

25knots – Ein Tool zur Verbesserung der gestalterischen Qualität von Artefakten im Hochschulkontext

Umsetzung vom Proof of Concept zur marktfähigen Webanwendung

BACHELORARBEIT

vorgelegt an der Technischen Hochschule Köln

Campus Gummersbach

Im Studiengang Medieninformatik

ausgearbeitet von

Christian Alexander Poplawski

MATRIKELNUMMER 11088931

Erster Prüfer: Prof. Dipl. Des. Christian Noss

Technische Hochschule Köln

Zweiter Prüfer: Dipl. Des. Liane Kirschner

Railslove GmbH

Gummersbach, im August 2017

Inhaltsverzeichnis

1	Einleitung	4
1.1	Motivation	5
1.2	Zielsetzung	5
1.3	Relevanz des Themas	7
1.4	Zielgruppe	8
1.5	Struktur der vorliegenden Arbeit	8
2	Konzeption der Anwendung	10
2.1	Generelle Struktur der Anwendung	10
2.2	Einstieg in die Anwendung	12
2.3	Typographie	13
2.4	Farben	15
2.5	Layouts & Grid	16
2.5.1	Anmerkung zur Umsetzung	17
2.6	Ergebnisse der Benutzung	18
3	Technischen Grundlagen	20
3.1	Diskussion verfügbarer Technologien	20
3.1.1	Rich Internet Applications	21
3.1.2	Single Page Applications	21
3.1.2.1	JavaScript Single Page Application Frameworks	22
3.2	Einstieg in React.js	24
3.2.1	Komponenten	24
3.2.1.1	Was ist eine Komponente?	25
3.2.1.2	Komponenten in React.js	25

3.2.1.3	Zustandslose Komponenten	26
3.2.2	JSX	27
3.3	Einstieg in Redux	28
4	Umsetzung der Anwendung	29
4.1	Gestaltung	29
4.1.1	Vorgehen	31
4.1.2	Abstände	32
4.1.3	Styleguide	34
4.2	Architektur	34
4.2.1	Redux	34
4.2.1.1	Beispielhaftes Verändern des Redux Store	35
4.2.2	CSS-Architektur	35
4.3	Interessante Aspekte in der Entwicklung	38
4.3.1	State in Komponenten	38
4.3.2	Dispatch von Actions	40
4.3.3	Anzeige von Inhalten nach Scope	41
4.3.4	Modularisierung der Anwendung	42
4.3.5	Erstellen von Farbkontrasten	43
4.3.6	Erstellen von PDF-Dateien	46
5	Veröffentlichung der Anwendung	48
5.1	Hosting	48
5.2	Weiterentwicklung	50
5.2.1	Sicherung der Code-Qualität	50
5.2.2	Dokumentation	51
5.2.3	Contribution Guidelines	52
5.2.4	Tests	52
5.3	Vermarktung	52
6	Fazit & Ausblick	53
6.1	Zielerreichungsgrad	53
6.2	Ausblick	54
6.3	Fazit	55

Abbildungsverzeichnis

2.1	Zielmedien der Nutzer und deren Unterkategorien	12
2.2	Der Bereich Typographie, im Praxisprojekt und der Abschlussarbeit	14
2.3	Vorschau der ausgewählten Farben in der Anwendung	17
4.1	Farben verschiedener interaktiver Elemente	30
4.2	Fehlermeldungen im Bereich Typographie	31
4.3	Buttons zum Auswählen einer Akzentfarbe	32

Kapitel 1

Einleitung

Die nachfolgende Arbeit beschäftigt sich mit der Entwicklung der Webanwendung *25Knots* zu einem Marktfähigen Produkt, basierend auf bereits im Praxisprojekt erarbeiteten Konzepten und einem Proof of Concept.

Ziel ist dabei nicht nur, die Verwendung durch die Community¹ nach Abschluss dieser Arbeit möglich zu machen, sondern diese auch aktiv an der Weiterentwicklung der Anwendung teilhaben zu lassen.

Die Anwendung *25Knots* zielt darauf ab, die gestalterische Qualität von Artefakten im Hochschulkontext zu verbessern. Dieses Ziel soll durch die Unterstützung von Studierenden während des Entwicklungsprozesses von Artefakten erreicht werden. Die Anwendung soll den Studierenden dabei eine Möglichkeit bieten, interaktiv Ergebnisse zu produzieren, die für ihren aktuellen Entwicklungsprozess hilfreich sind. Wissen soll dabei eher implizit, durch ein *Learning by Doing* Konzept vermittelt werden.

Generell sollen in dieser Arbeit alle Aspekte behandelt werden, die auf dem Weg von einem Prototypen zu einem marktfähigen Produkt eine Rolle spielen. Diese umfassen zum Beispiel die Struktur und Gestaltung der Anwendung, technische Entscheidungen die bei der Umsetzung getroffen werden müssen, die generelle Entwicklung, aber auch Bereiche wie Hosting oder Möglichkeiten zur Weiterentwicklung. Im Folgenden Kapitel sollen zunächst einige grundlegende Ziele und das generelle Vorgehen bei der Umsetzung

erläutert werden

1.1 Motivation

Wie bereits erwähnt basiert das Thema dieser Arbeit auf Konzepten, die im Rahmen des Praxisprojektes erstellt wurden. Bei der Erstellung dieser Konzepte wurde explizit auf eine spätere Umsetzbarkeit geachtet, so wurden zum Beispiel schon einige grobe Berechnungsmethoden entworfen, die in der Anwendung verwendet werden können.

Die Umsetzung des Proof of Concept im Bereich Typographie hat außerdem gezeigt, dass die erstellten Konzepte in einer Anwendung durchaus funktionieren.

Weiterhin konnte ich in persönlichen Gesprächen mit verschiedenen Personen innerhalb und auch außerhalb des Hochschulkontextes feststellen, dass eine Umsetzung der erarbeiteten Konzepte durchaus eine reale Zielgruppe besitzt.

Die Motivation für diese Arbeit ergibt sich daher aus dem Willen, aus diesem erarbeiteten Konzept einen Mehrwert schaffen zu wollen, der über den Erhalt von *Credit Points* hinaus geht. Dieser Mehrwert entsteht dabei zum einen für mich persönlich, zum anderen aber auch für andere Personen, die einen Mehrwert aus der Existenz einer solchen Anwendung ziehen können.

Die generelle Motivation für die Existenz einer Anwendung wie *25knots* lässt sich dem Kapitel *Relevanz des Themas* auf Seite 7 entnehmen.

1.2 Zielsetzung

Ziel der vorliegenden Arbeit soll es sein, ein von der Community verwendbares und erweiterbares Produkt zu entwickeln. Zunächst sei hier ein *verwendbares Produkt* im Rahmen dieser Arbeit definiert.

Ein verwendbares Produkt:

1. bietet eine befriedigende Nutzererfahrung

¹Als Teil der *Community* wird hier jede Person gesehen, die ein Interesse an der Anwendung besitzt.

2. ist einfach zugänglich

3. ist bei potentiellen Nutzern als Hilfsmittel zum Erreichen eines Zieles bekannt

Eine befriedigende Nutzererfahrung bedeutet für die Anwendung konkret, dass diese gut Strukturiert, ansprechend Gestaltet und ohne Probleme verwendbar sein muss. *Ohne Probleme verwendbar* impliziert hierbei eine hohe Qualität des geschriebenen Codes, um Fehler zu vermeiden. Der Großteil der Arbeit wird sich mit der Umsetzung dieses Bereiches beschäftigen.

Um die Anwendung einfach zugänglich zu machen, bietet sich das Hosting auf einem Server an (im Gegensatz zu beispielsweise nur der Bereitstellung des Anwendungscodes). Weitere details zu diesem Thema werden im Kapitel *Hosting* auf Seite 48 erläutert.

Damit die Anwendung genutzt werden kann, müssen ihre potentiellen Nutzer von ihrer Existenz wissen. Mit Blick auf die Zielgruppe bietet sich zunächst eine Bewerbung direkt an der Hochschule, beispielsweise durch die Teilnahme am *Medieninformatik Showcase* an. Aber auch eine Bewerbung im größeren Rahmen, beispielsweise durch einen Talk beim *DevHouse Friday Köln* ist denkbar.

Zuletzt sei auch die erweiterbarkeit der Anwendung konkret definiert. Eine mögliche Erweiterbarkeit bedeutet zunächst, dass der Code der Anwendung öffentlich verfügbar sein muss, beispielsweise auf der Plattform Github². Weiterhin muss der Code gut dokumentiert und verständlich geschrieben sein, um einen Einstieg in das bestehende Projekt für Dritte so einfach wie möglich zu halten. Eine erweiterbarkeit bezieht sich aber nicht nur auf geschriebenen Code, sondern kann (und soll) auch auf konzeptioneller Ebene erfolgen. Auch hier muss die Möglichkeit zur Beteiligung so simpel wie möglich gehalten werden. Eine ausführlichere Diskussion findet sich im Kapitel *Release*

Aus diesen Punkten kann eine zentrale Forschungsfrage für die Arbeit formuliert werden:

Wie kann der Community ein nutzbares und erweiterbares Produkt auf Basis eines Konzeptes und Prototypen bereitgestellt werden?

Diese lässt sich in zwei Unterfragen unterteilen, die es in dieser Arbeit zu beantworten gilt:

²<https://github.com>

1. Durch welche Maßnahmen kann eine aktive Nutzung und Weiterentwicklung durch die Community gewährleistet werden?
2. Welche technischen Entscheidungen müssen während der Entwicklung getroffen werden?

Auch wenn es Teil der Zielsetzung ist, ein marktfähiges Produkt zu erstellen kann hier nicht davon ausgegangen werden, dass das Endergebnis der Arbeit ein Produkt ist, das als fertig angesehen werden kann. Auch mit Blick auf die spätere Weiterentwicklung durch die Community muss es viel mehr das Ziel sein, eine hochwertige erste Version des Produktes, die als Grundlage für weitere Features dient, also ein *Minimum Viable Product* zu erstellen.

1.3 Relevanz des Themas

Die Relevanz der Anwendung an sich wurde bereits im Praxisprojekt erläutert, daher soll hier nur eine Kurzfassung der Erläuterung folgen. Der Grundgedanke der Relevanz ist dabei folgender:

Menschen bilden sehr schnell ein Urteil über die Gestaltung eines Artefaktes und dieses lässt sich nur schwer wieder ändern. Weiterhin wird dieser schlechte erste Eindruck auf andere Bereiche des Artefaktes übertragen und wirkt sich somit unter Umständen auch auf die Gesamtbewertung eines Artefaktes aus.

Gestützt wird dieser Gedanke durch Studien von [LFDB06], [CP96], und [Nic98].

Unterstützend seien hier noch zwei weitere Quellen aufgeführt, die die Relevanz weiter unterstreichen: [TCKS06] zeigen, dass die Ergebnisse der Studie von Lindgaard et al. auch mit anderen Parametern bestand haben und unterstreichen weiterhin die Wichtigkeit von guter Gestaltung für eine gute Nutzererfahrung [TKI00].

Neben der Relevanz des Produktes, das in Rahmen der Arbeit entstehen soll sei aber auch das übergreifende Thema der Arbeit angesprochen: Die Entwicklung von einem Konzept zu einem fertigen Produkt. Als abschließende Arbeit für den Studiengang Medieninformatik ist dieses ein passendes Thema, da hier viele Aspekte aus verschiedenen Modulen des gesamten Studiums vereint werden. Daher bietet das Thema eine gute Verbindung zwischen den verschiedenen Disziplinen innerhalb des Studiums und einer wissenschaftlichen

Diskussion verschiedener Vorgehensweisen und Abläufe.

1.4 Zielgruppe

Während der Konzeption im Praxisprojekt wurden Studenten der Technischen Hochschule Köln im Studiengang Medieninformatik als Zielgruppe festgelegt. Es wurde aber bereits dort deutlich, dass diese Zielgruppe leicht erweiterbar ist. Somit kann jede Person einen Mehrwert aus diesem Tool ziehen, die ein Artefakt erstellen muss dessen Hauptaugenmerk eigentlich nicht auf der Gestaltung, sondern auf einer bestimmten Funktion liegt. Die fehlende Aufmerksamkeit für die Gestaltung kann im Studium an einer fehlenden Bewertung dieser oder in der Wirtschaft an einem mangelndem Budget liegen. Es entstehen also häufig Situationen, in denen eine solide Gestaltung nicht als wichtig erachtet wird, diese jedoch, wie im vorhergehenden Kapitel erwähnt, durchaus Vorteile mit sich bringt.

