `stylelint`-Plugins (die in diesem Zusammenhang als Konfigurationsobjekte bezeichnet werden) vorhanden sind und somit nicht installiert werden müssen. Dieser Aspekt wird jedoch im folgenden Abschnitt *5.8.2 Struktur & Syntax* genauer betrachtet.

Wie TailwindCSS benötigt auch Bootstrap eine gewisse Anzahl an Abhängigkeiten, um alle Funktionen korrekt ausführen zu können. Im Vergleich zu den anderen Herangehensweisen ist der Tooling-Aufand hier am höchsten, denn neben den drei unbedingt benötigten Abhängigkeiten werden zusätzlich zehn weitere plus ein zusätzliches Konfigurationsobjekt [3] geladen (vgl. *4.4.3 Werkzeuge & Tooling*). Demnach werden im Vergleich zu CSS zwei und im Vergleich zu TailwindCSS vier Installationen mehr geladen. Die Anzahl der Skripte ist jedoch identisch zu der Herangehensweise mit handgeschriebenen CSS und weist somit nur zwei Skripte mehr auf als die Umsetzung mit TailwindCSS.

1 Mit Konfigurationsobjekte sind Konfigurationen für die Abhängigkeit stylelint gemeint – sprich Regelsammlungen, die wie Abhängigkeiten per npm installiert werden müssen.

2 Siehe Fußnote 1

3 Siehe Fußnote 1

Die folgende Tabelle veranschaulicht den Vergleich der Abhängigkeiten:

	CSS	TailwindCSS	Bootstrap
Benötigte Abhängigkeiten	0	3	3
Zusätzliche Abhängigkeiten	10	5	10
Konfigurationsobjekte insgesamt	2	0	1
Installationen insgesamt	12	8	14
Skripte insgesamt	9	7	9

Tabelle 5.1: Vergleich der Abhängigkeiten, Quelle: Eigene Darstellung

Anhand dieser Tabelle und den gemachten Erkenntnissen kann geschlussfolgert werden, dass der Tooling-Aufwand für diese explizite Umsetzung mit dem CSS-Framework TailwindCSS zwar am geringsten ist, aber im Vergleich zu der Umsetzung mit reinem CSS erzwungen und erwartet wird. Wenn nur die benötigten Abhängigkeiten geladen werden würden, würde die ursprüngliche Herangehensweise die wenigsten externen Werkzeuge aufweisen. Der höchste Tooling-Aufwand muss allerdings innerhalb des dritten Ansatzes getätigt werden. Sobald das Ergebnis der Bootstrap-Umsetzung auf einer Domain veröffentlicht werden soll, müssen zu den schon angesprochenen drei unbedingt notwendigen Abhängigkeiten ebenfalls Werkzeuge wie `webpack` und `webpack-cli` genutzt werden. Dadurch erhöht sich der Aufwand umso mehr (vgl. *4.4.11.1 Optimierungsmöglichkeiten*) und stellt somit eine große Hürde dar.

5.3 VORDEFINIERTES GESTALTUNGSSYSTEM

Innerhalb dieses Abschnittes werden die durch die CSS-Frameworks mitgelieferten Gestaltungssysteme betrachtet und erörtert, ob dadurch ein Vorteil entsteht oder nicht.

In der Umsetzung mit handgeschriebenen CSS kann auf kein vordefiniertes Gestaltungssystem zurückgegriffen werden. Dadurch können alle Werte und alle CSS-Eigenschaften wie gewünscht eingesetzt werden. Jedoch ist dadurch auch keine Vereinheitlichung oder Konsistenz vorgegeben, sodass ein eigenes Gestaltungssystem erstellt werden muss. So liegt es auch in der Verantwortung des Entwickelnden, ob beispielsweise Größen in rem oder in Pixel definiert werden oder ob eine Benennungskonvention genutzt wird (vgl. *4.2.2 Umsetzung der Gestaltungsangaben*).

TailwindCSS hingegen liefert bewusst nur eine eingeschränkte Auswahl an Utility-Klassen, da dadurch ein Gestaltungssystem mit gewissen Benennungskonventionen und Wertevorgaben entsteht. Auf Basis dessen

wird eine Vereinheitlichung und Konsistenz gewährleistet. Beispielsweise werden Entwickelnde dazu ermutigt, saubere rem Werte zu nutzen (vgl. *4.3.4 Vordefiniertes Gestaltungssystem*). Die Benennungskonvention der Utilities kann aber auch ein Nachteil darstellen, da sich die Werteangaben der Utility-Klassen nicht direkt in deren zugewiesene rem- oder Pixel-Werte überführen lassen. (vgl. „Spacing-Sytem“ in *4.3.4 Vordefiniertes Gestaltungssystem*). Da die Größenangaben des Screendesigns weiterhin häufig in Pixel vorliegen, wird dadurch mehr Zeit benötigt, die Pixelangaben in das „Spacing-System“ zu übersetzen (vgl. Tabelle 5.2).

Bootstrap liefert ebenfalls ein Gestaltungssystem, welches aus einem bereits komplett ausgebauten Gestaltungsansatz besteht. In diesem Gestaltungsansatz sind bereits alle Designparameter von Farben über Schriftgrößen zu Übergängen festgelegt (vgl. *4.4.4 Vordefiniertes Gestaltungssystem*). Dieser führt zwar zu einer Konsistenz und zu einer Vereinheitlichung aller Angaben, jedoch bietet dieses Gestaltungssystem keine Flexibilität und die Individualisierung der Umsetzung wird dadurch deutlich erschwert. Die Chance, dass dadurch ein großer Teil der Persönlichkeit der eigenen Webseite verloren geht, ist im Vergleich zu einer Umsetzung mit TailwindCSS oder CSS deutlich höher. Gleichzeitig besteht auch innerhalb dieses CSS-Frameworks die bereits angesprochene Problematik der Werteübersetzung (vgl. Tabelle 5.2). Da für die Entwicklung der individuellen Webseite die vordefinierte Sass-Map der Größenangaben überschrieben wurde, entspricht der gezeigte Wert in der folgenden Tabelle nicht dem von Bootstrap vordefinierten Wert. Allerdings wurde die Benennungskonvention von Bootstrap innerhalb der Sass-Map beibehalten, sodass die Werte von klein nach groß durchnummeriert wurden. Dabei wird der eigentliche Abstand zwischen den Werten ignoriert, sodass kein System hinter der Benennung steht (vgl. Abb. 4.34). Aus diesem Grund ist die Übersetzung dieser Utilities in entsprechende Pixelangaben um einiges schwieriger als in TailwindCSS.

Designvorgabe	CSS	TailwindCSS	Bootstrap
16px/1rem	spacing-16	{utility}-4	{utility}-3
20px/1.25rem	spacing-20	{utility}-5	{utility}-4
60px/3.75rem	spacing-60	{utility}-15	{utility}-8
160px/10rem	spacing-160	{utility}-40	{utility}-12

Tabelle 5.2: Vergleich der Wertebenennungen, Quelle: Eigene Darstellung

5.4 UMSETZUNG DER GESTALTUNGSPARAMETER

Der zweite Punkt, anhand dessen die unterschiedlichen Herangehensweisen an CSS verglichen werden sollen, ist die Umsetzung der Gestaltungsparameter.

Durch die Einbindung von Sass ist es während der Entwicklung mit handgeschriebenen CSS möglich, Variablen zu definieren, sodass die Wartbarkeit der Umsetzung deutlich leichter wird (vgl. *4.2.1.1 Prozessoren*). Bei dieser Umsetzungsmöglichkeit ist es in der Entscheidungsmacht des Entwickelnden, wie viele Variablen für die Entwicklung benötigt und definiert werden sollen. In dieser Umsetzung wurden die definierten Variablen in einer einzelnen Sass-Datei definiert, sodass es bei einer möglichen Anpassung der Variablen nur einen Anhaltspunkt zur Wartung gibt.

Die Definition der Gestaltungsparameter von TailwindCSS grenzt sich von der zuvor beschriebenen Variante deutlich ab. Die Gestaltungsparameter des Utility-Klassen basierten CSS-Frameworks werden innerhalb der Konfigurationsdatei `tailwind.config.js` festgehalten und die Utilities werden anhand dessen generiert (vgl. *4.3.5 Umsetzung der Gestaltungsangaben*). Die Individualisierbarkeit dieser Datei sorgt dafür, dass alle benötigten Gestaltungsparameter aus Abschnitt *3.2 Gestaltung* in Utilities umgewandelt werden konnten. Dadurch haben die Umsetzungen mit TailwindCSS und mit handgeschriebenen CSS die Gemeinsamkeit, dass es nur eine einzige Datei gibt, in der die Gestaltungsparameter definiert werden.

Auch in der Umsetzung mit Bootstrap werden die individuellen Gestaltungsparameter in einer Sass-Datei definiert. Da die Anpassung der Gestaltungsparameter allerdings nicht in der von Bootstrap mitgelieferten Variablendatei vorgenommen werden kann, muss eine eigene Sass-Datei erstellt werden, die die vordefinierten Bootstrap-Variablen überschreibt (vgl. *4.4.5.1 Definition von Gestaltungsparametern*). Dadurch wird nicht nur der Überschreibungsprozess verkompliziert, sondern auch die Wartbarkeit verschlechtert sich, da sich aufgrund dessen zwei Anlaufstellen für mögliche Variablendefinitionen auftun. Der Aspekt, dass Bootstrap insgesamt 1639 Zeilen an vordefinierten Variablen mitliefert, verschlechtert diese Situation nur noch. Zudem müssen die selbst erstellten Variablen entgegengesetzt der eigentlichen Kaskade von CSS eingebunden werden, damit die Überschreibung überhaupt greift (vgl. *4.4.5.2 Schwierigkeiten während der Umsetzung von Gestaltungsangaben*). Dies sind alles Gründe, warum Bootstrap in diesem Aspekt einen großen Nachteil im Vergleich zu den anderen Herangehensweisen aufweist.

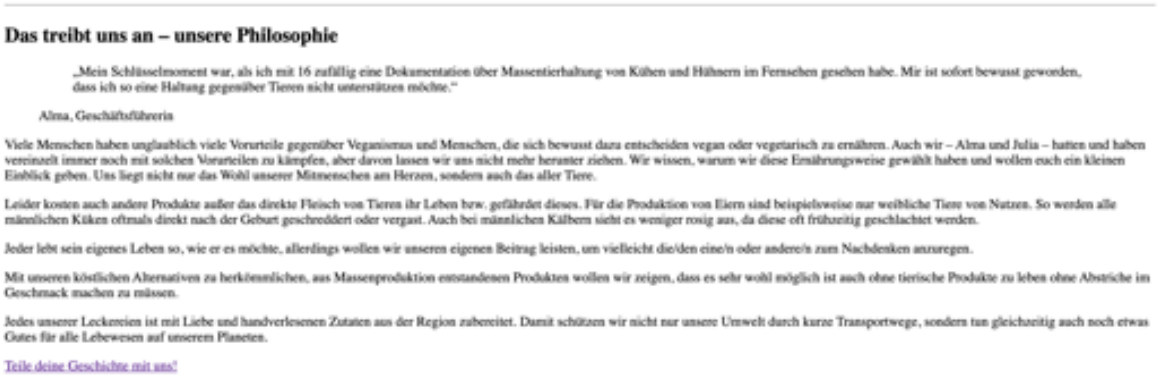
5.5 REALISIERUNG VON CROSS-BROWSER-RENDERING

Als Nächstes soll betrachtet werden, wie Cross-Browser-Rendering in allen drei Herangehensweisen realisiert wurde. Die Ergebnisse dessen werden anhand von Abbildungen verglichen. Zusätzlich wird die Browserkompatibilität aller Herangehensweisen hinzugezogen.

5.5.1 ANPASSEN DER USER-AGENT-STYLESHEETS

Um Cross-Browser-Rendering zu gewährleisten, wurden bei allen drei Herangehensweisen die User-Agent-Stylesheets der verschiedenen Browser untereinander angepasst. Dabei haben alle drei Herangehensweisen andere Ansätze gewählt. Um die Unterschiede dieser Ansätze zu veranschaulichen, wurde ein identischer Inhaltsbereich nur mit den entsprechenden CSS-Deklarationen der User-Agent-Stylesheet-Anpassungen geladen.

Da in der Umsetzung mit handgeschrieben CSS keine Inhalte vordefiniert sind, ist es freigestellt, ob die User-Agent-Stylesheets berücksichtigt werden sollen oder nicht. Für diese Anwendung wurde die Abhängigkeit `normalize-sass` genutzt, da sie im Gegensatz zu anderen Abhängigkeiten die Browser-Deklarationen untereinander anpasst und nicht zurücksetzt. Wie zuvor in Abschnitt *4.2.3 Realisierung von Cross-Browser Rendering* präzisiert, bleiben die visuellen Kennzeichnungen der entsprechenden HTML-Elemente dadurch erhalten. Dies lässt sich auch anhand von Abbildung 5.1 erkennen, da die Überschrift, das Zitat und der Link entsprechende visuelle Abhebungen aufweisen.



Das treibt uns an – unsere Philosophie

„Mein Schlüsselmoment war, als ich mit 16 zufällig eine Dokumentation über Massentierhaltung von Kühen und Hühnern im Fernsehen gesehen habe. Mir ist sofort bewusst geworden, dass ich so eine Haltung gegenüber Tieren nicht unterstützen möchte."

Alma, Geschäftsführerin

Viele Menschen haben unglaublich viele Vorurteile gegenüber Veganismus und Menschen, die sich bewusst dazu entscheiden vegan oder vegetarisch zu ernähren. Auch wir – Alma und Julia – hatten und haben vereinzelt immer noch mit solchen Vorurteilen zu kämpfen, aber davon lassen wir uns nicht mehr beranter ziehen. Wir wissen, warum wir diese Ernährungsweise gewählt haben und wollen euch ein kleinen Einblick geben. Uns liegt nicht nur das Wohl unserer Mitmenschen am Herzen, sondern auch das aller Tiere.

Leider kosten auch andere Produkte außer das direkte Fleisch von Tieren ihr Leben bzw. gefährdet dieses. Für die Produktion von Eiern sind beispielsweise nur weibliche Tiere von Nutzen. So werden alle männlichen Küken oftmals direkt nach der Geburt geschreddert oder vergast. Auch bei männlichen Kälbern sieht es weniger rosig aus, da diese oft frühzeitig geschlachtet werden.

Jeder lebt sein eigenes Leben so, wie er es möchte, allerdings wollen wir unseren eigenen Beitrag leisten, um vielleicht die/den eine/n oder andere/n zum Nachdenken anzuregen.

Mit unseren köstlichen Alternativen zu herkömmlichen, aus Massenproduktion entstandenen Produkten wollen wir zeigen, dass es sehr wohl möglich ist auch ohne tierische Produkte zu leben ohne Abstriche im Geschmack machen zu müssen.

Jedes unserer Leckereien ist mit Liebe und handverlesenen Zutaten aus der Region zubereitet. Damit schützen wir nicht nur unsere Umwelt durch kurze Transportwege, sondern tun gleichzeitig auch noch etwas Gutes für alle Lebewesen auf unserem Planeten.

Teile deine Geschichte mit uns!

Abb. 5.1. Cross-Browser-Rendering CSS, Quelle: Eigene Darstellung

Demgegenüber steht das CSS-Framework TailwindCSS, welches mit dessen „Preflight" bereits eine eingebaute Anpassung bzw. Zurücksetzung der User-Agent-Stylesheets mitliefert (vgl. *4.3.6.1 Preflight*). Dadurch entsteht ein großer Vorteil, da die Anpassung der User-Agent-Stylesheets somit nicht mehr in der Eigenverantwortung des Entwickelnden liegt und Zeit eingespart werden kann. Allerdings wird in Abbildung 5.2 veranschaulicht, dass die HTML-

Elemente keine entsprechenden visuellen Kennzeichnungen aufweisen. Diese Zurücksetzung der User-Agent-Stylesheet-Deklarationen kann dazu führen, dass bei fehlenden Gestaltungsangaben den Anwendenden beispielsweise Links nicht erkenntlich gemacht werden.

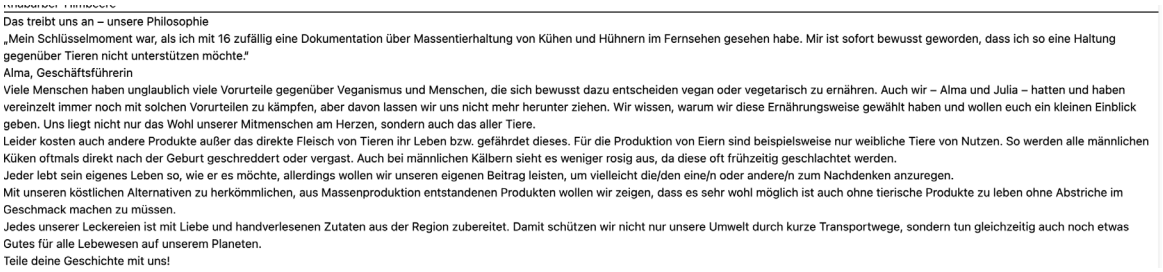

Das treibt uns an – unsere Philosophie
„Mein Schlüsselmoment war, als ich mit 16 zufällig eine Dokumentation über Massentierhaltung von Kühen und Hühnern im Fernsehen gesehen habe. Mir ist sofort bewusst geworden, dass ich so eine Haltung gegenüber Tieren nicht unterstützen möchte."
Alma, Geschäftsführerin
Viele Menschen haben unglaublich viele Vorurteile gegenüber Veganismus und Menschen, die sich bewusst dazu entscheiden vegan oder vegetarisch zu ernähren. Auch wir – Alma und Julia – hatten und haben vereinzelt immer noch mit solchen Vorurteilen zu kämpfen, aber davon lassen wir uns nicht mehr herunter ziehen. Wir wissen, warum wir diese Ernährungsweise gewählt haben und wollen euch ein kleinen Einblick geben. Uns liegt nicht nur das Wohl unserer Mitmenschen am Herzen, sondern auch das aller Tiere.
Leider kosten auch andere Produkte außer das direkte Fleisch von Tieren ihr Leben bzw. gefährdet dieses. Für die Produktion von Eiern sind beispielsweise nur weibliche Tiere von Nutzen. So werden alle männlichen Küken oftmals direkt nach der Geburt geschreddert oder vergast. Auch bei männlichen Kälbern sieht es weniger rosig aus, da diese oft frühzeitig geschlachtet werden.
Jeder lebt sein eigenes Leben so, wie er es möchte, allerdings wollen wir unseren eigenen Beitrag leisten, um vielleicht die/den eine/n oder andere/n zum Nachdenken anzuregen.
Mit unseren köstlichen Alternativen zu herkömmlichen, aus Massenproduktion entstandenen Produkten wollen wir zeigen, dass es sehr wohl möglich ist auch ohne tierische Produkte zu leben ohne Abstriche im Geschmack machen zu müssen.
Jedes unserer Leckereien ist mit Liebe und handverlesenen Zutaten aus der Region zubereitet. Damit schützen wir nicht nur unsere Umwelt durch kurze Transportwege, sondern tun gleichzeitig auch noch etwas Gutes für alle Lebewesen auf unserem Planeten.
Teile deine Geschichte mit uns!

Abb. 5.2. Cross-Browser-Rendering TailwindCSS, Quelle: Eigene Darstellung

Auch Bootstrap hat einen Vorteil gegenüber der Umsetzung mit CSS, da es wie TailwindCSS eine eingebaute Anpassung der User-Agent-Stylesheets mitliefert. Das vordefinierte „Reboot" von Bootstrap basiert auf der viel verwendeten Normalisierungsabhängigkeit `normalize.css` (vgl. *4.4.7.1 Reboot*). Wie in Abbildung 5.3 dargestellt, bleiben auch hier die visuellen Kennzeichnungen der HTML-Elemente erhalten.


Das treibt uns an – unsere Philosophie

„Mein Schlüsselmoment war, als ich mit 16 zufällig eine Dokumentation über Massentierhaltung von Kühen und Hühnern im Fernsehen gesehen habe. Mir ist sofort bewusst geworden, dass ich so eine Haltung gegenüber Tieren nicht unterstützen möchte."

Alma, Geschäftsführerin

Viele Menschen haben unglaublich viele Vorurteile gegenüber Veganismus und Menschen, die sich bewusst dazu entscheiden vegan oder vegetarisch zu ernähren. Auch wir – Alma und Julia – hatten und haben vereinzelt immer noch mit solchen Vorurteilen zu kämpfen, aber davon lassen wir uns nicht mehr herunter ziehen. Wir wissen, warum wir diese Ernährungsweise gewählt haben und wollen euch ein kleinen Einblick geben. Uns liegt nicht nur das Wohl unserer Mitmenschen am Herzen, sondern auch das aller Tiere.

Leider kosten auch andere Produkte außer das direkte Fleisch von Tieren ihr Leben bzw. gefährdet dieses. Für die Produktion von Eiern sind beispielsweise nur weibliche Tiere von Nutzen. So werden alle männlichen Küken oftmals direkt nach der Geburt geschreddert oder vergast. Auch bei männlichen Kälbern sieht es weniger rosig aus, da diese oft frühzeitig geschlachtet werden.

Jeder lebt sein eigenes Leben so, wie er es möchte, allerdings wollen wir unseren eigenen Beitrag leisten, um vielleicht die/den eine/n oder andere/n zum Nachdenken anzuregen.

Mit unseren köstlichen Alternativen zu herkömmlichen, aus Massenproduktion entstandenen Produkten wollen wir zeigen, dass es sehr wohl möglich ist auch ohne tierische Produkte zu leben ohne Abstriche im Geschmack machen zu müssen.

Jedes unserer Leckereien ist mit Liebe und handverlesenen Zutaten aus der Region zubereitet. Damit schützen wir nicht nur unsere Umwelt durch kurze Transportwege, sondern tun gleichzeitig auch noch etwas Gutes für alle Lebewesen auf unserem Planeten.

Teile deine Geschichte mit uns!

Abb. 5.3. Cross-Browser-Rendering Bootstrap, Quelle: Eigene Darstellung

Die aus dem Vergleich gewonnenen Erkenntnisse zeigen demnach ein klares Ergebnis. Da Bootstrap die einzige Herangehensweise ist, die eine korrekte, eingebaute Anpassung der User-Agent-Stylesheets mitliefert, hat es einen klaren Vorteil gegenüber CSS und TailwindCSS.

5.5.2 BROWSERKOMPATIBILITÄT

Ein großer Vorteil an der Umsetzung mit handgeschriebenen CSS ist die Browserkompatibilität, da die grundlegenden CSS-Funktionen in (fast) allen Browsern ausgeführt werden können und höchstens Probleme mit neuen CSS-Eigenschaften auftreten. Konträr dazu stehen die CSS-Frameworks TailwindCSS und Bootstrap, deren aktuelle Versionen nicht

mehr von dem Browser Internet Explorer 11 unterstützt werden (vgl. *4.3.6.2 Browserkompatibilität, 4.4.7.2 Browserkompatibilität*). Laut einer Statistik von Statista haben im August 2021 nur 1,65% der Bevölkerung IE als Browser verwendet.[4] Trotzdem werden diese 1,65% von der Nutzung einer Webseite, die mit TailwindCSS oder Bootstrap umgesetzt wurde, ausgeschlossen. Allerdings nutzen die beiden CSS-Frameworks die Abhängigkeit `autoprefixer`, sodass auf jeden Fall die Kompatibilität der CSS-Eigenschaften gewährleistet wird. In der Herangehensweise mit reinem CSS ist dies dem Entwickelnden selbst überlassen.

5.6 RESPONSIVE WEBDESIGN & MOBILE FIRST

Während der Verwendung von reinem CSS ist es den Entwickelnden freigestellt, welche Methode sie verwenden wollen, um eine Responsivität zu erreichen. Demnach wurde in dieser Umsetzung die Mobile First Strategie aus Abschnitt *4.1.1 Responsive Webdesign* verfolgt. Die Umbruchpunkte des Designs wurden in dieser Umsetzung von häufig genutzten Breakpoints inspiriert (vgl. *4.2.4 Responsive Webdesign & Mobile First*).

Auch TailwindCSS verfolgt den Mobile First Ansatz, der jedoch in der Konfigurationsdatei angepasst und individualisiert werden kann. Die bereits vordefinierten Breakpoint-Utilities lassen sich in der Konfigurationsdatei von TailwindCSS komplett anpassen und/oder überschreiben (vgl. *4.3.2.2 Umsetzung von Responsivität*).

Bootstrap verwendet ebenfalls die Mobile First Strategie, um Responsivität gewährleisten zu können. Diese Strategie kann, falls gewünscht, angepasst und umgeändert werden. Der Vorteil von Bootstrap besteht allerdings darin, dass die Responsivität bereits innerhalb der vorbestimmten Komponenten und Utilities mitberücksichtigt wird. Standardisiert werden allerdings nur eine geringe Anzahl der Utilities auch für responsive Zwecke mitgeliefert, sodass diese Konfigurationen oft im Nachhinein angepasst werden müssen (vgl. *4.4.5.3 Utilities anpassen*). Nicht nur durch die fehlenden responsiven Utilities wird die Entwicklung erschwert, sondern auch durch die fehlende Möglichkeit die vordefinierte Sass-Map der Breakpoints innerhalb des selbst erstellten Quellcodes zu referenzieren (vgl. *4.4.13.2 Schwierigkeiten, die speziell für die Umsetzung dieser Gestaltung auftreten*).

4 Vgl. Statista: Meistgenutzte Browser in Deutschland August 2021, in: Statista, 07.09.2021, https://de.statista.com/statistik/daten/studie/828151/umfrage/meistgenutzte-browserfamilien-im-internet-in-deutschland/ (abgerufen am 18.09.2021).

In beiden CSS-Frameworks wurden die mitgelieferten Breakpoints nicht angepasst oder überschrieben, damit diese gegenübergestellt und verglichen werden können. Um diesen Vergleich zu veranschaulichen, wurde die folgende Tabelle erstellt.

CSS	TailwindCSS	Bootstrap
640px	640px	576px
768px	768px	768px
1024px	1024px	992px
1280px	1280px	1200px
1650px	1536px	1400px

Tabelle 5.3: Vergleich der Breakpoints, Quelle: Eigene Darstellung

Aus Tabelle 5.3 lässt sich ablesen, dass sich die vordefinieren Breakpoints von Bootstrap zu den anderen Breakpoints stark unterscheiden. Dementsprechend treten Änderungen in Bootstrap bereits fast immer vorher in Kraft. Die größten Unterschiede spielen sich allerdings im letzten und somit größten Breakpoint ab, da Bootstrap dort den kleinsten und CSS den größten Wert aufweist. Insgesamt liegen zwischen den beiden Werten 250 Pixel. Dies zeigt auch Auswirkungen auf die Gestaltung, da somit die Breite des `<body>`-Elements in Bootstrap eher eingeschränkt wird, als bei den beiden anderen Herangehensweisen. Dies lässt sich auch in Abbildung 6.4 erkennen, da der gesamte Inhaltsbereich im dritten Beispiel bereits einen Rand aufweist und nicht vollflächig angezeigt wird. Dadurch geht die gewollte, vollflächige Kachelanmutung schneller verloren.

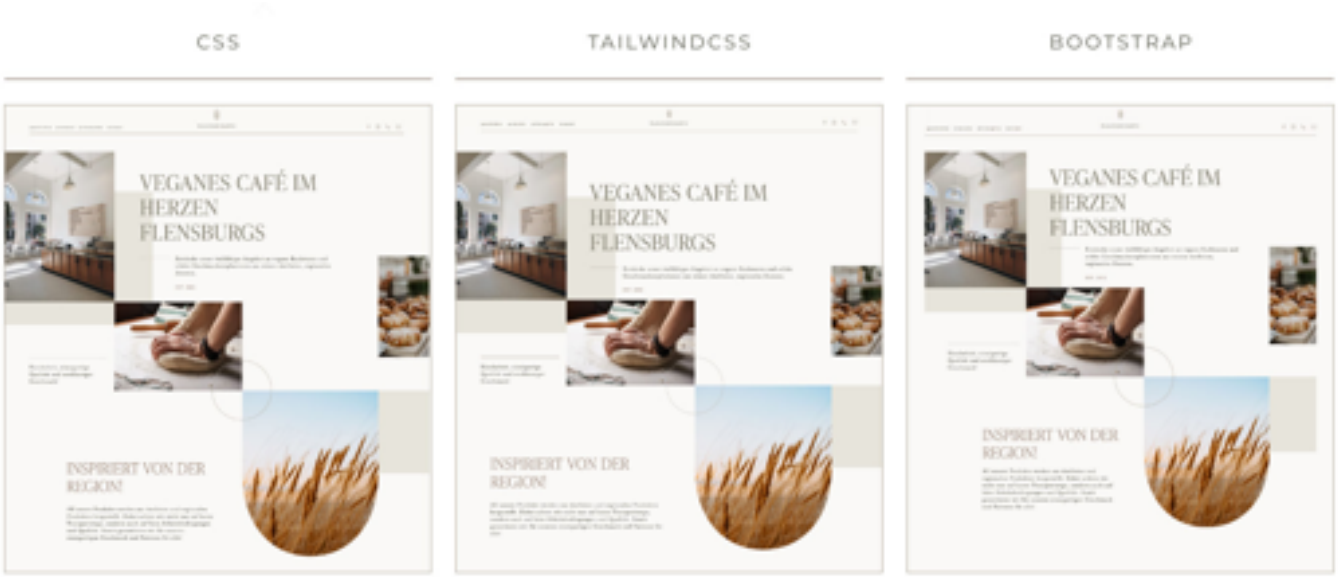


Abb. 5.4: Hero-Bereich bei 1400 Pixel Viewportbreite, Quelle: Eigene Darstellung

5.7 PROGRESSIVE ENHANCEMENT

Im folgenden Abschnitt wird verglichen, wie die verschiedenen Herangehensweisen an das Thema Progressive Enhancement herangegangen sind und ob bzw. wie es berücksichtigt wurde.