Für diese Arbeit werden aber zunächst weiterhin die Studenten des Studienganges Medieninformatik als Zielgruppe definiert, um eine Disparität zwischen dem Konzept und der Umsetzung auszuschließen.

1.5 Struktur der vorliegenden Arbeit

Außerhalb dieser Einleitung teilt sich die Arbeit in drei weitere Kapitel.

Im zweiten Kapitel werden zunächst einige Theoretische Grundlagen und Entscheidungen auf einer taktischen Ebene erläutert. Dort findet sich neben einigen strukturellen Erläuterungen auch eine Diskussion verschiedener Technologien, mit denen eine Umsetzung möglich gewesen wäre und ein Einstieg in verschiedene Aspekte der Programmierung mit React.js.

Im dritten Kapitel werden einige Aspekte der Umsetzung und Gestaltung der Anwendung angesprochen. Es finden sich beispielsweise Erläuterungen zu den einzelnen Bereichen der Anwendung, aber auch Bereichsübergreifende praktische Themen wie die CSS-Architektur.

Das vierte Kapitel beschäftigt sich mit Fragen die im Bezug mit der Veröffentlichung der Anwendung stehen.

NOTIZ: Hier wird am Ende noch einmal drüber geschaut, wenn die tatsächliche Struktur etwas deutlicher ist.

Kapitel 2

Konzeption der Anwendung

Zu Beginn sind der genaue Rahmen und der Ablauf der Anwendung zu definieren. Hierfür muss zunächst der Stand aus dem Praxisprojekt evaluiert werden, um festzustellen, an welchen Stellen weitere Konzeptionsarbeit geleistet werden muss, um eine konkrete Umsetzung der Konzepte zu ermöglichen.

Das folgende Kapitel beschäftigt sich mit der Spezifizierung des generellen Ablaufes der Anwendung und geht darauf folgend auf die einzelnen Abschnitte ein.

2.1 Generelle Struktur der Anwendung

Innerhalb des Praxisprojektes wurden verschiedene Themengebiete behandelt, die zunächst auf eine Umsetzbarkeit hin validiert werden müssen. Die im Praxisprojekt definierten Themengebiete [Pop16] sind:

- Typographie
- Layout & Struktur
- Whitespace
- Farben
- Bilder
- Interaktive Elemente

Der Bereich *Whitespace* konnte für eine Umsetzung schnell ausgeschlossen werden, da obgleich seiner Wichtigkeit für eine gute Gestaltung während des Praxisprojektes keine Konzepte gefunden werden konnten, auf deren Basis eine Umsetzung möglich wäre.

Vor der Zielsetzung, innerhalb der Abschlussarbeit eine Marktfähige (und damit in ihren Features vollständige) Anwendung zu erstellen und der gegebenen Zeit von 9 Wochen für deren Umsetzung, muss die Liste der zu behandelnden Gebiete noch weiter eingeschränkt werden.

Die Bereiche *Bilder* und *Interaktive Elemente* zeigen im Vergleich die niedrigste Relevanz für eine grundlegende Gestaltung auf. Der Bereich *Bilder* ist ein eher technischer, der sich auf die korrekte Handhabung von Bildern fokussiert. Der Bereich *Interaktive Elemente* ist zwar auch für die Grundlagen der Gestaltung von großer Bedeutung, jedoch ist dieser nur für zwei der drei definierten Zielmedien der Nutzer (vgl. Kapitel 2.2) von Bedeutung (Interaktive Elemente kommen in Textdokumenten nicht vor). Hieraus ergibt sich als Liste von Themen für die Umsetzung innerhalb der Abschlussarbeit:

- Typographie
- Layout & Struktur
- Farben

Außerdem sollten für die Anwendung jeweils ein nutzerfreundlicher Einstieg und Ausstieg gefunden werden. Diese wurden im Praxisprojekt nicht explizit ausgearbeitet und fallen somit auch konzeptionell in den Bereich der Abschlussarbeit und werden später in diesem Kapitel behandelt. Die finale Struktur der Anwendung für den Rahmen dieser Arbeit sieht also wie folgt aus (der Bereich *Layout & Struktur* wurde dabei in *Layout & Grids* umbenannt):

- Einstieg
- Typographie
- Layout & Grids
- Farben
- Ausstieg

2.2 Einstieg in die Anwendung

Bereits im Praxisprojekt wurde festgestellt, dass es sinnvoll ist, das Zielmedium des Nutzers zu kennen:

Da sich dieses Tool nicht über die plattformspezifischen Richtlinien hinwegsetzen soll, sollte von Anfang an die Plattform, für die der Nutzer gestaltet, bekannt sein. Die Inhalte des Tools sollten sich dementsprechend anpassen.

[Pop16]

Für die Zielgruppe der Studenten wurden hier drei mögliche Zielmedien definiert: Native App, Website und Textdokument. Diese Zielmedien können aber auch in sich noch weitere Unterkategorien aufweisen. So kann ein Textdokument beispielsweise für das Lesen an einem Bildschirm oder das Lesen in gedruckter Form, oder eine Native App für unterschiedliche Betriebssystem mit unterschiedlichen Gestaltungsrichtlinien entworfen werden. Eine komplette Auflistung der möglichen Zielmedien und ihrer Unterkategorien, die für die hier definierten Themengebiete von Bedeutung sind findet sich in Abbildung 2.1. Für den weiteren Verlauf dieses Dokumentes sollen die jeweiligen Zielmedien als *scopes* bezeichnet werden.

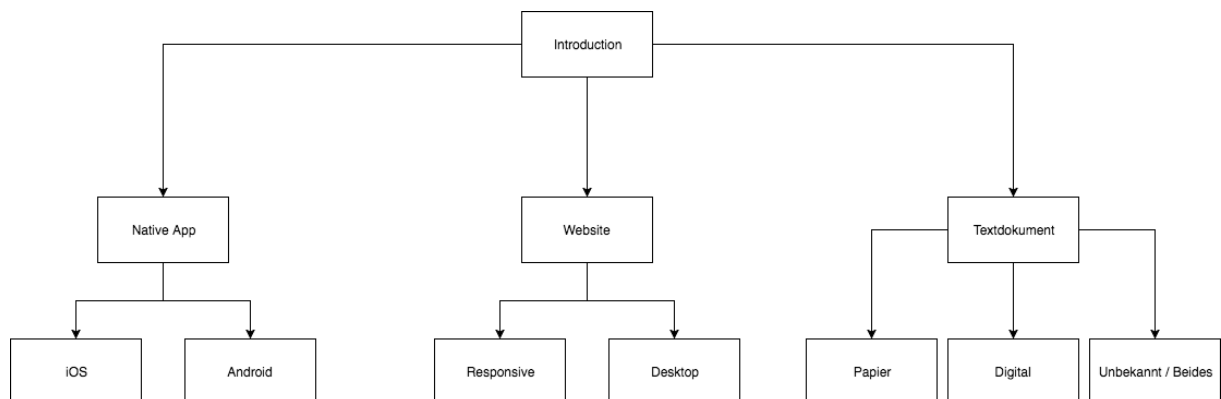


Abbildung 2.1: Zielmedien der Nutzer und deren Unterkategorien

Obwohl die Abgrenzung der Bereiche für die hier definierte Zielgruppe ausreichend ist, lassen sich bereits hier einige Stellen erkennen, die bei einer möglichen späteren Erweiterung der Zielgruppe überarbeitet werden müsste. Vorrangig betrifft das den Bereich *Website*. Hier ist die vorhandene Unterteilung in *Responsive* und *Desktop* für ein Echtwelt-Szenario unter Umständen zu allgemein gehalten.

Hier gilt es außerdem festzulegen, wie genau der Nutzer der Anwendung sein jeweiliges Zielmedium mitteilen soll. Ziel muss es dabei sein, die kognitive Arbeit¹ für diesen so gering wie möglich zu halten. Die in Abbildung 2.1 gezeigte Struktur legt hier bereits eine Möglichkeit nahe, dem Nutzer immer nur eine Ebene des Baumes zur gleichen Zeit zu zeigen und so die Zahl der Auswahlmöglichkeiten so gering wie möglich zu halten.

Der Nutzer soll hierfür einen Wizard² durchlaufen, der maximal drei Auswahlmöglichkeiten zur gleichen Zeit darstellt. Zur weiteren Unterstützung und zur einfacheren Identifizierbarkeit der Optionen sollen die Interface-Elemente für die verschiedenen Auswahlmöglichkeiten außerdem aus Icon und Text bestehen. Nach Beendigung des Wizards soll dem Nutzer seine getroffene Wahl noch einmal angezeigt werden, um so Fehler zu vermeiden, die unter Umständen zu einem späteren Zeitpunkt in der Anwendung den gesamten erarbeiteten Fortschritt nutzlos machen würden.

2.3 Typographie

Durch die Umsetzung des Bereiches Typographie als Proof of Concept innerhalb des Praxisprojektes wurde hier bereits viel grundlegende Konzeptarbeit verrichtet, auf der hier aufgebaut werden kann.

Die Grundlegende Interaktion wurde dabei beibehalten: Weiterhin sieht der Nutzer einen Text, der sich auf seine Eingaben hin verändert. Die Eingabe erfolgt weiterhin primär über Slider, die in einer Tab-Navigation gruppiert sind. Die Platzierung dieser beiden Hauptelemente wurde jedoch verändert, sodass diese jetzt nebeneinander angeordnet sind, und nicht übereinander (vgl. Abbildung 2.2). Diese Anordnung bietet die Möglichkeit, sowohl die Bedienelemente, als auch den gesetzten Text zu jeder Zeit sehen zu können und übermäßiges Scrollen zu vermeiden.

Eine weitere Verbesserung wurde im Bereich der Fehleranzeige vorgenommen: Hier wird über ein Icon in der Tab-Navigation deutlich gemacht, dass in einem bestimmten Bereich ein Fehler vorliegt.

¹Als kognitive Arbeit werden Prozesse bezeichnet, die ein Nutzer durchführen muss, obwohl diese nicht sein eigentliches Anwendungsziel unterstützen. [Goo09, S. 410]

²“A **wizard** [...] is an enforced sequence of actions; [...]” [Goo09, S. 418]

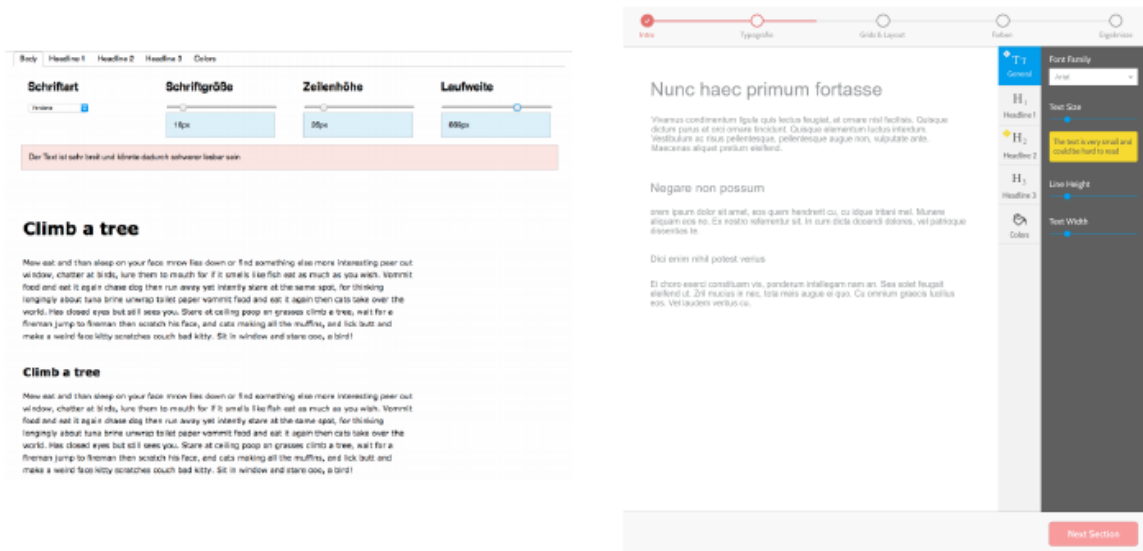


Abbildung 2.2: Der Bereich Typographie, im Praxisprojekt und der Abschlussarbeit

Einen weiteren Ansatz stellt die direkte Interaktion mit Elementen dar. So bestünde beispielsweise die Möglichkeit, eine Überschrift anzuklicken und deren Attribute daraufhin direkt zu editieren. Hier wird, im Vergleich zur Interaktion mit Tab-Navigation und Schieberegler, zwar schneller deutlich, auf welches Element die getätigte Eingabe wirkt, jedoch bietet dieser Ansatz auch einige Nachteile. So können sich Elemente innerhalb des Textes überschneiden (beispielsweise die gesamte Breite des Textes und ein einzelner Textabsatz), wodurch eine genaue Auswahl erschwert wird. Weiterhin ergeben sich Probleme bei einer übersichtlichen Fehleranzeige: Es stellt sich als kompliziert dar, deutlich zu machen, zu welchem Teil des gesetzten Textes ein Fehler zugehörig ist.