Innerhalb der Umsetzung mit CSS wurde JavaScript genutzt, um mit HTML und CSS nicht lösbare Probleme aufzulösen und um die Haptik der Webseite zu verbessern. Da in dieser Umsetzung keine grundlegenden Funktionen auf JavaScript basieren, wurde Progressive Enhancement berücksichtigt (vgl. *4.2.5. Progressive Enhancement*).

Da TailwindCSS nur vordefinierten Quellcode für CSS und nicht für JS mitliefert, sind auch in dieser Umsetzung alle Möglichkeiten offen, ob und wofür JS eingesetzt werden soll. Demnach wurden hier die gleichen JS-Dateien aus der vorherigen Umsetzung eingesetzt. Da allerdings TailwindCSS in ihrer Funktionsweise eingeschränkt ist, ließ sich das mobile Menü nicht ohne zusätzliches JavaScript umsetzen, sodass eine JS-Datei erweitert werden musste (vgl. *4.3.8 Progressive Enhancement & JavaScript*). Dadurch entsteht jedoch ein Nachteil zu der ursprünglichen Herangehensweise, da auf mobilen Endgeräten nun eine grundlegende Funktion auf JS basiert. Demzufolge ist diese Umsetzung aus Sicht des Progessive Enhancements nicht optimal.

In diesem Vergleichspunkt fällt aber besonders das CSS-Framework Bootstrap negativ auf, da es nicht nur vordefinierte Sass-Dateien, sondern auch JavaScript-Dateien mitliefert. Die Funktionsweise fast der Hälfte aller Komponenten sind dadurch von JavaScript abhängig (vgl. *4.4.9 Progressive Enhancement & JavaScript*). Dementsprechend liegt es auch nicht mehr in der Entscheidungsmacht des Nutzenden, bei welchen Komponenten JS eingesetzt werden soll. Demnach werden auch in der Umsetzung mit Bootstrap grundlegende Funktionen mit JavaScript ausgeführt, wie das mobile Menü. Dies wird auch in der folgenden Tabelle reflektiert, da die genutzten JS-Dateien von Bootstrap insgesamt die längste Zeilenlänge aufweisen.

	CSS	TailwindCSS	Bootstrap
Länge JS–Code	80 Zeilen	142 Zeilen	634 Zeilen

Tabelle 5.4: Vergleich Zeilenlänge des JS-Codes, Quelle: Eigene Darstellung

Folglich schneidet aus Sicht des Progressive Enhancements die Umsetzung mit CSS am besten und die Umsetzung mit Bootstrap am schlechtesten ab. Zudem weist Bootstrap knapp 500 Codezeilen mehr auf, um die gleichen Funktionen ausführen zu können wie TailwindCSS. Dies steht nicht im Verhältnis zueinander.

5.8 CODE-QUALITÄT

Der nächste Vergleichspunkt bezieht sich auf die Code-Qualität. Dabei werden Aspekte wie redundanter Code, Struktur und Syntax und die Ergebnisse der Validatoren erörtert.

5.8.1 REDUNDANTER CODE

Als Erstes werden die Herangehensweisen auf sich wiederholende Codeabschnitte untersucht. Grundsätzlich sollte redundanter Code innerhalb einer Entwicklung vermieden werden, da nicht nur die Code-Qualität darunter leidet, sondern auch die Wartbarkeit und weitere Aspekte wie die Teamarbeit mit anderen Entwickelnden erschwert werden (vgl. *4.2.6 Code-Qualität*).

Da in der ersten Herangehensweise der Präprozessor Sass genutzt wird, können mehrfach genutzte CSS-Deklarationen in meinem `@mixin` zusammengeführt werden. Daher wurden insgesamt zehn wiederverwendbare Sammlungen an CSS-Deklarationen erstellt und redundanter Code konnte komplett ausgeschlossen werden (vgl. *4.2.6.1 Verhinderung von redundantem Code*).

Das Verhindern von redundantem Code gestaltete sich in TailwindCSS deutlich schwieriger. TailwindCSS liefert eine Funktion mit der mittels Klassenselektoren mehrfach vorkommende Utility-Ansammlungen zusammengeschlossen und zusammen angewendet werden können. Diese Funktion entspricht allerdings nicht der eigentlichen Utility-First Funktionsweise und entspricht nicht dem Sinn hinter den Utility-Klassen (vgl. *4.3.9.1 Verhinderung von redundantem Code*). Sobald diese Funktion sehr häufig und nicht mehr gezielt eingesetzt wird, verschwimmt der eigentliche Unterschied zwischen CSS und TailwindCSS. Aus diesem Grund wurden sie in der Umsetzung der individuellen Gestaltung eher vorsichtig eingesetzt, sodass nicht alle wiederholenden Utility-Klassen verhindert werden konnten (z. B. identische Listenelemente weisen weiterhin die gleichen Utility-Klassen auf). Dies könnte sich ändern, sobald ein CMS oder ein JS-Framework hinzugezogen wird (vgl. *4.3.2.4 Erstellung von Komponenten*).

Bootstrap hingegen greift mit seinen vordefinierten Komponenten bereits auf wiederverwendbare Codeabschnitte zu und ist ein gutes Beispiel dafür, wie ein Quellcode modular aufgebaut sein kann. Gleichzeitig wurden in dieser Umsetzung eigene Komponenten erstellt. Der selbst erstellte Quellcode greift zudem auf die bereits bekannte Sass-Funktion der `@mixin` zurück, sodass auch hier kein redundanter Quellcode vorkommt (vgl. *4.4.10.1 Verhinderung von redundantem Code*).

Aus diesen Erkenntnissen erschließt sich, dass nur innerhalb der Umsetzung mit TailwindCSS redundanter Code nicht verhindert werden konnte.

5.8.2 STRUKTUR & SYNTAX

Für den Erhalt einer guten Struktur und einer guten Syntax wurde in der Umsetzung mit CSS die Abhängigkeit `stylelint` verwendet, die den Quellcode auf grundlegende Regeln überprüft (vgl. *4.2.6.2 Syntax & Struktur*). Um ein übersichtliches HTML-Dokument und eine Struktur innerhalb der Klassenbezeichnungen zu erhalten, wurde zudem die Methodik BEM angewendet (vgl. Abb. 5.5).

```
<main>
  <header id="hero" class="hero">
    <h1 class="hero__headline">Veganes Café im Herzen Flensburgs</h1>
    <span class="hero__date">Est. 2012</span>
    <p class="hero__description">Entdecke unser vielfältiges Angebot an vegane Backwaren und erlebe Geschmacks&shy;explosionen
      aus reinen tierfreien, regionalen Zutaten.</p>
    <p class="hero__usp">Handarbeit, einzigartige Qualität und erstklassiger Geschmack!</p>
    <picture class="hero__image hero__image--large">
      <source sizes="83px" media="(max-width: 400px)" srcset="src/images/flavour-earth-cafe-w-83.jpg 83w">
      <source sizes="324px" media="(max-width: 700px)" srcset="src/images/flavour-earth-cafe-w-324.jpg 324w">
      <source sizes="366px" media="(min-width: 900px)" srcset="src/images/flavour-earth-cafe-w-366.jpg 366w">
      <img src="src/images/flavour-earth-cafe-w-366.jpg" alt="Flavour Earth · Unserer Innenbereich">
    </picture>
    <picture class="hero__image hero__image--small circle circle--after circle--bottom-right">
      <source sizes="150px" media="(max-width: 700px)" srcset="src/images/flavour-earth-baking-w-150.jpg 150w">
      <source sizes="500px" media="(min-width: 700px)" srcset="src/images/flavour-earth-baking-w-500.jpg 500w">
      <img src="src/images/flavour-earth-baking-w-500.jpg" alt="Flavour Earth · Wir backen nur mit den besten Zutaten">
    </picture>
    <picture class="hero__image hero__image--hidden">
      <source sizes="150px" media="(max-width: 1280px)" srcset="src/images/falvour-earth-roll-seletction-w-150.jpg 150w">
      <source sizes="250px" media="(min-width: 1280px)" srcset="src/images/falvour-earth-roll-seletction-w-250.jpg 250w">
      <img src="src/images/falvour-earth-roll-seletction-w-250.jpg" alt="Flavour Earth · Unsere Auswahl an Leckereien">
    </picture>
  </header>
```

Abb. 5.5: HTML-Struktur CSS, Quelle: Eigene Darstellung

Für TailwindCSS gibt es keine Abhängigkeiten, die die Struktur und die Syntax der Utilities überprüfen. Im Gegensatz zu der CSS-Umsetzung können auch keine Methodiken genutzt werden, sodass die Übersichtlichkeit durch die langen Verkettungen der Utility-Klassen verloren geht (vgl. *4.3.9.1 Verhinderung von redundantem Code, 4.3.9.2 Syntax & Struktur*). Dieses Problem wird in Abbildung 5.6. reflektiert. Die Zeit, die in der Entwicklung eingespart wurde, indem keine bezeichnenden Klassennamen ausgedacht werden mussten, wurde somit zur Wartung des Codes und zum Durchkämmen des HTML-Dokuments benötigt.

In der Umsetzung mit Bootstrap wurde ebenfalls auf die Abhängigkeit `stylelint` zurückgegriffen. Hier wurden jedoch explizite Struktur- und Syntax-Regeln für Bootstrap selbst angewendet. Allerdings muss gesagt werden, dass in der Umsetzung mit Bootstrap durch die fehlenden Anwendungszwecke von Komponenten vermehrt auf Utilities zurückgegriffen werden musste. Dies hat zur Folge, dass die Übersichtlichkeit des HTML-Dokuments wie in der Umsetzung mit TailwindCSS, geschmälert wurde.

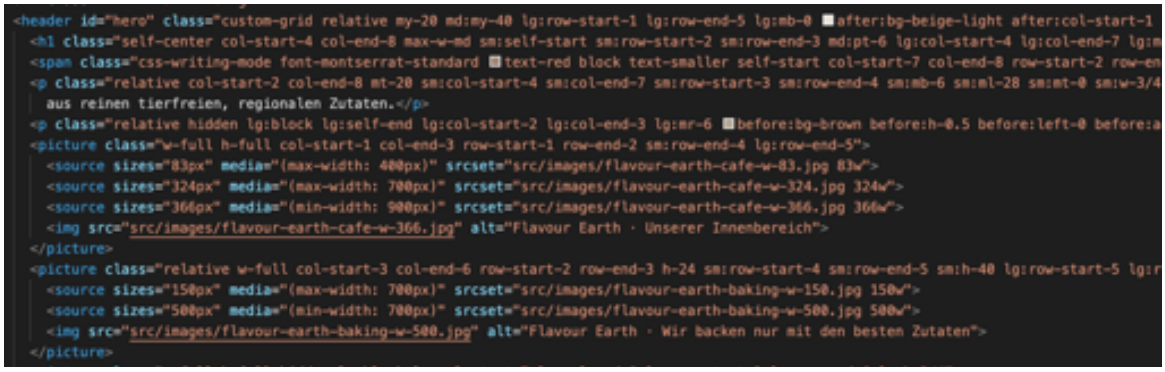


Abb. 5.6: Ausschnitt HTML-Struktur TailwindCSS, Quelle: Eigene Darstellung

Abb. 5.7: Ausschnitt HTML-Struktur Bootstrap, Quelle: Eigene Darstellung

Aus dem Vergleich von Abbildung 5.5, 5.6 und 5.7 kann somit Folgendes geschlussfolgert werden: Je mehr Utilities eingesetzt werden, desto unübersichtlicher und unstrukturierter wird das HTML-Dokument.

5.8.3 VALIDATOREN

Die zwei verwendeten Validatoren „Nu HTML Checker“ und „Jigsaw Validator“ konnten in der ersten und zweiten Herangehensweise keine Fehler innerhalb der HTML-Struktur und innerhalb des CSS-Quellcodes finden. Nur in der Umsetzung mit Bootstrap wurden von dem „Nu HTML Checker“ insgesamt zehn Fehler gefunden. Diese Fehler sind auf die von Bootstrap vorgegebene HTML-Struktur und den mitgelieferten JavaScript-Funktionen zurückzuführen (vgl. *4.4.10.3 Validatoren*).

Sobald alle Erkenntnisse aus den vorherigen Punkten zusammengeführt werden, kann geschlussfolgert werden, dass die Herangehensweise mit CSS das beste Ergebnis innerhalb der Code-Qualität erzielt. TailwindCSS dagegen kann weder alle redundanten Code-Abschnitte noch ein übersichtliches HTML-Dokument aufweisen, besitzt im Gegensatz zu Bootstrap jedoch eine valide HTML-Struktur. Dafür kann Bootstrap in Aspekten wie redundanter Code, Struktur und Syntax mit der ersten Umsetzung mithalten.

5.9 PERFORMANCE IM BROWSER

In diesem Abschnitt werden die Ergebnisse der Browsertests miteinander verglichen, sodass schlussendlich Aussagen über die Performance im Browser getroffen werden können.

5.9.1 PAGESPEED INSIGHTS

Die folgende Tabelle stellt alle Ergebnisse des „Google Pagespeed Tools“ gegenüber. Anhand dieser Tabelle kann geschlussfolgert werden, dass auf mobilen Endgeräten die CSS-Umsetzung im Vergleich zu TailwindCSS um 4% und im Vergleich zu Bootstrap um 8% schnellere Ladezeiten aufweist. Diese Unterschiede sind auf größeren Bildschirmen nicht mehr erkennen.