Da diese Art der Interaktion außerdem nicht sehr verbreitet ist und eine Erklärung für den Nutzer erfordern würde, wurde dieser Ansatz nicht weiter verfolgt.

Eine konzeptuelle Neuerung stellt ein Button zum zurücksetzen der Werte auf die Standardeinstellungen dar. In Situationen, in denen der Nutzer in verschiedenen Bereiche Werte verändert hat und mit diesen unzufrieden ist, bietet der Button eine komfortable Methode für einen Neustart des Schrittes.

Im Bezug auf die Standardeinstellungen stellt sich die Frage, welche Einstellungen hier zu verwenden sind. Es wurde sich bewusst gegen eine fehlerfreie Standardeinstellung entschieden, da es dem Nutzer nicht möglich ist, in der Anwendung einen Schritt vorwärts zu

gehen, wenn noch Warnungen angezeigt werden. So kann sicher gestellt werden, dass der Nutzer sich auf jeden Fall mit dem Bereich Typographie befasst. Auf der anderen Seite sollten die Standardeinstellung nicht zu viele Fehler enthalten, um den Nutzer nicht zu demotivieren. Hier wurde nur ein einziger fehlerhafter Wert gewählt, der im ersten Tab zu sehen und dadurch mit wenig Aufwand zu beheben ist.

2.4 Farben

Während des Praxisprojektes wurde auch hier bereits ein grundlegendes Konzept aufgestellt. Die wichtigsten Punkte, in wenigen Sätzen zusammen gefasst, sind dabei: Die Farbfindung geschieht in zwei Schritten, das Finden der Grundfarbe und das Finden der Akzentfarbe. Farben sollen dabei auch nach Emotionen oder Adjektiven wählbar sein. Für die Betriebssysteme iOS und Android bestehen dabei gesonderte Regeln. [Pop16]

Im ersten Schritt wurden die verschiedenen Darstellungen für die unterschiedlichen *scopes* festgelegt.

- **Finden einer Grundfarbe:** Hier wurde das Konzept aus dem Praxisprojekt übernommen. Für jeden *scope* werden die jeweiligen Farben zur Auswahl gezeigt, wobei diese Auswahl bei den mobilen Betriebssystemen durch deren Richtlinien begrenzt ist und somit in Form von Buttons realisiert werden kann. Für die *scopes* Website und Textdokument ist die Auswahl nicht begrenzt und die Selektion der Farbe findet hier durch einen Colorpicker statt.

Für den *scope* Android werden bei einer Auswahl durch den Nutzer die Abstufungen 300 und 500 automatisch zum Farbschema hinzugefügt, um den Raum für Fehler so gering wie möglich zu halten.

- **Finden einer Akzentfarbe:** Für den *scope* iOS entfällt dieser Schritt, für die beiden anderen Schritte wurde dem Nutzer die Möglichkeit gegeben, über Buttons die Art der Akzentfarbe zu wechseln (im Bereich Android über die Helligkeit, in anderen Bereichen in Form des gewählten Kontrastes).
- **Anzeigen des Farbschemas:** Wie bereits beim Einstieg in die Anwendung soll

dem Nutzer am Ende das gesamte Farbschema angezeigt werden, bevor er zum nächsten Schritt in der Anwendung wechselt. Dies ist vor allem Nötig, weil der Nutzer während der Erstellung immer nur Teile des gesamten Farbschemas sieht.

Als schwierig gestaltete sich die Zuordnung von Farben zu bestimmten Adjektiven. Viele Quellen machen zwar Angaben über die Wirkung von bestimmten Grundfarben, genaue Farbabstufungen (zum Beispiel in Form von HEX- oder RGB-Werten) lassen sich aber nicht finden. Ziel der Anwendung soll es aber sein, dem Nutzer einen Konkreten Farbwert zu empfehlen, mit dem dieser Arbeiten kann.

Die Farbfindung über Adjektive hat in der Anwendung vor allem den Zweck, dem Nutzer das Finden einer Farbe zugänglicher zu machen. Es ist **nicht** das Ziel der Anwendung, dem Artefakt des Nutzers eine bestimmte emotionale Wirkung zu geben. Daher ist hier ein subjektives Festlegen der genauen Farbwerte eine akzeptable Lösung. Konkret wurden aus [Wri17] und [Goo09] verschiedene Adjektive zu den Grundfarben extrahiert, denen dann konkrete Werte zugewiesen wurden.

Dem Nutzer soll weiterhin, bereits während der Auswahl, eine möglichst genaue Vorstellung von der Wirkung der von ihm gewählten Farbe gegeben werden. Deswegen verfärbt sich ein großer Teil des Hintergrundes während dieses Schrittes entsprechend der Auswahl des Nutzers. Diese Hintergrundfärbung wird auch bei der Wahl der Akzentfarben beibehalten, wobei die Akzentfarben als kleinere Flächen auf der Grundfarbe angezeigt werden. Auch hier soll ein direkter Einblick in die mögliche spätere Verwendung geschaffen werden. Abbildung 2.3 zeigt diesen Ansatz im fertigen Produkt.

2.5 Layouts & Grid

Im Bereich Layouts und Grid zeigten sich die deutlichsten Unterschiede in der Art und Weise, in der die Anwendung auf das Entwicklungsziel des Nutzer reagiert. So können für native Anwendungen im mobilen Bereich lediglich Empfehlungen und Verweise auf externe Quellen gegeben werden, während für Textdokumente die Möglichkeit besteht, während der Anwendung konkrete Ergebnisse zu erarbeiten [Pop16].

Während der Arbeit wurde für die Bearbeitung von Textdokumenten das Layout an den Bereich Typographie angepasst, da die Interaktionen hier ähnlich sind und somit für den

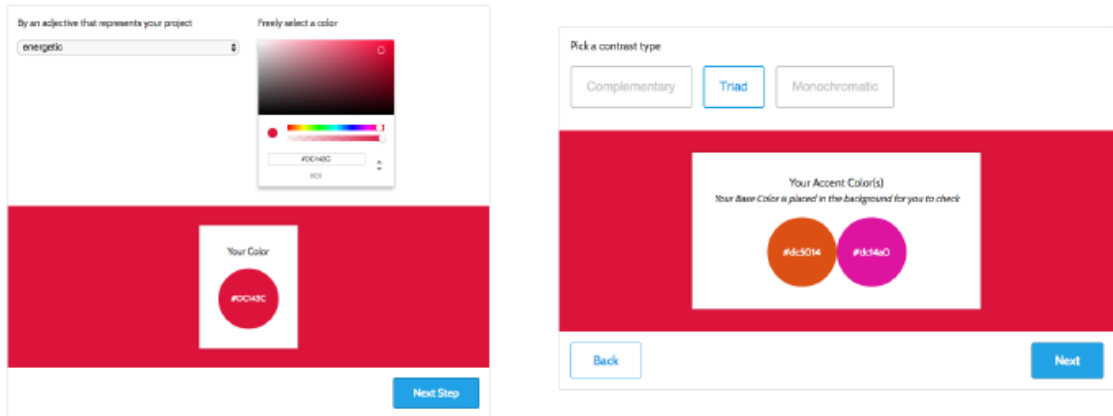


Abbildung 2.3: Vorschau der ausgewählten Farben in der Anwendung

Nutzer eine einheitlichere Nutzungserfahrung geschaffen werden konnte.

Für den Bereich Web wurde eine Möglichkeit entworfen, bei der der Nutzer Erfahrungen mit der Arbeit von Grids sammeln kann. Hier wird dem Nutzer eine Seite mit abstrakten geometrischen Elementen dargestellt, über denen ein Grid liegt. Der Nutzer kann die Werte dieses Grids in drei Bereichen verändern: Anzahl der Spalten, breite der Spalten und Abstand zwischen den Spalten. Die dargestellte Seite verändert sich zusammen mit dem Grid und zeigt dem Nutzer so, welche Auswirkung eine Veränderung des Grids haben kann.

2.5.1 Anmerkung zur Umsetzung

Da dieser Bereich der bisher komplexeste ist, konnte eine Umsetzung im Zeitrahmen nicht realisiert werden, hier konnte nur die Arbeit im Bereich Konzept und Gestaltung finalisiert werden. Vor dem Gesichtspunkt, dass es Ziel der Arbeit ist, eine Marktfähige Anwendung in form eines MVP zu erstellen, wurde sich gegen eine Implementation dieses Bereiches in einem unfertigen Zustand entscheiden.

Die begonnene Arbeit in der Entwicklung an diesem Bereich wurde aber in Form eines *Pull Requests* auf der Plattform Github veröffentlicht. Dieser soll Zeigen, dass die Anwendung auch in Zukunft weiter entwickelt werden soll und auch als Motivation für die aktive Mitarbeit von Dritten dienen.

2.6 Ergebnisse der Benutzung

Auch wenn es eines der Hauptziele der Anwendung ist, dem Nutzer während seiner Nutzung interaktiv Wissen zu vermitteln, ist die Anwendung dennoch darauf ausgelegt, den Nutzer bei der Gestaltung eines konkreten Artefaktes zu unterstützen. Hier ist es für den Nutzer hilfreich, auf die von ihm erarbeiteten Ergebnisse auch nach der Verwendung des Tools noch Zugriff zu haben.

Dieser Bereich wurde im Rahmen des Praxisprojektes nicht konzipiert, ist aber für die Wahrnehmung der Anwendung als fertiges Produkt durchaus wichtig. Eine gute Darstellung der Ergebnisse des Nutzers definiert einen ausschlaggebenden Teil der Nutzungserfahrung, da ohne diesen Schritt das Gefühl aufkommen würde, dass während der Zeit der Nutzung kein Mehrwert erwirtschaftet wurde. Im folgenden sollen also mögliche Darstellungen der Ergebnisse diskutiert und vorrangig die Frage beantwortet werden, welche Darstellungsweise den besten Kompromiss aus Umsetzbarkeit und Mehrwert für den Nutzer bietet.

Die einfachste Darstellung, die in jedem Falle gewählt werden sollte, ist eine transistente Darstellung am Ende einer Nutzung der Anwendung. Hier sollte dem Nutzer noch einmal aufgezeigt werden, welche Werte er innerhalb der Anwendung erarbeitet hat. Transistent ist diese Darstellung, weil diese zunächst nur im JavaScript-Programm im Browser des Nutzers gespeichert wird. Schließt dieser das Browserfenster, so wird auch das JavaScript-Programm beendet und die Ergebnisse gehen verloren.

Ein naheliegender Schritt ist also die Persistierung des Wissens für den Nutzer. Hierfür bieten sich verschiedene Möglichkeiten.

Denkbar wäre zum Beispiel das Speichern der Daten als Cookie³ im Browser des Nutzers. So könnten die Daten beim nächsten Besuch der Anwendung wieder angezeigt werden.

Von Nachteil ist hier, dass die Kontrolle über die Speicherung der Daten nicht explizit beim Nutzer liegt: Löscht der Browser den Cookie (zum Beispiel, weil dessen *Max-Age*⁴ Wert überschritten ist) sind die Daten ohne Eingriffsmöglichkeit des Nutzers verloren.

Eine andere Möglichkeit bietet das Entwickeln eines Backends, das eine entsprechende Datenhaltung verwalten kann. Hier könnten auch mehrere Projekte eines Nutzers gespeichert

³Ein Cookie erlaubt das Speichern eines Zustandes über das HTTP-Protokoll [Bar11]

werden, allerdings liegt der Entwicklungsaufwand für ein solche Backend außerhalb des zeitlichen Rahmens der Abschlussarbeit.

Als Kompromiss zwischen Nutzerfreundlichkeit und Entwicklungsaufwand wurde sich für die Persistierung des Wissen in einer Datei im PDF-Format entschieden, die der Nutzer Herunterladen kann.

Weitere Verbesserungen der Nutzererfahrung können im Aufbau und Inhalt der Datei erreicht werden. Optimal wäre eine Aufbereitung der Daten, sodass der Nutzer diese möglichst ohne weitere Bearbeitung in seinen Workflow übernehmen kann. Obwohl das Zielmedium des Nutzers bekannt ist, können daraus keine zweifelsfreien Rückschlüsse auf die benötigte Struktur und Form der Daten gezogen werden.