	Mobil	Desktop
CSS	95%	99%
TailwindCSS	91%	99%
Bootstrap	87%	99%

Tabelle 5.5: Vergleich der Ergebnisse des „Google Pagespeed Tools“, Quelle: Eigene Darstellung

5.9.2 PINGDOM

Die gemessenen Ladezeiten des „Pingdom Speed Tests“ stimmen mit den Angaben des vorherigen Browsertests überein, allerdings sind die weiteren gemessenen Werte nicht deckungsgleich mit den Ladezeiten. Obwohl die Umsetzung mit TailwindCSS das höchste Datenvolumen aufweist, hat es keine schlechtere Ladezeit als die Umsetzung mit Bootstrap, die wiederum das geringste Datenvolumen erzeugt. Weswegen diese Messwerte genau entstanden sind, konnte nicht festgestellt werden. Der folgende Abschnitt *5.9.3 Dateigrößen* liefert allerdings einen Ansatz des Grundes für die verschiedenen Ladezeiten.

	Datenmenge	Ladezeit	Abfragen
CSS	644.8 kB	282 ms	21
TailwindCSS	647.2 kB	300 ms	21
Bootstrap	644.5 kB	483 ms	21

Tabelle 5.6: Vergleich der Ergebnisse des „Pingdom Speed Tests“, Quelle: Eigene Darstellung

5.9.3 DATEIGRÖSSEN

Um einen Grund für die Ladezeiten zu finden, sollen die Dateigrößen miteinander vergleichen werden. Dafür sind die optimierten Dateigrößen ausschlaggebend, da dies die schlussendlich in der Produktion genutzten Dateien sind. Aus den Tabellen lässt sich bereits ein Zusammenhang zu den Ladezeiten erkennen, da die erste Herangehensweise mit der schnellsten Ladezeit auch die geringsten CSS- und JS-Dateigrößen aufweist. Gleichzeitig weist die langsamste Umsetzung, also die Herangehensweise mit Bootstrap, die größten CSS- und JS-Dateigrößen auf.

	styles.css nicht optimiert*	styles.css optimiert*	Differenz
CSS	4.300 Byte	3.600 Byte	700 Byte
TailwindCSS	105.000 Byte	4.800 Byte	100.200 Byte
Bootstrap	90.300 Byte	18.700 Byte	71.600 Byte

Tabelle 5.7: Vergleich der CSS-Dateigrößen, Quelle: Eigene Darstellung

*optimiert = komprimiert und mit Abhängigkeiten wie `purgecss` optimiert

	site.js nicht optimiert*	site.js optimiert*	Differenz
CSS	715 Byte	516 Byte	299 Byte
TailwindCSS	938 Byte	607 Byte	331 Byte
Bootstrap	6.100 Byte	5.000 Byte	1.100 Byte

Tabelle 5.8: Vergleich der JS-Dateigrößen, Quelle: Eigene Darstellung

Aus den Dateigrößen lassen sich nicht nur Schlussfolgerungen über die Ladezeiten im Browser ziehen, sondern auch, wie wichtig die Dateioptimierungen sind. Da in der ursprünglichen Herangehensweise der komplette Quellcode selbst erstellt und nicht automatisch generiert wird, können die Entwickelnden einen Einfluss auf die Dateigrößen nehmen. Dieser Einfluss wird den Entwickelnden bei der Nutzung eines CSS-Frameworks genommen, da viel Quellcode mitgeliefert und generiert wird, der schlussendlich oft keine Anwendung findet. Aufgrund dessen ist es in solchen Anwendungen besonders wichtig, Dateioptimierungen mithilfe von Abhängigkeiten wie `purgecss` oder `minify`, durchzuführen.

In Tabelle 5.7 ist die Einsparung des CSS-Quellcodes von TailwindCSS auffallend groß, allerdings scheint dem CSS-Framework diese Dateigröße auch bewusst zu sein, denn es liefert bereits eine eingebaute Möglichkeit, die Dateigröße mithilfe von `purgecss` zu minimieren (vgl. *4.3.10.1 Optimierungsmöglichkeiten*). Dementsprechend entsteht ein großer Vorteil gegenüber den beiden anderen Herangehensweisen, da die Entwickelnden

dazu gezwungen werden, die Vorteile dieser Abhängigkeit zu nutzen bzw. in Erwägung zu ziehen.

In Tabelle 5.8 springt ebenfalls eine Umsetzung besonders ins Auge, da die Einsparung des JS-Quellcodes besonders für die Bootstrap-Herangehensweise eine hohe Zahl generiert. Aus dieser Zahl können Rückschlüsse zu Abschnitt *5.7 Progressive Enhancement* gezogen werden, da dort bereits angesprochen wurde, dass Bootstrap nicht nur viel CSS-Quellcode, sondern auch viel JS-Quellcode mitliefert. Im Gegensatz zu TailwindCSS liefert Bootstrap allerdings keine eingebaute Möglichkeit, nicht genutzten Quellcode zu löschen. Dies kann sich zu einem großen Problem entwickeln, da die Ladezeiten und Dateigrößen von Bootstrap selbst nach der Einsparung mit Abstand die schlechtesten Werte aufweisen. Der Aspekt, dass Bootstraps Stylesheet für die gleiche Darstellung fast 14 Kilobyte größer ist als die Stylesheets der anderen Herangehensweisen, steht in keinem Verhältnis.

5.10 SCHWIERIGKEITEN BEI DER UMSETZUNG

Die Entwicklung mit einem CSS-Framework hat grundsätzlich das Ziel, kein eigenes CSS mehr schreiben zu müssen, sondern alle Designvorgaben und Inhalte nur mit den Funktionen und mitgelieferten Inhalten des CSS-Frameworks selbst umsetzen zu können. Wie bereits bekannt, mussten für die Umsetzung der individuellen Gestaltung sowohl für TailwindCSS als auch für Bootstrap auf eigenen Quellcode zurückgegriffen werden. Um zu veranschaulichen, wie viel der individuellen Gestaltung mit den CSS-Frameworks allein umgesetzt werden konnte, wurden Screenshots der Entwicklungsergebnisse der CSS-Frameworks ohne selbst erstellte Deklarationen gemacht. Diese werden der Umsetzung mit CSS entgegengestellt, sodass Schlussfolgerungen gezogen werden können, ob sich die Nutzung eines CSS-Frameworks überhaupt lohnt oder nicht.

5.10.1 HEADER

Anhand der folgenden Abbildung 5.8 soll gezeigt werden, wie sich die individuelle Anordnung innerhalb des Headers aus Abschnitt *3.2.4 Anordnung der Elemente umsetzen* ließ. Die beiden ersten Umsetzungen sind kaum auseinanderzuhalten, allerdings zeigt die dritte Umsetzung Probleme auf. Ohne selbst geschriebenen Quellcode, können in Bootstrap weder typografische Definitionen noch die Navigation oder das Menü-Icon korrekt umgesetzt werden. Dafür sind fehlende Utility-Klassen und die fehlende Möglichkeit, Komponenten zu individualisieren, die Hauptgründe (vgl. *4.4.13 Schwierigkeiten bei der Umsetzung*).

Abb. 5.8: Vergleich Header, Quelle: Eigene Darstellung

5.10.2 AN RASTER ORIENTIERTE INHALTSBLÖCKE

Abbildung 5.9 zeigt die Gegenüberstellung der Inhaltselemente, die auf das individuelle Raster angewiesen sind. Anhand dieser Gegenüberstellung lässt sich bereits erkennen, dass sich die ersten beiden Ergebnisse mit CSS und TailwindCSS sehr ähneln und kaum Unterschiede zeigen. Im Gegensatz dazu steht die rechte Abbildung, da nur in der Umsetzung mit Bootstrap das individuelle Raster ohne eigenen Quellcode nicht umgesetzt werden konnte (vgl. *4.4.13.2 Schwierigkeiten, die speziell für die Umsetzung dieser Gestaltung auftreten*).

Abb. 5.9: Vergleich Raster, Quelle: Eigene Darstellung

5.10.3 INHALTSBLÖCKE, DIE NACH DEN ETABLIERTEN WEBDESIGNSTANDARDS UMGESETZT SIND

In diesem Abschnitt soll die Umsetzbarkeit der etablierten Webdesignstandards, die in Abschnitt *3.1.2 Häufig verwendete Layouts und Raster* thematisiert wurden, untersucht werden. Dafür wurden ein dreispaltiges Raster und ein bildschirmfüllendes Bild mit zentriertem Text umgesetzt (vgl. Abb. 5.10, vgl. Abb. 5.11).

In beiden Vergleichen gerät wieder die Umsetzung mit Bootstrap in den Fokus. Bei der Umsetzung des dreispaltigen Rasters lassen sich beispielsweise die Überschriften nicht korrekt setzen oder die in der mobilen Ansicht versteckte Produktauflistung anzeigen (vgl. Abb. 5.10). Auch typografische Feinheiten lassen sich nicht umsetzen (vgl. Abb. 5.11). In Abbildung 5.11 wird ebenfalls anhand des bildschirmfüllenden Fotos veranschaulicht, dass sich die Höhe bestimmter Elemente nicht mit den mitgelieferten Utility-Werten anpassen lassen (vgl. *4.4.5.3 Utilities anpassen*).

In der Umsetzung mit TailwindCSS ließ sich lediglich das mittlere Bild der Produktauflistung nicht zuschneiden (vgl. Abb. 5.10)

Abb. 5.10: Vergleich Produktauflistung, Quelle: Eigene Darstellung

5.11: Vergleich Kontakt, Quelle: Eigene Darstellung

5.10.4 FOOTER

Anhand der Umsetzung des individuellen Footers können Schwächen in beiden CSS-Frameworks verdeutlicht werden. Sowohl mit TailwindCSS als auch mit Bootstrap lassen sich keine Hintergrundbilder in Pseudoelemente laden, sodass das Logo-Muster nicht umgesetzt werden konnte (vgl. *4.2.12. Schwierigkeiten bei der Umsetzung*). In der Umsetzung des Footers von Bootstrap spiegeln sich die zuvor genannten Probleme ebenfalls wider.

Abb. 5.12: Vergleich Footer, Quelle: Eigene Darstellung

5.11 VORKENNTNISSE & ZEITAUFWAND

Der letzte Vergleichspunkt bezieht sich auf die benötigten Vorkenntnisse, die für die einzelnen Ansätze benötigt worden sind. Im Zusammenhang dazu soll ebenfalls der Zeitaufwand gesetzt werden, der für jede Umsetzung in Anspruch genommen wurde.

Für die Entwicklung der Webseite des veganen Cafés mit handgeschriebenen CSS wurden ausgebaute Kenntnisse in HTML und CSS gefordert, da grundlegendes Wissen über die einzelnen CSS-Eigenschaften, über das Grid-System und über HTML-Semantik genutzt wurde, um die Webseite sauber umsetzen zu können (vgl. *4.2.10.1 Benötigte Vorkenntnisse*). Dadurch lag der Zeitaufwand für diese Herangehensweise bei 25 Stunden und 20 Minuten (vgl. *4.2.10.2 Benötigte Zeit zur Umsetzung*).

Auch in der Umsetzung mit TailwindCSS wurden grundlegende Vorkenntnisse über HTML und CSS benötigt, da die HTML-Struktur komplett selbstständig aufgebaut werden musste und Fachwissen über CSS-Eigenschaften benötigt wurden, um die individuelle Gestaltung umsetzen zu können (vgl. *4.3.13.1 Benötigte Vorkenntnisse*). Da sich mit TailwindCSS allerdings keine Selektorenverschachtelungen oder komplexe Sass-Funktionen abbilden lassen, muss kein Vorwissen über diese Funktionen vorhanden sein. Schlussendlich wurde für diese Umsetzung ein Zeitaufwand von 18 Stunden und 44 Minuten berechnet (vgl. *4.3.13.2 Benötigte Zeit zur Umsetzung*).

Bootstrap bietet im Gegensatz zu CSS und TailwindCSS einen großen Vorteil, da aufgrund der bereits fertig entwickelten, mitgelieferten Komponenten kein fundiertes Wissen über HTML, CSS oder JavaScript vorhanden sein

muss. Die Dokumentation liefert alle Informationen für einen Laien, sodass diese sich nur die gewünschten Inhalte heraussuchen und zusammenklicken müssen (vgl. *4.4.14 Vorkenntnisse*). Dadurch bietet Bootstrap für unerfahrene Entwickelnde den leichtesten Einstieg mit der geringsten Einstiegshürde. Der Einstieg mag im Vergleich zu den anderen Herangehensweisen zwar am leichtesten sein, jedoch ist die Lernkurve nicht so hoch. Aufgrund des fehlenden Zwangs, selbst erstellten Quellcode generieren zu müssen, wird keine Erfahrung in der Frontend-Entwicklung gesammelt. Dieser Zwang entsteht erst, wenn – wie in dieser Umsetzung – eine individuelle Gestaltung umgesetzt werden soll. In solchen Anwendungsfällen wird spezielles Vorwissen über bestimmte Eigenschaften und Funktionen von CSS benötigt, um beispielsweise Überschreibungen von Deklarationen tätigen zu können (vgl. *4.4.14.1 Benötigte Vorkenntnisse*). Dieser Zeitaufwand wird auch in der benötigten Zeit reflektiert. Insgesamt hat die Umsetzung mit Bootstrap 28 Stunden und 21 Minuten gedauert (vgl. *4.4.14.2 Benötigte Zeit zur Umsetzung*).

Dementsprechend wurde für die Umsetzung mit Bootstrap der höchste und für die Umsetzung mit TailwindCSS der geringste Zeitaufwand berechnet.

6 FAZIT

In der vorliegenden Arbeit wurde der Frage nachgegangen, welche Vor- und Nachteile verschiedene CSS-Frameworks im Vergleich zu handgeschriebenen CSS haben. Hierfür wurde das CSS-Framework TailwindCSS als Repräsentant des Utility-First-Ansatzes und Bootstrap als Komponenten basiertes CSS-Framework hinzugezogen. Diese wurden anhand der Umsetzung einer Webseite mit einer individuellen Gestaltung untersucht, sodass die drei Entwicklungsergebnisse in einem Vergleich gegenübergestellt werden konnten. Für die Gegenüberstellung wurden zudem mehrere Kriterien aus den verschiedenen Bereichen der Webentwicklung hinzugezogen.