So kann beispielsweise bekannt sein, dass der Nutzer eine Webanwendung entwickelt und das Setzen seiner Texte in der CSS-Syntax vornimmt. Trotzdem kann der Nutzer zum Beispiel verschiedene Preprozessoren wie SCSS, SASS oder LESS verwenden, die jeweils eine unterschiedliche Syntax verwenden. Oder es kann bekannt sein, dass der Nutzer an einer Textseite arbeitet, jedoch nicht, welches Textsatzprogramm er verwendet⁵.

Es ist dabei durchaus Möglich, die benötigten Informationen vom Nutzer zu erhalten und die Daten in Einzelfällen entsprechend aufzubereiten, jedoch liegen auch diese Anforderungen außerhalb des Zeitlichen Rahmens dieser Abschlussarbeit.

⁴Das *Max-Age* Attribut kennzeichnet die maximale Dauer, die ein Cookie im Browser behalten wird. [Bar11]

⁵Ein weiteres Problem stellt die Aufbereitung der Daten für einen Import in ein Textsatzprogramm dar.

Kapitel 3

Technischen Grundlagen

Das nachfolgende Kapitel soll die technischen Grundlagen erläutern, die für die Umsetzung der Anwendung von Bedeutung sind. Im Folgenden sollen verschiedene Möglichkeiten der Umsetzung diskutiert und der Prozess für die Wahl der verwendeten Technologien und Frameworks erläutert werden.

Weiterhin sollen im zweiten Teil dieses Kapitels die nötigen Grundlagen für ein Verständnis des Aufbaus der Anwendung und ihres Quellcodes geschaffen werden.

3.1 Diskussion verfügbarer Technologien

Durch die in Kapitel 2 definierten Konzepte können bereits einige Anforderungen an die Anwendung gefunden werden, die für die Umsetzung von Bedeutung sind. Die Anwendung:

- weist einen hohen Grad der Interaktivität auf
- benötigt kein Backend
- soll eine Webanwendung sein

Die Anwendung entspricht somit einer *Rich Internet Application* wie sie George Lawton definiert:

RIAs feature responsive user interfaces and interactive capabilities. This makes Internet-based programs easier to use and more functional, and also overcomes

problems with traditional Web applications such as slow performance and limited interactivity [Law08]

3.1.1 Rich Internet Applications

Die Idee, Webseiten dynamischer und interaktiver zu gestalten geht dabei fast so weit zurück wie die Idee des Internet selbst.

Very early on, software engineers realized that the client-server architecture of the Web provided a powerful platform in which the browser could be a universal user interface to applications that may run locally or remotely on a server [Jaz07]

Um diese Interaktivität und Dynamik umzusetzen wurden zunächst Plattformen mit eigenen Laufzeitumgebungen genutzt. Lawton [Law08] nennt hier beispielsweise Microsoft Silverlight, Adobe Flash oder JavaFX. Die Verwendung von eigenen Laufzeitumgebungen dieser Plattformen bedeutete für den Nutzer jedoch, dass dieser bestimmte Programme auf seinem Rechner oder Plugins in seinem Browser installieren musste, um zugriff auf die so gewonnene Interaktivität zu erhalten. Somit musste auf einen der größten Vorteile der Entwicklung von Webanwendungen verzichtet werden: Um Webanwendungen nutzen zu können muss der Nutzer keine Installation der Software an seinem Endgerät durchführen.

Durch das Verabschieden von neuen Web-Standards wie HTML5 oder ECMAScript 6 können viele Funktionen der oben genannten Plattformen nativ in einem Browser abgebildet werden. So empfiehlt Adobe beispielsweise, Flash nicht für Funktionen zu nutzen, die auch mit Hilfe von HTML5 abgebildet werden können¹.

Für die Realisierung dieser Interaktivität im Browser wird vor allem JavaScript verwendet.

3.1.2 Single Page Applications

Das Prinzip der *Single Page Applications (SPA)* lässt sich bereits an der Wortbedeutung gut erkennen: Es handelt sich um Anwendungen, die auf einer einzigen HTML-Seite

¹siehe dazu <https://blogs.adobe.com/conversations/2015/11/flash-html5-and-open-web-standards.html>, zuletzt abgerufen am 11.08.2017

ausgeliefert werden. Mikowski und Powell definieren SPAs als

[...] an application delivered to the browser that doesn't reload the page during use. Like all applications, it's intended to help the user complete a task, such as "write a document" or "administer a web server." We can think of an SPA as a fat client that's loaded from a web server. [MP13]

Single Page Applications können somit als Untereinheit der *Rich Internet Applications* bezeichnet werden.

Eine der ältesten Möglichkeiten, eine Single Page Application zu realisieren stellt AJAX dar. AJAX erlaubt das nachladen von Inhalten vom Server, ohne, dass die Seite, die im Client geöffnet ist, neu geladen werden muss [Pau05].

Da eine Anforderung an die zu entwickelnde Anwendung aber die Absenz eines Backends ist, kann AJAX als Möglichkeit zur Umsetzung ausgeschlossen werden.

Aufgrund der Möglichkeiten, die mit den bereits erwähnten, neuen Web-Standards einhergehen, entstanden in den letzten Jahren jedoch viele JavaScript-Frameworks, die sich auf die Entwicklung von Single Page Applications spezialisieren.

3.1.2.1 JavaScript Single Page Application Frameworks

Aus der Menge der verfügbaren JavaScript-Frameworks, die sich für die Umsetzung einer SPA eignen, wurden hier drei für eine Untersuchung auf die Tauglichkeit für die Umsetzung der konzipierten Anwendung hin untersucht. Die Auswahl der Frameworks erfolgte dabei nach Verbreitung und tatsächlicher Verwendung.

Da die genaue Messung der Verbreitung einer Software sich als schwierig gestaltet, wurde als Indikator für die Beliebtheit und Verbreitung die Zahl der *stars*², die den verschiedenen Frameworks auf der Plattform GitHub zugeordnet sind, verwendet.

Im folgenden werden die Frameworks Vue.js, Angular.js und React.js verglichen.

Vue.js[Eva] wurde im Dezember 2013 veröffentlicht und ist mittlerweile in der Version zwei . Vue beschreibt sich selbst in der Einführung der Documentation wie folgt:

Vue [...] is a progressive framework for building user interfaces. Unlike other

²Ein *star* sagt aus, dass eine Person Interesse an dieser Software zeigt.

monolithic frameworks, Vue is designed from the ground up to be incrementally adoptable. The core library is focused on the view layer only, and is very easy to pick up and integrate with other libraries or existing projects. [JS17a]

Vue.js ist dabei (wie die anderen hier behandelten Libraries und Frameworks auch) Komponentenbasiert. Um das User Interface effektiv zu aktualisieren, verwendet es einen *Virtual DOM*. Dieser ist eine Abstraktion des DOMs, wie ihn beispielsweise das w3c spezifiziert [Rob17]:

The Document Object Model (DOM) is a programming API for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.

Das *Virtual DOM* wird von Frameworks wie Vue.js dafür verwendet, Änderungen am DOM performanter durchzuführen. Die genaue Funktionsweise sei im Rahmen dieser Arbeit nicht erläutert, jedoch beschreibt die Vue.js Dokumentation die abstrakten Vorgänge wie folgt [JS17b]:

Under the hood, Vue compiles the templates into Virtual DOM render functions. Combined with the reactivity system, Vue is able to intelligently figure out the minimal amount of components to re-render and apply the minimal amount of DOM manipulations when the app state changes.

Für das Darstellen von Inhalten verwendet Vue.js eine Templating-Engine, Erweiterungen wie Routing oder Verwaltung des Zustandes der Anwendung müssen über externe Bibliotheken eingefügt werden.

Das von Google Entwickelte **Angular.js**[Goo] ist das älteste der hier behandelten Frameworks. Es wurde zuerst im Oktober 2010 veröffentlicht.

Insgesamt ist Angular deutlich komplexer als Vue.js und React.js. Viele Erweiterungen (wie zum Beispiel das Routing), die in Vue.js und React.js durch externe Bibliotheken implementiert werden müssen sind beispielsweise in Angular.js bereits enthalten. Auch weisen Angular-Anwendungen eine deutlich Komplexere Architektur auf, die unter anderem aus Teilen wie Modulen, Komponenten und Templates besteht. Auch hier wird für die Darstellung von Inhalten also eine Templating-Engine verwendet.

React.js[Fac] wurde im Juli 2013 von Facebook veröffentlicht.

Wie Vue.js implementiert auch React.js einen Virtual DOM um Änderungen in der Benutzeroberfläche performanter abbilden zu können. Anders als Vue.js und Angular.js verwendet React jedoch keine Templating Engine, React Applikationen sind daher als komplett in JavaScript-Funktionen geschrieben, die zur Laufzeit in HTML umgewandelt werden. Ähnlich wie Vue.js ist auch React.js in seiner Funktionalität sehr auf die wesentlichen Funktionen komprimiert, Erweiterungen wie Routing oder Application State müssen auch hier über externe Bibliotheken eingebunden werden.

Aus dem Vergleich wird zunächst deutlich, dass alle Frameworks ähnliche Ziele verfolgen und somit auch alle Frameworks für die Umsetzung der hier vorliegenden Anwendung grundsätzlich geeignet sind. Angular.js wurde ausgeschlossen, da die relativ komplexe Grundstruktur für eine eher kleinere Anwendung, wie sie hier entwickelt werden soll, als unpassend erachtet wurde. Die abschließende Entscheidung fiel auf das Framework React.js, aus dem pragmatischen Grund, dass bereits im Vorfeld dieser Arbeit ein Proof of Concept entstand, der ebenfalls auf dem Framework React.js aufbaut. Hiervon wurde sich eine simpleres übertragen bestimmter Teile aus dem Proof of Concept versprochen.

3.2 Einstieg in React.js

Das Framework React.js definiert einige spezielle Konzepte und Vorgänge, auf die im Folgenden eingegangen werden soll. Der Umfang soll dabei so gering wie möglich bleiben, aber trotzdem ein Verständnis der später in dieser Arbeit gezeigten Codebeispiele und des Quellcodes auf der Begleit-CD ermöglichen.

3.2.1 Komponenten

Die komponentenbasierte Entwicklung ist einer der Hauptbestandteile der Architektur einer React-Anwendung [Gac15, S. 28].

Dieser Abschnitt beschäftigt sich mit dem Aufbau, der Verwendung, den verschiedenen Formen und den Besonderheiten von Komponenten in React.

Bevor auf die Besonderheiten von Komponenten in React.js eingegangen wird, soll im Folgenden zunächst ein Überblick über das Konzept der Komponente an sich und im

Kontext der Software-Entwicklung gegeben werden.

3.2.1.1 Was ist eine Komponente?

Als erster Schritt zu Beantwortung der Frage, was eine Komponente ist, bietet sich die Betrachtung der Wortbedeutung an. Der Duden definiert eine Komponente als "Bestandteil eines Ganzen"[Dud06]. Diese Definition gibt bereits erste Hinweise auf die Eigenschaften einer Komponente in der Software-Entwicklung.

Weitere Eigenschaften definiert Clemens Szyperski:

One thing can be stated with certainty: components are for composition. [...] Composition enables prefabricated “things” to be reused by rearranging them in ever-new composites. [Szy02]

Und De Alfredo und Henzinger schreiben über Komponenten:

It does not constrain the environment but describes the behavior of the component in an arbitrary environment: “for all inputs x and y , if $y \neq 0$, then the output is $z = x/y$ “ [DAH01]

Für den Rahmen dieser Arbeit können daraus für Komponenten also folgende Eigenschaften gefolgert werden:

Eine Komponente ist nur ein einzelner Teil der ganzen Anwendung. Dieser Teil sollte (nach Möglichkeit) auch an anderen Stellen in der Anwendung wiederverwendbar sein und sollte weiterhin keine Abhängigkeit von seiner Umwelt besitzen.

3.2.1.2 Komponenten in React.js

Die React.js-Dokumentation selbst enthält eine weitere Definition einer Komponente:

Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called props) and return React elements describing what should appear on the screen. [Inc16a]

Jede Komponente in React.js ist eine Subklasse der Basisklasse `React.Component` und besitzt einen Lebenszyklus, den sogenannten *Component Lifecycle*. Dieser besteht aus drei Zuständen [Inc16c]:

1. Mounting
2. Updating
3. Unmounting

Für jeden dieser Zustände können verschiedene Funktionen aus der Basisklasse `React.Component` überschrieben werden, um diese in der Komponente zu verwenden.