Die Ergebnisse der Untersuchung haben gezeigt, dass mit dem herkömmlichen Ansatz, eine Webseite zu entwickeln, grundlegend weniger Einschränkungen vorhanden sind. Dies spiegelt sich nicht nur in der Nutzung aller CSS-Eigenschaften und -Funktionen wider, sondern auch in der Entscheidungsfreiheit über Aspekte wie dem Tooling-Aufwand, passende Methodiken oder genutzte JavaScript Funktionen. Für jede neue Umsetzung können somit alle Bausteine und Puzzleteile einer Webseitenentwicklung neu und individuell zusammengewürfelt werden, sodass es den perfekten Nutzen reflektiert und alle Bedürfnisse des Entwickelnden und potenziellen Kunden abdeckt. Auch in der Umsetzung der Webseite für das fiktive vegane Café konnten alle Wünsche und Designangaben aus dem Screendesign sauber und kompakt umgesetzt werden. Dadurch wies nicht nur die Code-Qualität, sondern auch die Dateigröße und die Ladezeit im Browser bessere Werte auf als die Ergebnisse mittels der CSS-Frameworks.

Da in der Herangehensweise mit handgeschriebenen CSS jedoch viele Aspekte wie ein Gestaltungssystem oder eine responsive Umsetzungsstrategie nicht vordefiniert und mitgeliefert werden, sondern von dem Entwickelnden selbst eingebaut, entwickelt oder entschieden werden müssen, entsteht eine hohe Einstiegshürde für Neueinsteiger in der Frontend-Entwicklung. Zudem wird viel Vorwissen und Erfahrung vorausgesetzt, um beispielsweise die bestmögliche Umsetzungsmöglichkeit anwenden zu können oder zu wissen, wie am besten mit den unterschiedlichen Gestaltungsvorgaben der Browser umgegangen werden kann. Diese Entwicklungsentscheidungen werden bei der Nutzung eines CSS-Frameworks abgenommen, sodass dadurch generell die Einstiegshürde gesenkt wird.

Allerdings hat diese Arbeit auch gezeigt, dass die beiden genutzten CSS-Frameworks große Unterschiede im Vergleich zueinander aufweisen. TailwindCSS liefert nicht nur das besser durchdachte und einfacher anzuwendende Gestaltungssystem, sondern ermöglicht durch die

anpassungsfähige Konfigurationsdatei eine Flexibilität, die von Bootstrap nicht ansatzweise abgebildet werden konnte. Dies wird auch in den Entwicklungsergebnissen reflektiert, da mit Bootstrap – ohne Hilfestellung durch selbst erstellte Sass-Dateien – kaum Gestaltungsvorgaben aus dem individuellen Screendesign umgesetzt werden konnten.

Anhand der Umsetzung mit dem Komponenten basierten CSS-Framework taten sich jedoch noch weitere negative Aspekte auf. Allein der Vorgang, eigene Gestaltungsparameter definieren zu können, erwies sich deutlich komplizierter als in den anderen Herangehensweisen und zeigte viele Fallstricke. Obwohl die Nutzung von Bootstraps Komponenten Entwicklungszeit einsparen sollte, hat deren Anpassung und Individualisierung deutlich mehr Zeit und Nerven gekostet als die beiden anderen Ansätze. Gleichzeitig ist das CSS-Framework überladen mit mitgelieferten Dateien und Inhalten, die komplett irrelevant waren für dieses Anwendungsbeispiel. Dieser Aspekt spitzt sich in der Browserperformance zu, da dieses Webseitenergebnis nicht nur die langsamste Ladezeit, sondern ebenfalls mit Abstand die größte zu verarbeitender Datenmenge aufweist.

Obwohl Bootstrap auch Vorteile gegenüber der Umsetzung mit TailwindCSS aufzuweisen hat, wie eine bessere Code-Qualität, ein übersichtlicheres HTML-Dokument und ein besseres Ergebnis im Cross-Browser-Rendering, ist die Funktions- und Arbeitsweise mit diesem CSS-Framework deutlich komplizierter und erzielt ein deutlich schlechteres Ergebnis.

Dadurch geht auch der eigentliche Vorteil von CSS-Frameworks gegenüber der herkömmlichen Herangehensweise einer Webseitenentwicklung verloren, da die mitgelieferten Inhalte und Funktionen von Bootstrap, die den Nutzenden eigentlich eine Erleichterung bieten sollten, zu nicht tragbaren Einschränkungen führen.

Letztendlich wurde durch die praktische Untersuchung gezeigt, dass sich mithilfe von CSS-Frameworks, insbesondere das Komponenten basierte CSS-Framework Bootstrap keine besseren Ergebnisse in Qualität, Dateigröße und Performance im Browser erzielen lassen. Es konnte allerdings festgestellt werden, dass das CSS-Framework TailwindCSS aufgrund des gut durchdachten Gestaltungssystems, die eingebaute Funktion zur Nutzung von Mobile First und das Löschen nicht genutzter Inhalte eine Erleichterung innerhalb der Entwicklung bringen kann, ohne dabei zu große Einschnitte in der Performance zu erhalten. Dementsprechend kann TailwindCSS die richtige Herangehensweise für Entwickelnde sein, die sich nicht jedes Mal hinsichtlich passender Abhängigkeiten und Methodiken neu orientieren, sondern auf ein funktionierendes System zurückgreifen wollen, welches mittels ihrer Utilities nicht alle, aber viele individuelle Gestaltungsvorgaben umsetzbar macht.

Der Vergleich der verschiedenen Herangehensweisen hat deutlich gemacht, dass Vor- und Nachteile der untersuchten CSS-Frameworks nicht für alle CSS-Frameworks verallgemeinert werden können. Sowohl TailwindCSS als auch Bootstrap verfolgen unterschiedliche Ansätze und weisen daher individuelle Vor- und Nachteile auf. Deutlich wurde, dass die Umsetzung mit TailwindCSS im Gegensatz zu Bootstrap einfacher war und ein besseres Endergebnis erzielt werden konnte. Wenn man jedoch die Umsetzung einer individuell gestalteten Webseite betrachtet, kann zusammenfassend gesagt werden, dass mithilfe von handgeschriebenen CSS individuelle, explizite Inhalte und Designvorgaben leichter umgesetzt werden konnten. Der Grund dafür liegt in der hohen Einflussnahme des Entwickelnden.

7 VERZEICHNISSE

7.1 ABKÜRZUNGSVERZEICHNIS

HTML	Hypertext Markup Language
W3C	World Wide Web Consortium
WHATWG	Web Hypertext Application Technology Working Group
CSS	Cascading Style Sheets
JS	JavaScript
DOM	Document Object Model
UI	User Interface
WCAG	Web Content Accessibility Guidelines
Sass	Syntactically awesome style sheets
BEM	Block-Element-Modifier
CMS	Content-Management-System
CLI	Command-Line-Interface
CDN	Content-Delivery-Network
IE	Internet Explorer (Browser)

7.2 ABBILDUNGSVERZEICHNIS

7.3 TABELLENVERZECIHNIS

7.4 LITERATURVERZEICHNIS

Barker, Tom: High Performance Responsive Design: Building Faster Sites Across Devices, Sebastopol, USA: O’Reilly Media, 2014.

Hahn, Martin: Webdesign: Das neue Handbuch zur Webgestaltung. Alles, was Webdesigner wissen müssen. Mit vielen inspirierenden Beispielen, 3. Aufl., Bonn, Deutschland: Rheinwerk Design, 2020.

Laborenz, Kai: CSS: Das umfassende Handbuch. Inkl. Responsive Webdesign, Animationen, Sass, 3. Aufl., Bonn, Deutschland: Rheinwerk Computing, 2015.

Lanciaux, Ryan: Modern Front-end Architecture: Optimize Your Front-end Development with Components, Storybook, and Mise en Place Philosophy, 1st ed., New York, USA: Apress, 2021.

Meiert, Jens Oliver: The Little Book of HTML/CSS Coding Guidelines, 2015, https://www.oreilly.com/library/view/the-little-book/9781492048459/.

Rappin, Noel: Modern CSS with Tailwind: Flexible Styling Without the Fuss, Raleigh, USA: Pragmatic Bookshelf, 2021.

Robbins, Jennifer Niederst: Learning Web Design: A Beginner’s Guide to HTML, CSS, JavaScript, and Web Graphics, Sebastopol, USA: O’Reilly, 2018.

Robbins, Jennifer Niederst: Webdesign mit (X)HTML und CSS: Das Praxisbuch zum Einsteigen, Auffrischen und Vertiefen, 1., Köln, Deutschland: O’Reilly Verlag GmbH & Co. KG, 2008.

Shenoy, Aravind/Anirudh Prabhu: CSS Framework Alternatives: Explore Five Lightweight Alternatives to Bootstrap and Foundation with Project Examples, 2018, doi:10.1007/978-1-4842-3399-3.

Wäger, Markus: ABC des Grafikdesigns: Grafik und Gestaltung visuell erklärt, 1. Aufl., Bonn, Deutschland: Rheinwerk Design, 2020.

7.5 ONLINE-QUELLEN

Adobe Fonts: Adobe Caslon from Adobe Originals, in: Adobe Fonts, o. D., https://fonts.adobe.com/fonts/adobe-caslon#licensing-section (abgerufen am 12.08.2021a).

Adobe Fonts: IvyPresto Display from Ivy Foundry, in: Adobe Fonts, o. D., https://fonts.adobe.com/fonts/ivypresto-display#licensing-section (abgerufen am 12.08.2021b).

Adobe Fonts: Montserrat from Google, in: Adobe Fonts, o. D., https://fonts.adobe.com/fonts/montserrat#licensing-section (abgerufen am 19.08.2021c).

Badach, Anatol: Content Delivery Network, in: researchgate.net, o. D., https://www.researchgate.net/profile/Anatol-Badach/publication/282008087_CDN_-_Content_Delivery_Network/links/560147d708ae07629e52bf24/CDN-Content-Delivery-Network.pdf (abgerufen am 24.08.2021).

Bjankord: GitHub - bjankord/stylelint-config-sass-guidelines: A stylelint config inspired by https://sass-guidelin.es/, in: GitHub, o. D., https://github.com/bjankord/stylelint-config-sass-guidelines (abgerufen am 15.08.2021a).

Blueprint: Blueprint: A CSS Framework | Spend your time innovating, not replicating, in: Blueprint, o. D., http://blueprintcss.org/ (abgerufen am 07.08.2021).

BMW: BMW Deutschland, in: BMW, o. D., https://www.bmw.de/de/home.html (abgerufen am 10.08.2021).

CERN: Tags used in HTML, in: CERN, o. D., http://info.cern.ch/hypertext/WWW/MarkUp/Tags.html (abgerufen am 13.08.2021).

Chrome Developers: Inspect network activity, in: Chrome Developers, o. D., https://developer.chrome.com/docs/devtools/network/ (abgerufen am 27.08.2021).

Dudenredaktion: Best Practice, in: Duden, o. D., https://www.duden.de/rechtschreibung/Best_Practice (abgerufen am 14.08.2021a).

Dudenredaktion: Content Management, in: Duden, o. D., https://www.duden.de/rechtschreibung/Content_Management (abgerufen am 23.08.2021b).

Dudenredaktion: Debugging, in: Duden, o. D., https://www.duden.de/rechtschreibung/Debugging (abgerufen am 13.08.2021c).

Dudenredaktion: Open Source Software, in: Duden, o. D., https://www.duden.de/rechtschreibung/Open_Source_Software (abgerufen am 13.08.2021d).

Dudenredaktion: Responsiv, in: Duden, o. D., https://www.duden.de/rechtschreibung/responsiv (abgerufen am 25.09.2021e).

Earth, Flavour: Flavour Earth Ein Veganes Cafe im Herzen Flensburgs, in: Flavour Earth, o. D., https://groenwald-thesis-bootstrap.netlify.app/ (abgerufen am 10.09.2021a).

Earth, Flavour: Flavour Earth Ein Veganes Cafe im Herzen Flensburgs, in: Flavour Earth, o. D., https://groenwald-thesis-tailwindcss.netlify.app/ (abgerufen am 10.09.2021b).

Earth, Flavour: Flavour Earth Ein Veganes Cafe im Herzen Flensburgs, in: Flavour Earth, o. D., https://6139f2042e67d70007823815--groenwald-thesis-bootstrap.netlify.app/ (abgerufen am 10.09.2021c).

Earth, Flavour: Flavour Earth Ein Veganes Cafe im Herzen Flensburgs, in: Flavour Earth, o. D., https://613ca71473ad54659fc6b665--groenwald-thesis-bootstrap.netlify.app/ (abgerufen am 11.09.2021d).

Earth, Flavour: Flavour Earth Ein Veganes Cafe im Herzen Flensburgs, in: Flavour Earth, o. D., https://613ca658433efd0007ba9b82--groenwald-thesis-bootstrap.netlify.app/ (abgerufen am 11.09.2021e).

Earth, Flavour: Flavour Earth Ein Veganes Cafe im Herzen Flensburgs, in: Flavour Earth, o. D., https://612869205069b80007939083--groenwald-thesis-css.netlify.app/ (abgerufen am 27.09.2021f).

Earth, Flavour: Flavour Earth Ein Veganes Cafe im Herzen Flensburgs, in: Flavour Earth, o. D., https://6141eca2a896ee0008de70e5--groenwald-thesis-css.netlify.app/ (abgerufen am 27.09.2021g).

Earth, Flavour: Flavour Earth Ein Veganes Cafe im Herzen Flensburgs, in: Flavor Earth, o. D., https://groenwald-thesis-css.netlify.app/ (abgerufen am 11.09.2021h).

Earth, Flavour: Flavour Earth Ein Veganes Cafe im Herzen Flensburgs, in: Flavour Earth, o. D., https://6145c1a3b509460007050e87--groenwald-thesis-tailwindcss.netlify.app/ (abgerufen am 30.09.2021i).

Earth, Flavour: Flavour Earth Veganes Cafe im Herzen Flensburgs, in: Flavour Earth, o. D., https://612f7151beeeb90007a076ca--groenwald-thesis-tailwindcss.netlify.app/ (abgerufen am 01.09.2021j).

eightshapes: EightShapes Contrast Grid, in: eightshapes, o. D., https://contrast-grid.eightshapes.com/?background-colors=&foreground-colors=%23faf9f7%2C%20Extra-Light%0D%0A%23e7e4db%2C%20Light%0D%0A%23b3a898%2C%20Brown%0D%0A%239D887A%2C%20Red%0D%0A%238E8B7C%2C%20Green%0D%0A%235f6053%2C%20Dark%0D%0A%0D%0A&es-color-form__tile-size=compact (abgerufen am 12.08.2021).

Fonts.com: Adobe Caslon Font Family Typeface Story, in: Fonts.com, o. D., https://www.fonts.com/de/font/adobe/adobe-caslon/story (abgerufen am 12.08.2021).