Mounting und *Unmounting* beschreiben den Anfang beziehungsweise das Ende des Lebenszyklus und Funktionen, die diesen Zuständen zugehörig sind, werden demnach nur einmal aufgerufen. Funktionen, die dem Zustand *Updating* zugehörig sind, können beliebig oft aufgerufen werden, jedes Mal, wenn ein Update der Komponente initiiert wird. Von besonderer Bedeutung ist hier die Funktion `render()`, in der definiert wird, welchen Inhalt die Komponente ausgeben soll.

Für das Arbeiten mit Komponenten in React ist weiterhin das Konzept der *props* und des *state* relevant [Inc16a].

Dabei bezeichnen *props* JavaScript-Objekte, die von einer Elternkomponente zur Kindkomponente übergeben werden können. Besonders interessant ist die Möglichkeit, Referenzen auf Funktionen übergeben zu können. Somit kann in einer Elternkomponente auf die Interaktion mit einer Kindkomponente reagiert werden.

Der *state* bezeichnet im weitesten Sinne den Zustand einer Komponente. Dieser Zustand ist ein Attribut einer Komponente, in dem Werte gespeichert werden, die für die Anzeige des Zustandes der Komponente relevant sind (zum Beispiel, welche Kindkomponente als aktiv dargestellt werden soll).

Hier zeigt sich außerdem der Grundgedanke der Interaktivität in React.js: Werden *props* oder *state* aktualisiert, werden die Funktionen des *Updating*-Zustandes im Lebenszyklus dieser Komponente aufgerufen. Somit kann die Ausgabe der Komponente auf die neuen Werte angepasst werden.

3.2.1.3 Zustandslose Komponenten

Nicht jede Komponente in React.js muss einen Zustand verwalten, viele Komponenten innerhalb einer Anwendung dienen nur der Darstellung von Inhalten, die ihnen von ihren

Elternkomponenten übergeben werden. Vor den in Kapitel 3.2.1.1 definierten Anforderungen an eine Komponente ist es sinnvoll, viele in sich abgeschlossene und in ihren Funktionen limitierte Komponenten zu implementieren, die dann von einigen wenigen Komponenten kontrolliert werden.

Da zustandslose Komponenten keine der im vorhergehenden Kapitel erwähnten Methoden des Lebenszyklus implementieren, wurde mit dem Release von React 0.14³ eine simplifizierte Schreibweise für diese Komponenten eingeführt (Listing 3.1 zeigt diese). Hier wird zum Einen das unnötige Schreiben von Code vermeiden, zum anderen kann während der Entwicklung auf den ersten Blick festgestellt werden, ob eine Komponente einen Zustand verwaltet oder nicht.

Listing 3.1: Simplifizierte Schreibweise für Komponenten ohne *state*

```
1  const Button = (props) => {  
2    return (  
3      <button onClick={props.onClick}>Click me!</button>  
4    )  
5  }  
6  
7  export default Button
```

Ein Beispiel für die Verwendung von Komponenten mit und ohne *state* innerhalb der hier geschriebenen Anwendung findet sich in Kapitel 4.3.1.

3.2.2 JSX

JSX ist eine (eigens von Facebook für die Verwendung mit React entwickelte) Syntax-Erweiterung für ECMAScript, die es Erlaubt, React-Komponenten innerhalb der `render()`-Methode ähnlich wie HTML-Tags zu deklarieren.

JSX ist dabei lediglich eine syntaktische Verschönerung der Funktion `React.createElement(component, props)` [Inc16b], die für die tatsächliche Definition von Komponenten verwendet wird. Obwohl die JSX-Syntax der HTML-Syntax ähnlich sieht, gibt es hier einige Besonderheiten, die Beachtet werden müssen. Zum einen werden die Deklarationen von React-Komponenten und regulären HTML-Tags durch unterschiedliche Schreibweisen getrennt (React-Komponenten werden Groß geschrieben). Zum anderen können hier keine Schlüsselwörter verwendet werden, die durch die ECMAScript-Spezifikation geschützt sind⁴. Dies betrifft in erste Linie

³Siehe dazu <https://facebook.github.io/react/blog/2015/09/10/react-v0.14-rc1.html>, zuletzt abgerufen am 11.08.2017

das `class`-Attribut, das in HTML für das vergeben von Klassen verwendet wird. Für die Vergabe von CSS-Klassen muss in JSX auf `className` zurück gegriffen werden.

3.3 Einstieg in Redux

Mit zunehmender Komplexität einer React-Anwendung kann das Verwalten von Zuständen unübersichtlich werden. Diese Gefahr besteht vor allem dann, wenn Zustände über verschiedene Bestandteile der Anwendung hinweg bekannt sein müssen.

Um das Verwalten des Zustandes der Anwendung an einem zentralen Ort zu ermöglichen und auf diesen Zustand strukturiert zuzugreifen und diesen zu verändern kann die Library `Redux.js`⁵ verwendet werden. Die Dokumentation der Library definiert deren Ziel wie folgt:

Redux attempts to make state mutations predictable by imposing certain restrictions on how and when updates can happen. [Red16b]

Für ein Grundverständnis der Funktionsweise der Library sind drei Konzepte von Bedeutung:

- *Store*
- *Actions*
- *Reducer*

Der *Store* ist ein JavaScript-Objekt, in dem der Zustand der Anwendung gespeichert ist. *Actions* beschreiben, welche Veränderungen am *Store* vorgenommen werden sollen. Auch diese sind einfache JavaScript-Objekte. *Reducer* definieren, auf welche Weise der Zustand der Anwendung bei Aufruf einer *Action* verändert werden soll. [Red16a]

Die Implementierung von Redux in der hier zu entwickelnden Anwendung wird in Kapitel 4.2.1 beschrieben.

⁴https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar

⁵<http://redux.js.org/>, zuletzt abgerufen am 11.08.2017

Kapitel 4

Umsetzung der Anwendung

Bei der Textgestaltung und automatischen Änderung von Abbildungsnummern, Querverweisen, Seitenzahlen, Gliederungen, Literaturhinweisen etc. bietet sich der Rückgriff auf moderne Textverarbeitungsprogramme an. Nutzen Sie diese zur besseren Lesbarkeit und Strukturierung des Textes, aber vermeiden Sie überflüssige Spielereien. Da besonders bei Textdokumenten mit eingebundenen Objekten wie Bildern, Formeln

4.1 Gestaltung

Bereits in Kapitel XZC “Relevanz” wird deutlich, welche zentrale Rolle die Gestaltung in allen Projekten einnimmt. Auch dieses Projekt ist davon nicht ausgenommen. Um eine gute Benutzbarkeit des Tools zu gewährleisten, ist eine solide Gestaltung unabdingbar. Insgesamt ist 25knots eine sehr interaktive Anwendung, in der Informationen eher Grafisch und durch Interaktionen des Benutzers übertragen werden, als Beispielsweise durch Texte. Steve Krug schreibt über die Gestaltung von Webseiten:

Die Seiten offensichtlich zu gestalten, ist wie eine gute Beleuchtung in einem Geschäft: Alles erscheint einfach besser. Die Nutzung einer Website, die uns nicht zum Nachdenken über Unwichtiges zwingt, fühlt sich mühelos an, wogegen das Kopfzerbrechen über Dinge, die uns nichts bedeuten, Energie und Enthusiasmus raubt — und Zeit. [Kru14, S. 19]

Daher wurde bei der Gestaltung großer Wert darauf gelegt, klar zu kommunizieren, welche

Bereiche interaktiv sind und welche Auswirkungen eine Interaktion mit diesen Bereichen hat. Um diese Abgrenzung zu gewährleisten, wurde ein schlichtes Farbschema verwendet, in dem nur zwei Farben, Blau und Rot, verwendet wurden, die zur Anzeige von Interaktivität dienen. Die beiden Farben haben dabei festgelegte Rollen: Blau zeigt Interaktivität innerhalb eines Schrittes der Anwendung an, Rot zeigt zwischen verschiedenen Schritten übergreifende Aktivitäten an. Abbildung 4.1 zeigt die Ausführung dieser Idee am Beispiel des Einstiegs in die Anwendung. Die blauen Elemente erlauben eine Auswahl innerhalb des Einstiegs (nächster Zwischenschritt, Auswahl eines Entwicklungsziels), der rote Button am unteren Rand erlaubt das wechseln zum nächsten Abschnitt in der Anwendung.

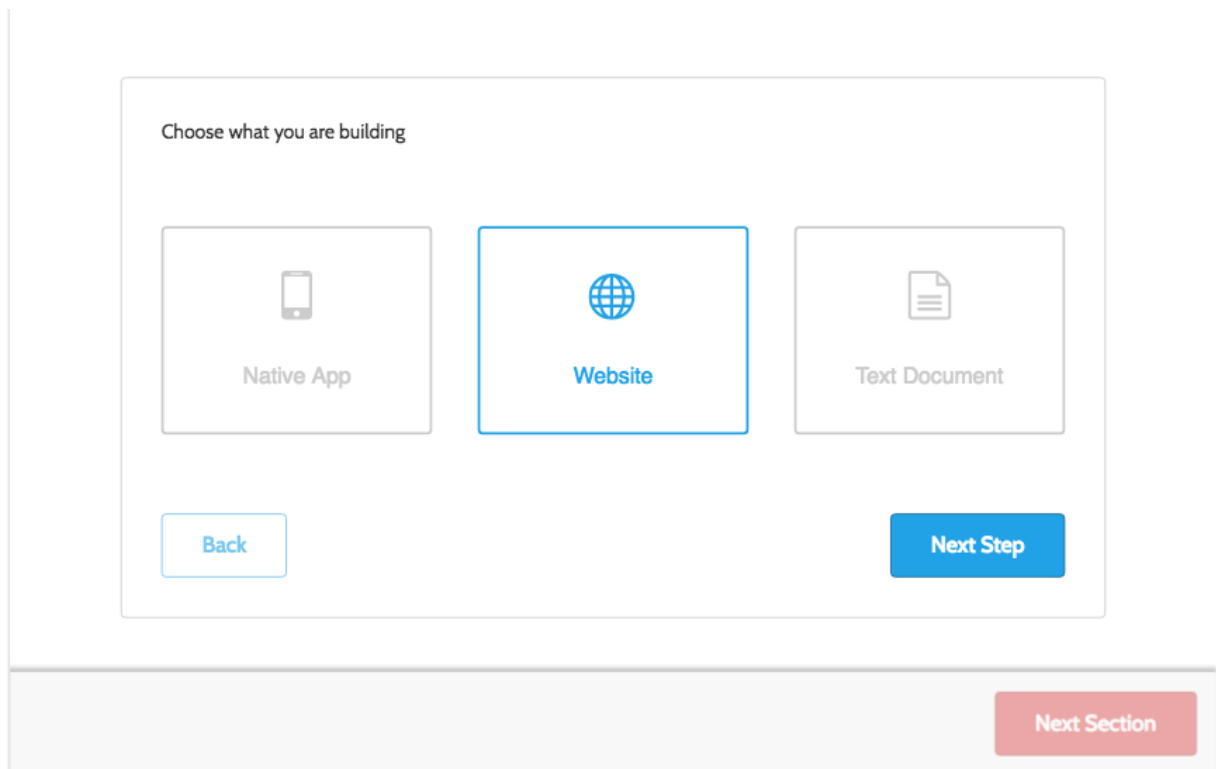


Abbildung 4.1: Farben verschiedener interaktiver Elemente

Weiterhin gilt es zu beachten, dass der Nutzer die Anwendung verwendet, um eine Gestaltung für sein Projekt zu erstellen. Der Fokus der Anwendung sollte daher darauf liegen, dem Nutzer die von ihm erarbeitete Gestaltung zu zeigen. Die Anwendung sollte sich nur in bestimmten Fällen (beispielsweise beim Auftreten eines Fehlers oder wenn eine Interaktion notwendig ist) in den Vordergrund stellen. Daher wurde darauf geachtet, die Anwendung so simpel wie möglich zu gestalten und auf unnötige gestalterische Spielereien zu verzichten. Der größte Teil der Anwendung ist in Weiß- und Grautönen gehalten, um sie die Gestaltung des Nutzers und dessen Entscheidungen besser in den Vordergrund

stellen zu können.