Giraudel, Kitty: Sass Guidelines, in: Sass Guidelines, o. D., https://sass-guidelin.es/ (abgerufen am 26.08.2021).

Marcotte, Ethan/Mike Wills/Eva PenzeyMoog/Tom Greenwood/Preston So/Cathy Dutton: Responsive Web Design, in: A List Apart, 01.05.2019, https://alistapart.com/article/responsive-web-design/ (abgerufen am 14.08.2021).

MDN Web Docs: - HTML: HyperText Markup Language | MDN, in: MDN Web Docs, 09.09.2021a, https://developer.mozilla.org/de/docs/Web/HTML/Element/div (abgerufen am 10.09.2021).

MDN Web Docs: Background-image - CSS | MDN, in: MDN Web Docs, 01.09.2021b, https://developer.mozilla.org/de/docs/Web/CSS/background-image (abgerufen am 01.09.2021).

MDN Web Docs: Clip-path - CSS | MDN, in: MDN Web Docs, 01.09.2021c, https://developer.mozilla.org/de/docs/Web/CSS/clip-path (abgerufen am 01.09.2021).

MDN Web Docs: Cross browser testing - Learn web development | MDN, in: MDN Web Docs, 27.04.2021d, https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Cross_browser_testing (abgerufen am 16.08.2021).

MDN Web Docs: Flexbox - Lerne Webentwicklung | MDN, in: MDN Web Docs, 01.08.2021e, https://developer.mozilla.org/de/docs/Learn/CSS/CSS_layout/Flexbox (abgerufen am 01.08.2021).

MDN Web Docs: Herstellerpräfix - Glossar | MDN, in: MDN Web Docs, 29.08.2021f, https://developer.mozilla.org/de/docs/Glossary/Vendor_Prefix (abgerufen am 30.08.2021).

MDN Web Docs: HTML: HyperText Markup Language | MDN, in: MDN Web Docs, 28.07.2021g, https://developer.mozilla.org/de/docs/Web/HTML (abgerufen am 06.08.2021).

MDN Web Docs: Introducing the CSS Cascade - CSS: Cascading Style Sheets | MDN, in: MDN Web Docs, 11.06.2021h, https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade#origin_of_css_declarations (abgerufen am 07.08.2021).

MDN Web Docs: Introducing the CSS Cascade - CSS: Cascading Style Sheets | MDN, in: MDN Web Docs, 13.08.2021i, https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade (abgerufen am 13.08.2021).

MDN Web Docs: Mobile first - Progressive web apps (PWAs) | MDN, in: MDN Web Docs, 01.06.2021j, https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Responsive/Mobile_first (abgerufen am 14.08.2021).

MDN Web Docs: Responsive images - Learn web development | MDN, in: MDN Web Docs, 08.07.2021k, https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Responsive_images#resolution_switching_same_size_different_resolutions (abgerufen am 16.08.2021).

MDN Web Docs: Spezifität - CSS | MDN, in: MDN Web Docs, 30.07.2021l, https://developer.mozilla.org/de/docs/Web/CSS/Specificity#die_!important_ausnahme (abgerufen am 06.08.2021).

MDN Web Docs: Using media queries - CSS: Cascading Style Sheets | MDN, in: MDN Web Docs, 13.08.2021m, https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries (abgerufen am 14.08.2021).

MDN Web Docs: Viewport - MDN Web Docs Glossary: Definitions of Web-related terms | MDN, in: MDN Web Docs, 05.11.2020, https://developer.mozilla.org/en-US/docs/Glossary/Viewport (abgerufen am 14.08.2021).

MDN Web Docs: What is CSS? - Learn web development | MDN, in: MDN Web Docs, 02.07.2021n, https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS#what_is_css_for (abgerufen am 06.08.2021).

MDN Web Docs: What is CSS? - Learn web development | MDN, in: MDN Web Docs, 02.07.2021o, https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS (abgerufen am 25.08.2021).

MDN Web Docs: What is CSS? - Learn web development | MDN, in: MDN Web Docs, 02.07.2021p, https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS#css_syntax (abgerufen am 07.08.2021).

mehr als lernen: Mehr als lernen e.V. - Gemeinnützige Bildungsinitiative, in: mehr als lernen e.V., 01.09.2021, https://mehralslernen.org/ (abgerufen am 10.08.2021).

@mrmrs: TACHYONS - Css Toolkit, in: Tachyons, o. D., https://tachyons.io/ (abgerufen am 10.08.2021).

Netlify: Netlify: Develop & deploy the best web experiences in record time, in: Netlify, o. D., https://www.netlify.com/ (abgerufen am 19.08.2021).

npmjs: Npm - a JavaScript package manager, in: npmjs, 12.08.2021a, https://www.npmjs.com/package/npm (abgerufen am 14.08.2021).

npmjs: Npm: babel-loader, in: npmjs, 26.11.2020a, https://www.npmjs.com/package/babel-loader (abgerufen am 09.09.2021).

npmjs: Npm: bootstrap, in: npmjs, 04.08.2021b, https://www.npmjs.com/package/bootstrap (abgerufen am 05.09.2021).

npmjs: Npm: minify, in: npmjs, 04.05.2021c, https://www.npmjs.com/package/minify (abgerufen am 15.08.2021).

npmjs: Npm: nodemon, in: npmjs, 10.07.2021d, https://www.npmjs.com/package/nodemon (abgerufen am 15.08.2021).

npmjs: Npm: node-sass, in: npmjs, 24.06.2021e, https://www.npmjs.com/package/node-sass (abgerufen am 26.08.2021).

npmjs: Npm: normalize-sass, in: npmjs, 30.08.2014, https://www.npmjs.com/package/normalize-sass (abgerufen am 22.09.2021).

npmjs: Npm: npm-run-all, in: npmjs, 24.11.2018, https://www.npmjs.com/package/npm-run-all (abgerufen am 15.08.2021).

npmjs: Npm: postcss, in: npmjs, 21.07.2021f, https://www.npmjs.com/package/postcss (abgerufen am 26.08.2021).

npmjs: Npm: postcss, in: npmjs, 21.07.2021g, https://www.npmjs.com/package/postcss (abgerufen am 26.08.2021).

npmjs: Npm: postcss-cli, in: npmjs, 12.12.2020b, https://www.npmjs.com/package/postcss-cli (abgerufen am 15.08.2021).

npmjs: Npm: serve, in: npmjs, 08.06.2021h, https://www.npmjs.com/package/serve (abgerufen am 15.08.2021).

npmjs: Npm: stylelint-config-standard, in: npmjs, 24.04.2021i, https://www.npmjs.com/package/stylelint-config-standard (abgerufen am 15.08.2021).

npmjs: Npm: stylelint-config-twbs-bootstrap, in: npmjs, 19.07.2021j, https://www.npmjs.com/package/stylelint-config-twbs-bootstrap (abgerufen am 05.09.2021).

npmjs: Npm: tailwindcss, in: npmjs, 30.08.2021k, https://www.npmjs.com/package/tailwindcss (abgerufen am 05.09.2021).

npmjs: Npm: webpack, in: npmjs, 04.08.2021l, https://www.npmjs.com/package/webpack (abgerufen am 05.09.2021).

npmjs: Npm: webpack-cli, in: npmjs, 15.08.2021m, https://www.npmjs.com/package/webpack-cli (abgerufen am 09.09.2021).

npmjs: Scripts | npm Docs, in: npmjs, o. D., https://docs.npmjs.com/cli/v7/using-npm/scripts/ (abgerufen am 15.08.2021).

npmjs: Specifying dependencies and devDependencies in a package.json file | npm Docs, in: npmjs, 30.08.2021n, https://docs.npmjs.com/specifying-dependencies-and-devdependencies-in-a-package-json-file/ (abgerufen am 22.09.2021).

Nu HTML Checker: Ready to check - Nu Html Checker, in: Nu HTML Checker, o. D., https://validator.w3.org/nu/ (abgerufen am 16.08.2021).

Otto, Mark/Jacob Thornton: Bootstrap, in: GetBootstrap, o. D., https://getbootstrap.com/ (abgerufen am 04.09.2021a).

Otto, Mark/Jacob Thornton: Breakpoints, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.1/layout/breakpoints/ (abgerufen am 04.09.2021b).

Otto, Mark/Jacob Thornton: Browsers and devices, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.0/getting-started/browsers-devices/#internet-explorer (abgerufen am 08.09.2021c).

Otto, Mark/Jacob Thornton: Browsers and devices, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.0/getting-started/browsers-devices/ (abgerufen am 08.09.2021d).

Otto, Mark/Jacob Thornton: Buttons, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.0/components/buttons/ (abgerufen am 08.09.2021e).

Otto, Mark/Jacob Thornton: Components, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.1/customize/components/ (abgerufen am 04.09.2021f).

Otto, Mark/Jacob Thornton: Contents, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.0/getting-started/contents/ (abgerufen am 05.09.2021g).

Otto, Mark/Jacob Thornton: CSS Grid, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.1/layout/css-grid/ (abgerufen am 12.09.2021h).

Otto, Mark/Jacob Thornton: Customize, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.1/customize/overview/#overview (abgerufen am 04.09.2021i).

Otto, Mark/Jacob Thornton: Download, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.1/getting-started/download/#source-files (abgerufen am 04.09.2021j).

Otto, Mark/Jacob Thornton: Introduction, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.0/getting-started/introduction/#js (abgerufen am 09.09.2021k).

Otto, Mark/Jacob Thornton: Introduction, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.0/getting-started/introduction/#js (abgerufen am 08.09.2021l).

Otto, Mark/Jacob Thornton: Optimize, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.1/customize/optimize/#unused-css (abgerufen am 10.09.2021m).

Otto, Mark/Jacob Thornton: Options, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.0/customize/options/ (abgerufen am 08.09.2021n).

Otto, Mark/Jacob Thornton: Sass, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.1/customize/sass/#variable-defaults (abgerufen am 06.09.2021o).

Otto, Mark/Jacob Thornton: Sizing, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.0/utilities/sizing/ (abgerufen am 23.09.2021p).

Otto, Mark/Jacob Thornton: Spacing, in: GeBootstrap, o. D., https://getbootstrap.com/docs/5.1/utilities/spacing/#maps (abgerufen am 06.09.2021q).

Otto, Mark/Jacob Thornton: Utilities for layout, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.1/layout/utilities/ (abgerufen am 04.09.2021r).

Otto, Mark/Jacob Thornton: Utility API, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.0/utilities/api/ (abgerufen am 23.09.2021s).

Otto, Mark/Jacob Thornton: Webpack and bundlers, in: GetBootstrap, o. D., https://getbootstrap.com/docs/5.1/getting-started/webpack/ (abgerufen am 10.09.2021t).

Otto, Mark/Jacob Thornton Thornton: Cards, in: GeBootstrap, o. D., https://getbootstrap.com/docs/5.1/components/card/#header-and-footer (abgerufen am 10.09.2021u).

PageSpeed Insights: PageSpeed Insights, in: PageSpeed Insights, o. D., https://developers.google.com/speed/pagespeed/insights/?hl=DE (abgerufen am 27.09.2021).

PayPal: Bargeldloses Bezahlen - Online Shopping | PayPal DE, in: PayPal, o. D., https://www.paypal.com/de/home (abgerufen am 14.08.2021).

Pingdom: Pingdom Tools, in: Pingdom, o. D., https://tools.pingdom.com/ (abgerufen am 25.09.2021a).

Pingdom: Pingdom Tools, in: Pingdom, o. D., https://tools.pingdom.com/#5f06f2159e000000 (abgerufen am 27.09.2021b).

Pingdom: Pingdom Tools, in: Pingdom, o. D., https://tools.pingdom.com/#5f06f275d5400000 (abgerufen am 25.09.2021c).

Pingdom: Pingdom Tools, in: Pingdom, o. D., https://tools.pingdom.com/#5f06f1fa0cc00000 (abgerufen am 25.09.2021d).

postcss: PostCSS - a tool for transforming CSS with JavaScript, in: postcss, o. D., https://postcss.org/ (abgerufen am 15.08.2021).

SELFHTML: Spezifikation – SELFHTML-Wiki, in: SELFHTML, o. D., https://wiki.selfhtml.org/wiki/Spezifikation (abgerufen am 07.08.2021).

Sindresorhus: GitHub - sindresorhus/modern-normalize: 😎 Normalize browsers' default style, in: GitHub, o. D., https://github.com/sindresorhus/modern-normalize#browser-support (abgerufen am 30.08.2021b).

Stadtwerke Flensburg: Stadtwerke Flensburg, in: Stadtwerke Flensburg, o. D., https://www.stadtwerke-flensburg.de/#/acquisition (abgerufen am 01.08.2021).

State of CSS: The State of CSS 2020: CSS Methodologies, in: State of CSS, o. D., https://2020.stateofcss.com/en-us/technologies/methodologies/ (abgerufen am 15.08.2021a).

State of CSS: The State of CSS 2020: Demographics, in: State of CSS, o. D., https://2020.stateofcss.com/en-us/demographics/ (abgerufen am 06.08.2021b).

State of CSS: The State of CSS 2020: Pre-/Post-processors, in: State of CSS, o. D., https://2020.stateofcss.com/en-us/technologies/pre-post-processors/ (abgerufen am 17.08.2021c).

Statista: Anzahl veganer Gastronomiebetriebe in Deutschland bis 2021, in: Statista, 01.07.2021a, https://de.statista.com/statistik/daten/studie/381076/umfrage/anzahl-veganer-gastronomiebetriebe-in-deutschland/ (abgerufen am 07.08.2021).

Statista: Meistgenutzte Browser in Deutschland August 2021, in: Statista, 07.09.2021b, https://de.statista.com/statistik/daten/studie/828151/umfrage/meistgenutzte-browserfamilien-im-internet-in-deutschland/ (abgerufen am 18.09.2021).

Statista: Umfrage in Deutschland zur Anzahl der Veganer bis 2020, in: Statista, 16.07.2021c, https://de.statista.com/statistik/daten/studie/445155/umfrage/umfrage-in-deutschland-zur-anzahl-der-veganer/ (abgerufen am 07.08.2021).

Strukchinsky, Vsevolod Vladimir Starkov And Contributors: BEM — Block Element Modifier, in: BEM, o. D., http://getbem.com/introduction/ (abgerufen am 15.08.2021).

Stylelint: Home | Stylelint, in: Stylelint, o. D., https://stylelint.io/ (abgerufen am 15.08.2021).

TailwindCSS: Breakpoints - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/breakpoints (abgerufen am 16.09.2021a).

TailwindCSS: Browser Support - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/browser-support (abgerufen am 30.08.2021b).

TailwindCSS: Configuration - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/configuration (abgerufen am 25.08.2021c).

TailwindCSS: Configuration - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/configuration#creating-your-configuration-file (abgerufen am 25.08.2021d).

TailwindCSS: Customizing Spacing - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/customizing-spacing#default-spacing-scale (abgerufen am 03.09.2021e).

TailwindCSS: Editor Support - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/editor-support#intelli-sense-for-vs-code (abgerufen am 31.08.2021f).

TailwindCSS: Extracting Components - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/extracting-components#extracting-template-components (abgerufen am 24.08.2021g).

TailwindCSS: Extracting Components - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/extracting-components#extracting-component-classes-with-apply (abgerufen am 24.08.2021h).

TailwindCSS: Functions & Directives - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/functions-and-directives#tailwind (abgerufen am 24.08.2021i).

TailwindCSS: Hover, Focus & Other States - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/hover-focus-and-other-states (abgerufen am 16.09.2021j).

TailwindCSS: Installation - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/installation#using-tailwind-cli (abgerufen am 24.08.2021k).

TailwindCSS: Just-in-Time Mode - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/just-in-time-mode#pseudo-element-variants (abgerufen am 23.08.2021l).

TailwindCSS: Preflight - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/preflight (abgerufen am 30.08.2021m).

TailwindCSS: Responsive Design - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/responsive-design#overview (abgerufen am 19.08.2021n).

TailwindCSS: Responsive Design - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/responsive-design#overview (abgerufen am 19.08.2021o).

TailwindCSS: Tailwind CSS - Rapidly build modern websites without ever leaving your HTML., in: TailwindCSS, 15.11.2020, https://tailwindcss.com/ (abgerufen am 19.08.2021).

TailwindCSS: Utility-First - Tailwind CSS, in: TailwindCSS, o. D., https://tailwindcss.com/docs/utility-first (abgerufen am 06.08.2021p).

tailwindlabs: How to add Custom Fonts to a Website using TailwindCss? · Discussion #2060 · tailwindlabs/tailwindcss, in: GitHub, o. D., https://github.com/tailwindlabs/tailwindcss/discussions/2060 (abgerufen am 01.09.2021a).

tailwindlabs: Release v0.1.0 · tailwindlabs/tailwindcss, in: GitHub, o. D., https://github.com/tailwindlabs/tailwindcss/releases/tag/v0.1.0 (abgerufen am 19.08.2021b).

TMetric: Free Time Tracking Software & App - TMetric, in: TMetric, o. D., https://tmetric.com/ (abgerufen am 23.08.2021).

Twbs: Build software better, together, in: GitHub, o. D., https://github.com/twbs/stylelint-config-twbs-bootstrap/blob/main/css/index.html (abgerufen am 10.09.2021c).

Twbs: Issue with variable overrides · Issue #43 · twbs/bootstrap-npm-starter, in: GitHub, o. D., https://github.com/twbs/bootstrap-npm-starter/issues/43 (abgerufen am 08.09.2021d).

W3C: About the Nu Html Checker, in: Nu HTML Checker, o. D., https://validator.w3.org/nu/about.html (abgerufen am 16.08.2021a).

W3C: About W3C, in: World Wide Web Consortium (W3C), o. D., https://www.w3.org/Consortium/ (abgerufen am 06.08.2021b).

W3C: HTML 4.0 Specification, in: World Wide Web Consortium (W3C), o. D., https://www.w3.org/TR/1998/REC-html40-19980424/ (abgerufen am 07.08.2021c).

WHATWG: FAQ — WHATWG, in: WHATWG, o. D., https://whatwg.org/faq#living-standard (abgerufen am 13.08.2021a).

WHATWG: HTML Standard, Edition for Web Developers, in: WHATWG, o. D., https://html.spec.whatwg.org/dev/introduction.html#history-2 (abgerufen am 13.08.2021b).

Wium Lie, Håkon: Cascading HTML Style Sheets -- A Proposal, in: World Wide Web Consortium W3C, o. D., https://www.w3.org/People/howcome/p/cascade.html (abgerufen am 07.08.2021).

8 ANHANG

8.1 GESTALTUNG

UNSERE GESCHICHTE

Das Abenteuer begann bei unserer unbegrenzten Liebe zum Backen und beim Ausprobieren neuer Kreationen. Nachdem wir - Alma und Julia - unsere Ausbildungen abgeschlossen hatten, haben wir beide uns beide spontan dazu entschieden, unsere erste gemeinsame Bäckerei zu eröffnen. Da wir uns seit mehr als 5 Jahren vegan ernähren, sahen wir dies als unsere Chance mit unseren leckeren Rezepten zu überzeugen! Somit ist 2012 unsere vegane Café „Flavour Earth" entstanden. Seitdem entdecken wir jeden Tag neue Rezepte und Zutaten, die unsere Produkte verfeinern.

Seit 2012 bieten wir neben Brot, Brötchen und belegte Sandwiches auch Heiß- und Kaltgetränke an. Von Kaffee Latte bis Erdbeer-Bananen-Kiwi-Smoothie gibt es alles was das Herz begehrt!

Wer wie Alma es eher etwas Süßer mag, wird unsere Auswahl an saisonal-passende Kuchen, Muffins, Croissants und andere Leckereien lieben!

ENTDECKE UNSERE VEGANE VIELFALT

BROT & BRÖTCHEN

BROT

Bauernbrot

Dinkel-Vollkorn Brot

Dinkel-Sojajogurt-Brot

Gewürzbrot mit
Fenchel & Koriander

Glutenfreies Brot

ALLES SÜßE

KUCHEN

Schokokuchen mit Blaubeeren

Mohnkuchen

Pflaumen-Hefekuchen

Niederländischer Apfelkuchen

DURSTLÖSCHER

HEISSETRÄNKE

Espresso

Cappuccino

Latte Machiato

Mocha Latte

Mocha Kakao

Americano

BROT & BRÖTCHEN

BROT

Bauernbrot
Dinkel-Vollkorn Brot
Dinkel-Sojajogurt-Brot
Gewürzbrot mit
Fenchel & Koriander
Glutenfreies Brot
Haferbrot
Sauerteigbrot

BRÖTCHEN

Bauernbrot
Dinkel-Vollkorn Brötchen
Glutenfreie Brötchen
Roggenbrötchen
Laugenbrötchen
Proteinbrötchen
Schokobrötchen
Quarkbrötchen

ALLES SÜßE

KUCHEN

Schokokuchen mit Blaubeeren
Mohnkuchen
Pflaumen-Hefekuchen
Niederländischer Apfelkuchen
Zitronenkuchen

GEBÄCK & MUFFINS

Hefeknoten mit Zimt-Nuss
Franzbrötchen
Apfel-Streuseltaler
Vanille-Müsli-Kekse
Muffins
mit Blaubeeren
mit Erdbeeren & Schokolade
mit Rhabarber

DURSTLÖSCHER

HEIßGETRÄNKE

Espresso
Cappuccino
Latte Machiato
Mocha Latte
Mocha Kakao
Americano
Tee

KALTGETRÄNKE

Karaffe Wasser
Ice Tea
Saft

SMOOTHIES

Erdbeeren-Banane-Kiwi
Apfel-Drachenfrucht-Kakao
Rhabarber-Himbeere

DAS TREIBT UNS AN – UNSERE PHILOSOPHIE

„MEIN SCHLÜSSELMOMENT WAR, ALS ICH MIT 16 ZUFÄLLIG EINE DOKUMENTATION ÜBER MASSENTIERHALTUNG VON KÜHEN UND HÜHNERN IM FERNSEHEN GESEHEN HABE. MIR IST SOFORT BEWUSST GEWORDEN, DASS ICH SO EINE HALTUNG GEGENÜBER TIEREN NICHT UNTERSTÜTZEN MÖCHTE."

ALMA, GESCHÄFTSFÜHRERIN

Viele Menschen haben unglaublich viele Vorurteile gegenüber Veganismus und Menschen, die sich bewusst dazu entscheiden vegan oder vegetarisch zu ernähren. Auch wir – Alma und Julia – hatten und haben vereinzelt immer noch mit solchen Vorurteilen zu kämpfen, aber davon lassen wir uns nicht mehr herunter ziehen. Wir wissen, warum wir diese Ernährungsweise gewählt haben und wollen euch ein kleinen Einblick geben. Uns liegt nicht nur das Wohl unserer Mitmenschen am Herzen, sondern auch das aller Tiere.

Leider kosten auch andere Produkte außer das direkte Fleisch von Tieren ihr Leben bzw. gefährdet dieses. Für die Produktion von Eiern sind beispielsweise nur weibliche Tiere von Nutzen. So werden alle männlichen

Jeder lebt sein eigenes Leben so, wie er es möchte, allerdings wollen wir unseren eigenen Beitrag leisten, um vielleicht die/den eine/n oder andere/n zum Nachdenken anzuregen.

Mit unseren köstlichen Alternativen zu herkömmlichen, aus Massenproduktion entstandenen Produkten wollen wir zeigen, dass es sehr wohl möglich ist auch ohne tierische Produkte zu leben ohne Abstriche im Geschmack machen zu

gegenüber Veganismus und Menschen, die sich bewusst dazu entscheiden vegan oder vegetarisch zu ernähren. Auch wir – Alina und Julia – hatten und haben vereinzelt immer noch mit solchen Vorurteilen zu kämpfen, aber davon lassen wir uns nicht mehr herunter ziehen. Wir wissen, warum wir diese Ernährungsweise gewählt haben und wollen euch ein kleinen Einblick geben. Uns liegt nicht nur das Wohl unserer Mitmenschen am Herzen, sondern auch das aller Tiere.

Leider kosten auch andere Produkte außer das direkte Fleisch von Tieren ihr Leben bzw. gefährdet dieses. Für die Produktion von Eiern sind beispielsweise nur weibliche Tiere von Nutzen. So werden alle männlichen Küken oftmals direkt nach der Geburt geschreddert oder vergast. Auch bei männlichen Kälbern sieht es weniger rosig aus, da diese oft frühzeitig geschlachtet werden.

TEILE DEIN WEG ZUM VEGANISMUS MIT UNS!

Jeder lebt sein eigenes Leben so, wie er es möchte, allerdings wollen wir unseren eigenen Beitrag leisten, um vielleicht die/den einz/n oder andere/n zum Nachdenken anzuregen.

Mit unseren köstlichen Alternativen zu herkömmlichen, aus Massenproduktion entstandenen Produkten wollen wir zeigen, dass es sehr wohl möglich ist auch ohne tierische Produkte zu leben ohne Abstriche im Geschmack machen zu müssen.

Jedes unserer Leckereien ist mit Liebe und handverlesenen Zutaten aus der Region zubereitet. Damit schützen wir nicht nur unsere Umwelt durch kurze Transportwege, sondern tun gleichzeitig auch noch etwas Gutes für alle Lebewesen auf unserem Planeten.

MÖCHTEST DU MEHR VON UNS ERFAHREN? DANN SCHAU DOCH MAL VORBEI ...

Filiale: Norderstraße 31, Flensburg

FLAVOUR EARTH

Diese Webseite hat keinen integrierten Inhalt und ist Bestandteil der Bachelorthesis von Dorian Grünwald.

Die Website ist mit TailwindCSS als CSS Framework umgesetzt worden.

FLAVOUR EARTH

VEGANES CAFÉ IM HERZEN FLENSBURGS

EST. 2012

Entdecke unser vielfältiges Angebot an veganen Backwaren und erlebe Geschmacksexplosionen aus reinen tierfreien, regionalen Zutaten.

INSPIRIERT VON DER REGION!

All unsere Produkte werden aus tierfreien und regionalen Produkten hergestellt. Dabei achten wir nicht nur auf kurze Transportwege, sondern auch auf faire Arbeitsbedingungen und Qualität. Somit garantieren wir für unseren einzigartigen Geschmack und Fairness für alle!

UNSERE GESCHICHTE

Das Abendteuer begann bei unserer unbegrenzten Liebe zum Backen und beim Ausprobieren neuer Kreationen. Nachdem wir - Alma und Julia - unsere Ausbildungen abgeschlossen hatten, haben wir beide uns beide spontan dazu entschieden, unsere erste gemeinsame Bäckerei zu eröffnen.

Da wir uns seit mehr als 5 Jahren vegan ernähren, sahen wir dies als unsere Chance mit unseren leckeren Rezepten zu überzeugen! Somit ist 2012 unsere vegane Café „Flavour Earth" entstanden. Seitdem entdecken wir jeden Tag neue Rezepte und Zutaten, die unsere Produkte verfeinern.

Seit 2012 bieten wir neben Brot, Brötchen und belegte Sandwiches auch Heiß- und Kaltgetränke an. Von Kaffee Latte bis Erdbeer-Bananen-Kiwi-Smoothie gibt es alles was das Herz begehrt!

Wer wie Alma es eher etwas Süßer mag, wird unsere Auswahl an saisonal-passende Kuchen, Muffins, Croissants und andere Leckereien lieben!

ENTDECKE UNSERE VEGANE VIELFALT

ENTDECKE UNSERE VEGANE VIELFALT

BROT & BRÖTCHEN

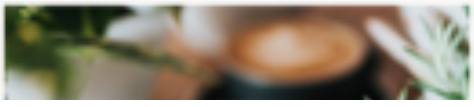
BROT

BRÖTCHEN

ALLES SÜßE

KUCHEN

GEBÄCK & MUFFINS



DURSTLÖSCHER

HEIßGETRÄNKE

KALTGETRÄNKE

SMOOTHIES

DAS TREIBT UNS AN – UNSERE PHILOSOPHIE

„MEIN SCHLÜSSELMOMENT WAR, ALS ICH MIT 16 ZUFÄLLIG EINE DOKUMENTATION ÜBER MASSENTIERHALTUNG VON KÜHEN UND HÜHNERN IM FERNSEHEN GESEHEN HABE. MIR IST SOFORT BEWUSST GEWORDEN, DASS ICH SO EINE HALTUNG GEGENÜBER TIEREN NICHT UNTERSTÜTZEN MÖCHTE."

ALMA, GESCHÄFTSFÜHRERIN

Viele Menschen haben unglaublich viele Vorurteile gegenüber Veganismus und Menschen, die sich bewusst dazu entscheiden vegan oder vegetarisch zu ernähren. Auch wir – Alma und Julia – hatten und haben vereinzelt immer noch mit solchen Vorurteilen zu kämpfen, aber davon lassen wir uns nicht mehr herunter ziehen. Wir wissen, warum wir diese Ernährungsweise gewählt haben und wollen euch ein kleinen Einblick geben. Uns liegt nicht nur das Wohl unserer Mitmenschen am Herzen, sondern auch das aller Tiere.