Dieses Vorgehen lässt sich anhand von zwei Beispielen gut verdeutlichen. Zum Einen seien hier die Fehlermeldungen im Bereich Typographie genannt. Diese sind in einem auffälligen Gelb hinterlegt, dass nur an dieser Stelle (im Sinne von: Zum anzeigen von Fehlern) verwendet wird und dem Nutzer dafür schnell auffällt (siehe Abbildung 4.2). Zum Anderen zeigt die Auswahl von Farben gut, wie Interaktive Elemente der Anwendung gleichzeitig auch die Gestaltung des Nutzer zeigen können. Die Buttons zur Auswahl einer Farbe (siehe Abbildung 4.3) bestehen hier lediglich aus der Farbe selbst, die Gestaltung der Anwendung wird hier als visuelle Zwischenebene also komplett entfernt.



Abbildung 4.2: Fehlermeldungen im Bereich Typographie

4.1.1 Vorgehen

Während der Gestaltung wurde sehr agil gearbeitet. Diese Agilität zeigt sich auf verschiedene Weise. So waren der Entwicklungsprozess (gemeint ist hier die tatsächliche Programmierung) und der Gestaltungsprozess (das Entwickeln von MockUps in einem entsprechenden Programm) nicht strikt von einander getrennt, sondern überschnitten sich. Diese Überschneidung zeigt sich sowohl in einer Langfristigen betrachten, als auch in einer kurzfristigen. Es gab nicht einen Gestaltungsprozess, nach dessen Fertigstellung die Entwicklung startete, sondern vielmehr einen Gestaltungsprozess pro Schritt der Anwendung, der umgesetzt wurde. Auch innerhalb eines Schrittes war das Vorgehen nicht linear und es wurden häufig während der Entwicklung Aspekte in der Gestaltung verändert.

Choose your Accent Color

Pick a contrast shade

a100

a200

a400

a700

Pick a contrast color

#ff1744

#f50057

#d500f9

#651fff

#3d5afe

#2979ff

#00b0ff

#00e5ff

#1de9b6

#00e676

#76ff03

#c6ff00

#ffea00

#ffc400

#ff9100

#ff3d00

Abbildung 4.3: Buttons zum Auswählen einer Akzentfarbe

Die Gründe hierfür sind vor allem eine bessere Möglichkeit zum testen der Interaktion während der Entwicklung.

Durch die in Kapitel 4.1.2 und 4.1.3 erläuterten Aspekte konnten hier auch häufig kleinere Änderungen in der Gestaltung während der Entwicklung, ohne das Anpassen von MockUps, vorgenommen werden.

4.1.2 Abstände

Um den Gedanken von innerhalb der Anwendung wiederverwendbaren Komponenten zu unterstützen liegt der Gedanke nahe, auch Abstände innerhalb der Gestaltung (und später auch in der Umsetzung) wiederverwendbar zu gestalten. Die Grundlage für diesen Gedanken lieferte ein Artikel von Nathan Curtis [Cur16]. Der Artikel enthält zwei Grundgedanken:

1. Die Größen von Abständen sollten festgelegt und ihre Anzahl übersichtlich sein.

2. Es gibt verschiedene Arten von Abständen, die die verschiedenen Größen auf unterschiedliche Weise einsetzen und kombinieren.

Diese Gedanken wurden in der Gestaltung und Umsetzung des Projektes übernommen. Zunächst wurden die verschiedenen Abstände, aufbauend auf der von Curtis empfohlenen Basisgröße von 16px, definiert. Aufbauend auf der Basisgröße wurden anschließend Abstufungen in beide Richtungen Erstellt, die nach Kleidergrößen, von XS bis XXL, benannt wurden. Diese Abstufungen wurden in einer eigenen Datei als JavaScript-Objekt deklariert, so dass über die gesamte Anwendung hinweg diese festgelegten Größen verwendet werden können.

Curtis definiert 6 verschieden Arten von Abständen (vgl. Abb. ZXC), von denen 5 innerhalb der Anwendung als eigenständige Komponenten definiert wurden. Die Komponenten nehmen die Größe des Abstandes als **prop** in einer der definierten Kleidergrößen entgegen und erzeugen ein **div**, dass die Abstände als padding oder margin anwendet. Die Pixelwerte für die jeweilige Abstandsgröße erhält die Komponente dabei durch den Aufruf des für die Abstandsgrößen zuständigen JavaScript-Objekts (zum Beispiel `spacing.1`). Alle diese Komponenten rendern außerdem die ihnen übergebenen Kinder, sodass eine Verwendung der `SpacingInset` Komponente wie in Listing 4.1 möglich wird.

Listing 4.1: Beispielhafte Verwendung einer Komponente für Abstände

```
1 <SpacingInset size='1' >
2   <h1> A Headline </h1>
3   <p> Some Text </p>
4 </SpacingInset>
```

Hier stellt sich die Frage, inwiefern es Sinn ergibt, Komponenten zu definieren, die eine ausschließlich visuelle Funktion haben. So könnte deren Funktion auch innerhalb der CSS-Regeln von anderen Komponenten definiert und so ein übersichtlicheres Markup geschaffen werden. Während der Arbeit stellte sich heraus, dass die Definition er Abstände als eigene Komponenten ein sehr einfaches Entwickeln von Interfaces ermöglichte. Durch die eingegrenzten Möglichkeiten ist auch während der Entwicklung ein testen von anderen Abständen sehr einfach möglich. Weiterhin ist der Raum für Inkonsistenzen begrenzt, da die Abstände nur in den vorgegebenen Größen angegeben werden können. Dies wurde, gerade mit Blick auf die spätere Weiterentwicklung und Veröffentlichung, als ausreichend großer Vorteil angesehen, um eine Definition als eigenständige Komponente zu rechtfertigen.

4.1.3 Styleguide

Da zu einem späteren Zeitpunkt unter Umständen verschiedene Personen an der Weiterentwicklung der Anwendung beteiligt sein werden, macht das Festhalten der bisher gestalteten Elemente und der Grundlagen der Gestaltung durchaus Sinn. Während der Gestaltung wurde nur ein minimalistischer Styleguide mit Informationen über Farben, Schriftgrößen und Abstände geführt, der für die Gestaltung dieser ersten Version mit nur einer Person im Team ausreichend war.

Mit Blick auf die spätere Weiterentwicklung durch ist ein Zentraler Ort, der einen Überblick über die bereits erstellten Komponenten gibt, von großem Vorteil. Hierfür wurde die Bibliothek Storybook¹ verwendet. Die Bibliothek läuft lokal im Browser und ermöglicht es, verschiedene Komponenten aus der Anwendung gekapselt darzustellen. Hierbei ist kein doppelter Code notwendig, die Komponenten können direkt aus dem Anwendungscode übernommen werden. Das hat außerdem den Vorteil, dass Komponenten zunächst für sich, außerhalb der Anwendung und ohne mögliche Abhängigkeiten entwickelt werden können.

4.2 Architektur

Bei der Textgestaltung und automatischen Änderung von Abbildungsnummern, Querverweisen, Seitenzahlen, Gliederungen, Literaturhinweisen etc. bietet sich der Rückgriff auf moderne Textverarbeitungsprogramme an. Nutzen Sie diese zur besseren Lesbarkeit und Strukturierung des Textes, aber vermeiden Sie überflüssige Spielereien. Da besonders bei Textdokumenten mit eingebundenen Objekten wie Bildern, Formeln

4.2.1 Redux

Im Laufe der Anwendung müssen bestimmte Informationen über die gesamte Anwendung hinweg für bestimmte Komponenten abrufbar sein. Dies betrifft vor allem, aber nicht ausschließlich die von Nutzer zu Beginn der Interaktion mit der Anwendung definierten *scopes*, die von jedem Teil der Anwendung für die korrekte Darstellung der angepassten

¹<https://github.com/storybooks/storybook>

Inhalte überprüft werden. Ein weiteres Beispiel stellt die Zusammenfassung am Ende der Interaktion mit der Anwendung dar, die Zugriff auf alle vom Nutzer definierten Werte benötigt. Der Redux-Store für die Anwendung umfasst daher Werte aus den Schritten *Intro*, *Typographie* und *Farben*, die vollständige Struktur kann Listing ZXC entnommen werden. Um die Arbeit mit dem Store zu vereinfachen und ein übermäßiges traversieren durch das Store-Objekt zu vermeiden, wurde für jeden der Bereiche ein dedizierter reducer geschrieben, der sich ausschließlich mit der Manipulation der diesem Schritt zugehörigen Werte befasst. In der Datei `ApplicationState.js` werden diese mit Hilfe der Funktion `combineReducers`, die von Redux zur Verfügung gestellt wird dann zu einem Objekt zusammengefügt (siehe Listing 4.2).

Listing 4.2: Zusammenfügen der dedizierten Reducer zu einem Objekt

```
1  const ApplicationState = combineReducers({  
2    setup: setup,  
3    typography: typography,  
4    colors: colors  
5  })
```

4.2.1.1 Beispielhaftes Verändern des Redux Store

4.2.2 CSS-Architektur

Für den Umgang mit CSS in React.js bieten sich verschiedene Möglichkeiten an. Nativ sind zwei Vorgehensweisen möglich: Anwenden von CSS über ausgelagerte Stylesheets (wie es in der Regel bei allen Webseiten gemacht wird) oder das verwenden von Inline-Styles.

Das verwenden von klassischen Stylesheets unterscheidet sich nicht sehr von der Verwendung bei einer statischen Webseite. Auch in React werden Klassennamen oder IDs festgelegt, über die dann im Stylesheet das Aussehen definiert wird (natürlich sind auch alle anderen validen CSS Selektoren anwendbar, Klassen und IDs sind hier nur als die populärsten Varianten beispielhaft genannt). Auch die Verwendung von CSS-Präprozessoren wie zum Beispiel SCSS stellt kein Problem dar, die Kompilierung dieser Dateien muss lediglich in den Build-Prozess von React.js mit eingebunden werden. Eine vereinfachte Anwendung sähe beispielsweise wie folgt aus:

```
1  <ReactComponent className='myClass'>  
2    Content  
3  </ReactComponent>
```

Die kompilierte Dom-Node wäre dabei

```
1  <ReactComponent class='myClass'>
2    Content
3  </ReactComponent>
```

Und könnte im Stylesheet über

```
1  .myClass {
2    color: red
3  }
```

Angesprochen werden. Auch die Verwendung von Methodiken wie BEM oder SMACCS ist mit diesem Ansatz ohne Probleme möglich. Für den Rahmen dieses Projektes wurde sich jedoch gegen diese Vorgehensweise entschieden, da bewusst ein Fokus auf eine komponentenbasierte Architektur gelegt werden sollte. Auch mit einer komponentenbasierten CSS-Architektur ist eine Komponente immer noch auf zwei Orte aufgeteilt: Die Funktion und das Markup, und das Styling.

[HIER LIESSE SICH AUCH DIE DISKUSSION NOCH AUSFÜHREN, DASS EIGENTLICH LEUTE LANGE DAFÜR GESPROCHEN HABEN, AUSSEHEN UND MARKUP ZU TRENNEN UND WARUM DAS HIER OKAY IST]

Um das Styling und die Funktion einer Komponente an einem Ort zu halten, bieten sich inline-styles an. Wie auch bei statischen Webseiten werden inline-styles direkt im **style**-Attribut eines Elementes definiert. In React.js werden diese als JavaScript-Objekt übergeben. Eine Beispielhafte Anwendung findet sich in Quellcode XYZ

```
1  const styles = {
2    backgroundColor: 'red',
3    fontSize: '12px'
4  }
5
6  export default function myComponent(props) {
7    return (
8      <div styles={{styles}} >
9        {props.children}
10      </div>
11    )
12  }
```

Auf den ersten Blick wirkt die Verwendung von inline-styles problematisch, vielleicht weil diese auf statischen Seiten viele Nachteile mit sich bringen. In einem Komponentenbasierten System wie React.js sind diese Nachteile jedoch nicht present. Durch die Komponentenbasierte Struktur und das damit einhergehende Ziel der Wiederverwendung von Komponenten, müssen Stylingänderungen auch hier nur an einer Stelle vorgenommen werden. Da jede Komponente nur für ihr eigenes Styling verantwortlich ist und nicht für

das von Kindern, und auch kein CSS außerhalb der inline-styles verwendet wird (abgesehen von CSS-Reset und einigen globalen Regeln wie der Schriftart), kann es zu keinen Problemen mit der Spezifität von CSS-Regeln kommen.

Allerdings weisen inline-styles einige Limitierungen auf. So können beispielsweise keine Pseudo-Elemente wie `:after` oder `:before` verwendet werden. Auch das definieren von hover-states via `:hover` wird nicht unterstützt.

[HIER NOCH VERWEIS AUF SPEZIFIKATION]

Zwar sind diese Limitierung zum aktuellen Stand des Projektes noch nicht von allzu großer Bedeutung, mit Blick auf eine Weiterentwicklung der Anwendung nach der Abschlussarbeit können diese in Zukunft jedoch eine größere Rolle spielen. Um diese Limitierungen zu umgehen, wurde das Framework Aphrodite verwendet. Das Framework erlaubt das Festlegen von styles innerhalb der Komponente ähnlich wie inline-styles, erzeugt aber für jedes neue style-objekt eine einzigartige CSS-Klasse, die via `className` angewendet wird. Die Einzigartigkeit der Klasse wird durch das hinzufügen eines Hauses am Ende des Klassennamens gewährleistet. Listing XYZ zeigt ein Beispiel

```
1  import React from 'react'
2  import { StyleSheet, css } from 'aphrodite'
3
4  function myComponent(props) {
5    <div className={css(styles.componentStyles)} >
6      {props.children}
7    </div>
8
9    const styles = StyleSheet.create({
10      componentStyles: {
11        color: 'blue'
12      }
13    })
14  }
```

Die Struktur der style-objekte ist dabei der von inline-style-objekten gleich.

Mit der Verwendung von Aphrodite können also Aussehen und Funktion von Komponenten in der gleichen Datei gehalten werden, ohne dabei den Limitierungen von inline-styles zu unterliegen.

4.3 Interessante Aspekte in der Entwicklung

4.3.1 State in Komponenten

Das Konzept des *state* in React.js wurde bereits in Kapitel 3.2.1.3 angesprochen. Hier soll an einem konkreten Beispiel verdeutlicht werden, wie der *state* genutzt werden kann, um auf Nutzereingaben zu reagieren und an welcher Stelle sich der *Component State* und der *Application State* unterscheiden. Als Beispiel wurde die Auswahl des Zielmediums durch den Nutzer im ersten Schritt der Anwendung gewählt, die in mehreren Schritten durchgeführt wird. Der Ablauf der Interaktion mit der Anwendung sieht dabei wie folgt aus: Die Anwendung präsentiert dem Nutzer drei Optionen, aus denen dieser wählen kann. Wählt der Nutzer eine der Optionen aus, gibt die Anwendung ihm eine visuelle Rückmeldung über die ausgewählte Option. Ist der Nutzer mit seiner Wahl zufrieden, bestätigt er diese durch einen Button und ihm wird der nächste Schritt im Wizard angezeigt.

Für diese Interaktion werden verschiedene Komponenten eingesetzt (vgl. Abbildung ZXC), von denen die meisten *stateless* sind, lediglich die Komponente `SetupProgress`, die in Listing 4.3 zu sehen ist², verwaltet einen Zustand.

Im *state* der Komponente `SetupProgress` wird eine Zahl gespeichert, die angibt, welche der angezeigten Optionen momentan aktiv ist (ist keine aktiv, wird der Wert auf `false` gesetzt). Beim rendern der `IconButton` Komponenten wird für jede Komponente abgeglichen, ob diese im *state* als aktive Option gespeichert ist. Jede der `IconButton`-Komponenten, die angezeigt wird, ruft über einen Callback die Funktion `handleIconButtonClick(key)` auf, wenn der User auf diese klickt und übergibt einen Key, der wiederum in den *state* der `SetupProgress` Komponente geschrieben wird. Durch das aktualisieren des *state* wird nun die `render()` Funktion der Komponente erneut aufgerufen und die entsprechende `IconButton` Komponente wird als aktiv markiert. Der `IconButton` Komponente selbst ist dabei der Kontext, in dem sie verwendet wird, nicht bekannt.

Listing 4.3: Die Komponente `SetupProgress` in gekürzter Form

```
1 class SetupProgress extends React.Component {  
2   constructor(props) {  
3     super(props)  
4   }
```

²Teile des Quellcodes, die für diesen Anwendungsfall nicht relevant sind, wurden gekürzt. Die gesamte Datei kann der beiliegenden CD entnommen werden


```

5      // Shortened for readability
6
7      this.state = {
8          activeOption: false
9      }
10 }
11
12 handleIconButtonClick(key) {
13     this.setState({
14         activeOption: key
15     })
16 }
17
18 handleButtonClick() {
19     this.props.setScope(this.state.activeOption)
20
21     // If the current setup step is the last, also set the setup state to finished
22     if (this.props.setupStep === this.props.setupSteps) {
23         this.props.setSetupToFinished()
24     }
25
26     // Reset this components' internal state to disable the button
27     this.setState({
28         activeOption: false
29     })
30 }
31
32 handleBackButtonClick() {
33     this.props.previousSetupStep()
34 }
35
36 /**
37  * Generates the main content for the setup component (i.e. IconButtons).
38  * Determines the correct subset of options and calls constructIconButtons()
39  * with that subset.
40  *
41  * @return {Array} An array of IconButtons ready for rendering
42  */
43 generateContent() {
44
45     // Shortened for readability
46
47     return this.constructIconButtons(setupOptions)
48 }
49
50 /**
51  * Constructs an array of IconButtons based on a given set of options
52  *
53  * NOTE: This will construct an IconButton for every element in the given options
54  * and does not validate them.
55  *
56  * @param {Array} setupOptions
57  * @returns {Array} An Array of iconButtons
58  */
59 constructIconButtons(setupOptions) {
60     let iconButtons = []
61     for (var i = 0; i < setupOptions.length; i++) {
62         let currentOption = setupOptions[i]
63
64         iconButtons.push(
65             <div className={css(styles.iconButtonWrapper)}>
66                 <IconButton
67                     icon={currentOption.icon}
68                     onClick={this.handleIconButtonClick}
69                     key={i}
70                     identifier={currentOption.value}
71                     active={this.state.activeOption === currentOption.value}
72                 >
73                     {currentOption.text}
74                 </IconButton>
75             </div>

```

```

76
77     )
78   }
79
80   return iconButtons
81 }
82
83 render() {
84   return (
85     <BorderedBox>
86       <SpacingInset size='1'>
87         <span>Choose what you are building</span>
88         <SpacingInset size='1' />
89         <div className={css(styles.buttonWrapperStyles)}>
90           {this.generateContent()}
91         </div>
92         <SpacingInset size='1' />
93         <div className={css(styles.buttonWrapperStyles)}>
94           <SecondaryButton inactive={this.props.setupStep < 2} onClick={this.
             handleBackButtonClick} variant='outline'>Back</SecondaryButton>
95           <SecondaryButton inactive={this.state.activeOption == false} onClick={this.
             handleButtonClick}>Next Step</SecondaryButton>
96         </div>
97       </SpacingInset>
98     </BorderedBox>
99   )
100 }
101 }
102
103 // Shortened for readability
104
105 export default SetupProgress

```

Der hier gezeigte Ablauf hätte sich auch durch die Verwendung des Redux-Stores realisieren lassen, jedoch ist die gewählte Option zunächst nicht für die gesamte Anwendung von Relevanz (so kann der Nutzer seine Auswahl zum Beispiel noch ändern). Erst, wenn der Nutzer sich durch den Klick auf den *Next*-Button auf einen Wert festlegt, wird dieser auch der ganzen Anwendung, über den Redux-Store, bekannt gemacht.

4.3.2 Dispatch von Actions

Das Hauptaugenmerk in der Entwicklung lag in diesem Schritt im setzen der Scopes für die Anwendung. Da auf die Scopes in allen späteren Teilen der Anwendung zugegriffen werden können muss wurde schnell deutlich, dass diese im Application State, also dem Redux Store gehalten werden müssen. Hier stellte sich jedoch vor allem die Fragen nach der Datenstruktur der Scopes, wobei die Möglichkeiten auf die Haltung in einem Array oder als JavaScript Objekt begrenzt wurden. Ein Array bietet dabei eine höhere Flexibilität, jedoch eine deutlich ungenauere Beschreibung der Scopes. So können andere Komponenten herausfinden, ob der für ihren Anwendungsfall relevanten Scope sich im Array der Scopes befindet, ohne Kenntnis darüber haben zu müssen, wie der Schlüssel des Scopes innerhalb

des JavaScript Objektes lautet. Auch spielt die Langer der verschiedenen Aste des Baumes bei der Verwendung des Arrays eine weniger groe Rolle, wahrend bei der Verwendung eines Objektes immer auch gepruft werden muss, ob der key im store uberhaupt vorhanden ist. Daher wurde an dieser Stelle ein Array verwendet. [HIER SPATER NOCH MAL SCHAUEN, OB DAS NOCH STIMMT]

4.3.3 Anzeige von Inhalten nach Scope

Ein Hauptaugenmerk wahrend der Entwicklung lag auf der Anwendung der vom Nutzer gewahlten *scope* in der Anwendung. In Abhangigkeit dieser Scopes muss die Anwendung verschiedene Daten presentieren. Dabei muss die Moglichkeit bestehen, diese Daten zu erweitern oder zu verandern, ohne dass die Anwendung selbst dafur umstrukturiert werden muss. Diese Datenhaltung soll hier am Beispiel der verschiedenen Schriftfamilien, die im Bereich Typographie Verwendung finden konnen, gezeigt werden.

Da die Anwendung keine Datenbank implementiert, werden diese Daten in eigenen Dateien als JavaScript-Objekte gespeichert. Listing 4.4 zeigt das Objekt, in dem die verschiedenen Schriftarten der *scopes* als Arrays gespeichert sind. Bei Betrachtung des Objektes fallt auf, dass sich Daten teilweise wiederholen. Obwohl hier gegen das Prinzip *Don't Repeat Yourself* verstoen wird, ist eine solche Struktur mit Blick auf eine Weiterentwicklung der Anwendung notig, um ein moglichst einfaches Verandern eines einzelnen *scopes* gewahrleisten zu konnen.

Listing 4.4: Aufbau des FONTS Objektes

```
1 export const FONTS = {
2   DISPLAY: [
3     'Verdana', 'Arial', 'Tahoma', 'TrebuchetMS'
4   ],
5   RESPONSIVE: [
6     'Verdana', 'Arial', 'Tahoma', 'TrebuchetMS'
7   ],
8   NOT_RESPONSIVE: [
9     'Verdana', 'Arial', 'Tahoma', 'TrebuchetMS'
10  ],
11  PAPER_DISPLAY: [
12    'Verdana', 'Arial', 'Tahoma', 'TrebuchetMS', 'Times New Roman', 'Georgia', 'Palatino'
13  ],
14  PAPER: [
15    'Times New Roman', 'Georgia', 'Palatino'
16  ],
17  ANDROID: [
18    'Roboto', 'Noto'
19  ],
20  IOS: [
21    'San Francisco'
```

```
22 |   ]
23 | }
```

Da die *keys* im **FONTS** Objekt dabei exakt den möglichen *scopes* entsprechen³, ist ein einfaches Ermitteln der benötigten Schriftfamilien, wie es in Listing 4.5 gezeigt wird, möglich.

Listing 4.5: Zugriff auf Werte des **FONTS** Objektes

```
1  determineFontFamilies() {
2    let scope = this.props.scopes[1]
3    return FONTS[scope]
4  }
```

4.3.4 Modularisierung der Anwendung

Hier ist gemeint: Im Praxisprojekt wurden mehr Komponenten gebraucht, was lief hier besser? Im Vergleich zum Praxisprojekt wurde weiterhin der Komponentenbasierte Aufbau verbessert. Während im Praxisprojekt noch 21 Komponenten für die Darstellung verwendet wurden, konnte diese Zahl im Rahmen der Abschlussarbeit auch 10 verringert werden.

[WIE KONNTE DAS ERREICHT WERDEN?]

Dies lässt sich beispielhaft an der Komponente aufzeigen, die verschiedene Möglichkeiten zur Manipulation von Attributen von Überschriften darstellt. In der Anwendung gibt es drei Überschriften verschiedener Ordnung, die in den Werten Größe, Abstand nach oben und Abstand nach unten verändert werden können. Im Praxisprojekt gab es hier für jede Überschrift verschiedener Ordnung eine eigene Komponente, im Rahmen der Abschlussarbeit konnte dies auf nur eine Komponente reduziert werden.