Leider kosten auch andere Produkte außer

MÖCHTEST DU MEHR VON UNS ERFAHREN? DANN SCHAU DOCH MAL VORBEI ...

Filiale: Norderstraße 31, Flensburg

FLAVOUR EARTH

Diese Webseite hat reinen imaginären Inhalt und ist Bestandteil der Bachelorthesis von Dorien Grönwald.

Die Website ist mit TailwindCSS als CSS Framework umgesetzt worden.

8.2 UMSETZUNG CSS

erste gemeinsame Bäckerei zu eröffnen.

Da wir uns seit mehr als 5 Jahren vegan ernähren, sahen wir dies als unsere Chance mit unseren leckeren Rezepten zu überzeugen! Somit ist 2012 unsere vegane Café „Flavour Earth" entstanden. Seitdem entdecken wir jeden Tag neue Rezepte und Zutaten, die unsere Produkte verfeinern.

Seit 2012 bieten wir neben Brot, Brötchen und belegte Sandwiches auch Heiß- und Kaltgetränke an. Von Kaffee Latte bis Erdbeer-Bananen-Kiwi-Smoothie gibt es alles was das Herz begehrt!

Wer wie Alma es eher etwas Süßer mag, wird unsere Auswahl an saisonal-passende Kuchen, Muffins, Croissants und andere Leckereien lieben!

ENTDECKE UNSERE VEGANE VIELFALT!

BROT & BRÖTCHEN

BROT

BRÖTCHEN

ALLES SÜSSE

KUCHEN

GEBÄCK & MUFFINS

DURSTLÖSCHER

HEISSGETRÄNKE

KALTGETRÄNKE

SMOOTHIES

DAS TREIBT UNS AN – UNSERE PHILOSOPHIE

„MEIN SCHLÜSSELMOMENT WAR, ALS ICH MIT 16 ZUFÄLLIG EINE DOKUMENTATION ÜBER MASSENTIERHALTUNG VON KÜHEN UND HÜHNERN IM FERNSEHEN GESEHEN HABE. MIR IST SOFORT BEWUSST GEWORDEN, DASS ICH SO EINE HALTUNG GEGENÜBER TIEREN NICHT UNTERSTÜTZEN MÖCHTE."

— ALMA, GESCHÄFTSFÜHRERIN

Viele Menschen haben unglaublich viele Vorurteile gegenüber Veganismus und Menschen, die sich bewusst dazu entscheiden vegan oder vegetarisch zu ernähren. Auch wir – Alma und Julia – hatten und haben vereinzelt immer noch mit solchen Vorurteilen zu kämpfen, aber davon lassen wir uns nicht mehr herunter ziehen. Wir wissen, warum wir diese Ernährungsweise gewählt haben und wollen euch ein kleinen Einblick geben. Uns liegt nicht nur das Wohl unserer Mitmenschen am Herzen, sondern auch das aller Tiere.

Leider kosten auch andere Produkte außer das direkte Fleisch von Tieren ihr Leben bzw. gefährdet dieses. Für die Produktion von Eiern sind beispielsweise nur weibliche Tiere von Nutzen. So werden alle männlichen Küken oftmals direkt nach der Geburt geschreddert oder vergast. Auch bei männlichen Kälbern sieht es weniger rosig aus, da diese oft frühzeitig geschlachtet werden.

Jeder lebt sein eigenes Leben so, wie er es möchte, allerdings wollen wir unseren eigenen Beitrag leisten, um vielleicht die/den eine/n oder andere/n zum Nachdenken anzuregen.

Mit unseren köstlichen Alternativen zu herkömmlichen, aus Massenproduktion entstandenen Produkten wollen wir zeigen, dass es sehr wohl möglich ist auch ohne tierische Produkte zu leben ohne Abstriche im Geschmack machen zu

sieht es weniger rosig aus, da diese oft frühzeitig geschlachtet werden.

Jeder lebt sein eigenes Leben so, wie er es möchte, allerdings wollen wir unseren eigenen Beitrag leisten, um vielleicht die/den eine/n oder andere/n zum Nachdenken anzuregen.

Mit unseren köstlichen Alternativen zu herkömmlichen, aus Massenproduktion entstandenen Produkten wollen wir zeigen, dass es sehr wohl möglich ist auch ohne tierische Produkte zu leben ohne Abstriche im Geschmack machen zu müssen.

Jedes unserer Leckereien ist mit Liebe und handverlesenen Zutaten aus der Region zubereitet. Damit schützen wir nicht nur unsere Umwelt durch kurze Transportwege, sondern tun gleichzeitig auch noch etwas Gutes für alle Lebewesen auf unserem Planeten.

TEILE DEINE GESCHICHTE MIT UNS!

MÖCHTEST DU MEHR ERFAHREN? DANN SCHAU DOCH EINFACH VORBEI ...

Café: Norderstraße 31, Flensburg

FLAVOUR EARTH

Diese Webseite hat reinen imaginären Inhalt und ist Bestandteil der Bachelorthesis von Dorian Grönwald.

Die Website ist mit handgeschriebenen CSS umgesetzt worden. Als Präprozessor wurde SASS und als Methodik zur Klassenbenennung des CSS wurde BEM verwendet.

geschichte produkte

philosophie kontakt

8.3 UMSETZUNG TAILWINDCSS

Da wir uns seit mehr als 5 Jahren vegan ernähren, sahen wir dies als unsere Chance mit unseren leckeren Rezepten zu überzeugen! Somit ist 2012 unsere vegane Café „Flavour Earth" entstanden. Seitdem entdecken wir jeden Tag neue Rezepte und Zutaten, die unsere Produkte verfeinern.

Seit 2012 bieten wir neben Brot, Brötchen und belegte Sandwiches auch Heiß- und Kaltgetränke an. Von Kaffee Latte bis Erdbeer-Bananen-Kiwi-Smoothie gibt es alles was das Herz begehrt!
Wer wie Alma es eher etwas Süßer mag, wird unsere Auswahl an saisonal-passende Kuchen, Muffins, Croissants und andere Leckereien lieben!

ENTDECKE UNSERE VEGANE VIELFALT!

BROT & BRÖTCHEN

BROT

BRÖTCHEN

ALLES SÜSSE

KUCHEN

GEBÄCK & MUFFINS

DURSTLÖSCHER

HEISSGETRÄNKE

KALTGETRÄNKE

SMOOTHIES

DAS TREIBT UNS AN – UNSERE PHILOSOPHIE

„MEIN SCHLÜSSELMOMENT WAR, ALS ICH MIT 16 ZUFÄLLIG EINE DOKUMENTATION ÜBER MASSENTIERHALTUNG VON KÜHEN UND HÜHNERN IM FERNSEHEN GESEHEN HABE. MIR IST SOFORT BEWUSST GEWORDEN, DASS ICH SO EINE HALTUNG GEGENÜBER TIEREN NICHT UNTERSTÜTZEN MÖCHTE."

ALMA, GESCHÄFTSFÜHRERIN

Viele Menschen haben unglaublich viele Vorurteile gegenüber Veganismus und Menschen, die sich bewusst dazu entscheiden vegan oder vegetarisch zu ernähren. Auch wir – Alma und Julia – hatten und haben vereinzelt immer noch mit solchen Vorurteilen zu kämpfen, aber davon lassen wir uns nicht mehr herunter ziehen. Wir wissen, warum wir diese Ernährungsweise gewählt haben und wollen euch ein kleinen Einblick geben. Uns liegt nicht nur das Wohl unserer Mitmenschen am Herzen, sondern auch das aller Tiere.

Leider kosten auch andere Produkte außer das direkte Fleisch von Tieren ihr Leben bzw. gefährdet dieses. Für die Produktion von Eiern sind beispielsweise nur weibliche Tiere von Nutzen. So werden alle männlichen Küken oftmals direkt nach der Geburt geschreddert oder vergast. Auch bei männlichen Kälbern sieht es weniger rosig aus, da diese oft frühzeitig geschlachtet werden.

Jeder lebt sein eigenes Leben so, wie er es möchte, allerdings wollen wir unseren eigenen Beitrag leisten, um vielleicht die/den eine/n oder andere/n zum Nachdenken anzuregen.

Mit unseren köstlichen Alternativen zu herkömmlichen, aus Massenproduktion entstandenen Produkten wollen wir zeigen, dass es sehr wohl möglich ist auch ohne tierische Produkte zu leben ohne Abstriche im Geschmack machen zu

männlichen Küken oftmals direkt nach der Geburt geschreddert oder vergast. Auch bei männlichen Kälbern sieht es weniger rosig aus, da diese oft frühzeitig geschlachtet werden.

Jeder lebt sein eigenes Leben so, wie er es möchte, allerdings wollen wir unseren eigenen Beitrag leisten, um vielleicht die/den eine/n oder andere/n zum Nachdenken anzuregen.

Mit unseren köstlichen Alternativen zu herkömmlichen, aus Massenproduktion entstandenen Produkten wollen wir zeigen, dass es sehr wohl möglich ist auch ohne tierische Produkte zu leben ohne Abstriche im Geschmack machen zu müssen.

Jedes unserer Leckereien ist mit Liebe und handverlesenen Zutaten aus der Region zubereitet. Damit schützen wir nicht nur unsere Umwelt durch kurze Transportwege, sondern tun gleichzeitig auch noch etwas Gutes für alle Lebewesen auf unserem Planeten.

TEILE DEINE GESCHICHTE MIT UNS!

MÖCHTEST DU MEHR ERFAHREN? DANN SCHAU DOCH EINFACH VORBEI ...

Café: Norderstraße 31, Flensburg

FLAVOUR EARTH

Diese Webseite hat reinen imaginären Inhalt und ist Bestandteil der Bachelorthesis von Dorien Grönwald.

Die Website ist mit dem Utility-Klassen basierten CSS-Framework TailwindCSS umgesetzt worden.

geschichte produkte

philosophie kontakt

8.4 UMSETZUNG BOOTSTRAP

Earth* entstanden. Seitdem entdecken wir jeden Tag neue Rezepte und Zutaten, die unsere Produkte verfeinern.

Seit 2012 bieten wir neben Brot, Brötchen und belegte Sandwiches auch Heiß- und Kaltgetränke an. Von Kaffee Latte bis Erdbeer-Bananen-Kiwi-Smoothie gibt es alles was das Herz begehrt!

Wer wie Alma es eher etwas Süßer mag, wird unsere Auswahl an saisonal-passende Kuchen, Muffins, Croissants und andere Leckereien lieben!

ENTDECKE UNSERE VEGANE VIELFALT!

Brot & Brötchen

BROT

BRÖTCHEN

Alles Süße

KUCHEN

GEBÄCK & MUFFINS

Durstlöscher

Durstlöscher

HEISSGETRÄNKE

KALTGETRÄNKE

SMOOTHIES

DAS TREIBT UNS AN – UNSERE PHILOSOPHIE

„MEIN SCHLÜSSELMOMENT WAR, ALS ICH MIT 16 ZUFÄLLIG EINE DOKUMENTATION ÜBER MASSENTIERHALTUNG VON KÜHEN UND HÜHNERN IM FERNSEHEN GESEHEN HABE. MIR IST SOFORT BEWUSST GEWORDEN, DASS ICH SO EINE HALTUNG GEGENÜBER TIEREN NICHT UNTERSTÜTZEN MÖCHTE."

ALMA, GESCHÄFTSFÜHRERIN

Viele Menschen haben unglaublich viele Vorurteile gegenüber Veganismus und Menschen, die sich bewusst dazu entscheiden vegan oder vegetarisch zu ernähren. Auch wir – Alma und Julia – hatten und haben vereinzelt immer noch mit solchen Vorurteilen zu kämpfen, aber davon lassen wir uns nicht mehr herunter ziehen. Wir wissen, warum wir diese Ernährungsweise gewählt haben und wollen euch ein kleinen Einblick geben. Uns liegt nicht nur das Wohl unserer Mitmenschen am Herzen, sondern auch das aller Tiere.

Leider kosten auch andere Produkte außer das direkte Fleisch von Tieren ihr Leben bzw. gefährdet dieses. Für die Produktion von Eiern sind beispielsweise nur weibliche Tiere von Nutzen. So werden alle männlichen Küken oftmals direkt nach der Geburt geschreddert oder vergast. Auch bei männlichen Kälbern sieht es weniger rosig aus, da diese oft frühzeitig geschlachtet werden.

Jeder lebt sein eigenes Leben so, wie er es möchte, allerdings wollen wir unseren eigenen Beitrag leisten, um vielleicht die/den eine/n oder andere/n zum Nachdenken anzuregen.

Mit unseren köstlichen Alternativen zu herkömmlichen, aus Massenproduktion entstandenen Produkten wollen wir zeigen, dass es sehr wohl möglich ist auch ohne

vergast. Auch bei männlichen Kälbern sieht es weniger rosig aus, da diese oft frühzeitig geschlachtet werden.

Jeder lebt sein eigenes Leben so, wie er es möchte, allerdings wollen wir unseren eigenen Beitrag leisten, um vielleicht die/den eine/n oder andere/n zum Nachdenken anzuregen.

Mit unseren köstlichen Alternativen zu herkömmlichen, aus Massenproduktion entstandenen Produkten wollen wir zeigen, dass es sehr wohl möglich ist auch ohne tierische Produkte zu leben ohne Abstriche im Geschmack machen zu müssen.

Jedes unserer Leckereien ist mit Liebe und handverlesenen Zutaten aus der Region zubereitet. Damit schützen wir nicht nur unsere Umwelt durch kurze Transportwege, sondern tun gleichzeitig auch noch etwas Gutes für alle Lebewesen auf unserem Planeten.

TEILE DEINE GESCHICHTE MIT UNS!

MÖCHTEST DU MEHR ERFAHREN? DANN SCHAU DOCH EINFACH VORBEI ...

Café: Norderstraße 31, Flensburg

FLAVOUR EARTH

Diese Webseite hat reinen imaginären Inhalt und ist Bestandteil der Bachelorthesis von Dorien Grönwald.

Die Website ist mit dem Komponenten basierten CSS Framework Bootstrap umgesetzt worden.

geschichte produkte

philosophie kontakt

ERKLÄRUNG DER SELBSTSTÄNDIGKEIT

Ich versichere, dass ich die vorliegende Thesis ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Quellen benutzt habe.

Flensburg, den 06.10.2021