Der Hauptgrund hierfür ist das Auslagern der Logik in andere Bereiche der Anwendung. So werden die Fehler nicht mehr innerhalb der Komponente errechnet, sondern lediglich dargestellt. Alle anderen Werte werden der Komponente beim Aufruf aus der Elternkomponente übergeben. Die `prop area` gibt dabei an, welchen Bereich des Stores diese Komponente verändert.

```
1  <HeadlineControls
2    onChange={this.props.setValueInArea}
3    area={'headline1'}
4    title={'Headline 1'}
```

³Auch die verschiedenen *scopes* sind Konstanten, die in einem Objekt gespeichert werden.

```

5   componentErrors={errors.headline1}
6   {...this.props.headline1}
7 />

```

4.3.5 Erstellen von Farbkontrasten

Die Grundlegende Logik zum Errechnen von bestimmten Kontrasten wurde bereits im Praxisprojekt definiert. Im ersten Schritt muss die Grundfarbe hierfür in den HSL-Farbraum überführt werden. Für diese Umwandlung wurde in der Anwendung die Bibliothek `tinycolor`⁴ verwendet, die verschiedene Funktionen zur Arbeit mit Farben bereit stellt (unter anderem auch das Umwandeln in den HSL-Farbraum). Nach der Umwandlung in den HSL-Farbraum gibt die Bibliothek ein JavaScript-Objekt zurück, in dem *Hue*, *Saturation*, *Lightness* und *Alpha* als *Key-Value*-Paare vorhanden sind, mit denen die Berechnungen für die Farbkontraste vorgenommen werden können.

Für einen Komplementär-Kontrast muss der *Hue*-Wert der Grundfarbe um 180° verändert werden. Die Berechnung erwies sich mit Hilfe des HSL-JavaScript-Objektes als recht simpel, hier musste lediglich darauf geachtet werden, den einen Wert von 360 nicht zu überschreiten. Listing 4.6 zeigt die Funktion `calculateComplementary`, die diese Berechnung ausführt.

Listing 4.6: Berechnung eines Komplementär-Kontrastes

```

1  export function calculateComplementary(baseColor) {
2    let complementary = Object.assign({}, baseColor)
3    let hue = complementary.h
4
5    hue += 180
6    if (hue > 360) {
7      hue -= 360
8    }
9
10   complementary.h = hue
11   return complementary
12 }

```

Hier wurde außerdem darauf geachtet, das Farbobjekt mit Hilfe der Funktion `Object.assign()` zu kopieren, um keine ungewünschten Veränderungen im `baseColor`-Objekt zu erzeugen⁵.

Die Berechnung des triadischen Kontrastes gestaltet sich ähnlich, Hier wurde der *Hue*-Wert jedoch um 30° erhöht beziehungsweise verringert, um den gewünschten Effekt zu

⁴<https://github.com/bgrins/Tinycolor>, zuletzt abgerufen am 10.8.2017

⁵Objekte in JavaScript werden als *Referenz* übergeben, siehe dazu [Agg17]

erzielen.

Deutlich komplexer gestaltet die Generierung von monochromatischen Farbschemata, da die Farben hier in ihrem *Hue*-Wert unverändert bleiben, jedoch in ihrem *Saturation* und/oder ihrem *Lightness*-Wert verändert werden können. Weiterhin werden für dies Farbschema mehr Farben benötigt (die Anwendung arbeitet mit der Grundfarbe und drei weiteren, veränderten Farben).

Für jede Farbe müssen hier also verschiedene Entscheidungen getroffen werden. Zunächst muss entschieden werden, welche Werte verändert werden können. Möglich ist hier einer von drei Fällen:

- Nur der *Saturation*-Wert
- Nur der *Lightness*-Wert
- Sowohl der *Saturation*- als auch der *Lightness*-Wert

Um hier dynamischere Ergebnisse liefern zu können, wird diese Entscheidung in der Funktion `calculateMonochromaticColors`, die Listing 4.7 zeigt, zufällig getroffen.

Listing 4.7: Berechnung eines Monochromatischen Farbschemas

```
1  calculateMonochromaticColors(amount) {
2
3      let colors = []
4
5      for (var i = 0; i < amount; i++) {
6          let currentColor = Object.assign({}, convertToHsl(this.props.baseColor))
7          // Figure out if only one or two values should be changed
8          // Random: 0 = Lightness, 1 = Saturation, 2 = Both
9          let randomOption = Math.floor(Math.random() * 3)
10         let changedColor = this.changeValuesOfColor(randomOption, currentColor)
11
12         if (colors.length < 1) {
13             colors.push(changedColor)
14         } else {
15             let similarColors = this.checkForSimilarColors(colors, changedColor)
16             while (similarColors) {
17                 changedColor = this.changeValuesOfColor(randomOption, changedColor)
18                 similarColors = this.checkForSimilarColors(colors, changedColor)
19             }
20             colors.push(changedColor)
21         }
22     }
23
24     return colors
25 }
```

Im nächsten Schritt müssen die Veränderungen in den bestimmten Werten vorgenommen werden. Auch hier werden diese Werte zufällig gewählt, wobei die Farben nicht zu Dunkel oder zu hell sein dürfen. Diese Zuweisung findet in der Funktion `changeValuesOfColor`

statt (siehe Listing 4.8).

[HIER NOCH ERKLÄREN, WIE HELLE UND DUNKLE FARBEN AUSGESCHLOSSEN WERDEN]

Listing 4.8: Setzen der HSL-Werte

```
1  changeValuesOfColor(values, color) {
2    let manipulatedColor = Object.assign({}, color)
3    // Switch case
4    // Do the changes that need to be DropdownController
5    switch (values) {
6      case 0:
7        // change lightness
8        manipulatedColor.l = Math.random().toFixed(2)
9        break
10     case 1:
11       // change Saturation
12       manipulatedColor.s = Math.random().toFixed(2)
13       break
14     case 2:
15       // change lightning and saturation
16       manipulatedColor.l = Math.random().toFixed(2)
17       manipulatedColor.s = Math.random().toFixed(2)
18       break
19     default:
20       throw new 'Oops, seems like the randomizer messed something up.'
21   }
22
23   return manipulatedColor
24 }
```

Nachdem die Farben festgelegt sind muss außerdem überprüft werden, ob eine generierte Farbe a) zu ähnlich der Grundfarbe oder b) zu ähnlich einer anderen generierten Farbe ist. Als *zu ähnlich* zueinander wurden hier zwei Farben definiert, der *Saturation-* und *Lightness*-Werte eine Differenz kleiner als 0.1 aufweisen. Farben, die nur in einem der Beiden Werte eine zu kleine Differenz aufweisen, werden nicht als *zu ähnlich* verstanden. Die Ähnlichkeit zweier Farben wird in der Funktion `checkForSimilarColors` in Listing 4.9 deutlich. Die Funktion gibt dabei `true` zurück, wenn die beiden übergebenen Werte zu ähnlich sind. Anstatt der betroffenen Farbe wird dann eine neue generiert.

Listing 4.9: Überprüfen der Ähnlichkeit zweier Farben

```
1  checkForSimilarColors(colors, candidate) {
2    let similarValues = false
3    for (var j = 0; j < colors.length; j++) {
4      let saturationDifference = Math.abs(colors[j].s - candidate.s)
5      let lightnessDifference = Math.abs(colors[j].l - candidate.l)
6
7      if (saturationDifference < 0.1 && lightnessDifference < 0.1) {
8        similarValues = true
9      }
10   }
11
12   return similarValues
13 }
```

4.3.6 Erstellen von PDF-Dateien

Wie im Kapitel 2.1.2 “Ergebnisse der Benutzung” bereits angesprochen, dient dieser letzte Schritt der Anwendung dazu, dem Nutzer eine Zusammenfassung der Ergebnisse zu liefern. Da alle vom Nutzer getroffenen Entscheidungen im redux store gespeichert werden ist es kein Problem, hier eine Zusammenfassung dieser Daten darzustellen. Diese Daten werden dem Nutzer in der aktuellen Version nach Themengebiet aufgeschlüsselt, untereinander dargestellt. Für die aktuell implementierten Gebiete und die daraus resultierende Menge an Informationen ist diese Lösung ausreichend, jedoch muss mit weiterer Entwicklung der Anwendung hier vermutlich auch ein anderer Lösungsansatz gefunden werden.

Als Anspruchsvoller stellte sich die Implementierung der Möglichkeit heraus, die Ergebnisse auch als PDF-Datei speichern zu können. Hier bieten sich verschiedene Möglichkeiten der PDF-Generierung an. Da die Applikation momentan eine reine Client-Anwendung ist, war die Auswahl an Möglichkeiten dadurch limitiert. Hier sollte nicht nur für die Möglichkeit der PDF-Generierung auch ein Server in die Anwendung eingespielt werden.

Die simpelste der Möglichkeiten ist ein Drucken als PDF Datei. Hierbei müsste für die Seite lediglich ein entsprechendes Stylesheet hinterlegt werden, dass das Layout für den Druck anpasst. Diese Lösung ist allerdings nur beschränkt verfügbar. Das Betriebssystem macOS bieten den Druck als PDF nativ an, das Betriebssystem Windows beispielsweise aber erst seit der neusten Version, Windows 10. Eine weitere Möglichkeit stellt die Bibliothek `html2canvas`⁶ dar. Die Bibliothek erlaubt das Speichern von Seiten als Bilddateien. Die Verfügbarkeit ist hier deutlich höher als beim Drucken als PDF, jedoch bringt das Speichern als Bild einige Restriktionen mit sich. So können beispielsweise Werte nicht markiert und kopiert werden, was einen erhöhten Arbeitsaufwand für den Nutzer bedeutet.

Die Entscheidung fiel aus diesen Gründen hier auf die Bibliothek `jsPDF`⁷. Diese erlaubt das erstellen von PDF-Dateien im Browser und auch das einfügen von DOM-Elementen in PDF-Dateien. Das einfügen bringt allerdings einige Limitierungen mit sich, so werden teilweise Texte, die zu tief in DOM-Elementen verschachtelt sind, nicht dargestellt. Daher wurde die PDF per hand zusammen gesetzt. `jsPDF` bietet dafür eine API, mit der verschiedene Elemente (wie Text oder geometrische Formen), unter Angabe der Position

⁶<https://github.com/niklasvh/html2canvas>

auf der x- und y-Achse, in die Datei eingefügt werden können. Listing XZY zeigt das beispielhafte Einfügen einer Textzeile und das anschließende Speichern der Datei.

Hier besteht jedoch ein hoher manueller Aufwand, der mit potentiellen weiteren Themengebieten in der Anwendung erneut evaluiert werden muss. Abb. XZY zeigt eine beispielhafte PDF-Datei, die von der Anwendung erstellt wurde.

Hier wurde außerdem deutlich, dass eine mögliche Nomenklatur des Projektes durch den Nutzer Vorteile hätte. So könnte dieser Name auf der PDF-Datei vermerkt werden, um diese eindeutiger zuordnen zu können. Außerdem wird dem Nutzer somit eher das Gefühl vermittelt, dass er einen Prozess durchläuft, an dessen Ende er ein Ergebnis für sein Projekt erzielt hat.

⁷<https://github.com/MrRio/jsPDF>

Kapitel 5

Veröffentlichung der Anwendung

Bei der Textgestaltung und automatischen Änderung von Abbildungsnummern, Querverweisen, Seitenzahlen, Gliederungen, Literaturhinweisen etc. bietet sich der Rückgriff auf moderne Textverarbeitungsprogramme an. Nutzen Sie diese zur besseren Lesbarkeit und Strukturierung des Textes, aber vermeiden Sie überflüssige Spielereien. Da besonders bei Textdokumenten mit eingebundenen Objekten wie Bildern, Formeln

5.1 Hosting

Um den Zugang zur Anwendung für die Community möglichst einfach zu gestalten, bietet es sich an, diese zu hosten (einer andere, weniger geeignete Möglichkeit, wäre beispielsweise nur die Veröffentlichung des Quellcodes und das lokale hosten durch den Nutzer selbst). Da es sich bei der Anwendung um eine rein Clientseitige handelt, stehen eine Vielzahl von Möglichkeiten für das hosting zu Verfügung, da der Server lediglich statische html- und JavaScript-Dateien ausliefern muss.

In die nähere Auswahl kamen in diesem Projekt Github Pages und Heroku. Ein hosting auf einem privaten Server wurde schnell ausgeschlossen, um die mögliche zusätzliche Arbeit möglichst gering zu halten. Da der Quellcode der Anwendung bereits auf Github veröffentlicht wurde, scheint das Hosting über Github Pages zunächst die naheliegendste und einfachste Lösung. Ein Deployment auf Github Pages erfolgt nach entsprechenden Einstellungen im Repository einfach durch einen push auf den gh-pages branch. Die Anwendung

ist danach über die Domain `username.github.io/rpository` erreichbar. Größter Vorteil ist dabei, dass kein externer Service benötigt wird. Es treten aber auch einige Limitierungen auf. Beim testen der Hosting-Möglichkeiten wurde deutlich, dass das Modul `React-Router` den zusätzlichen Path nicht ohne weiteres unterstützt. Zwar bieten sich hier relative einfach Lösungen wie das verwenden von `hashes` anstatt von Pfaden im `React-Router` an, jedoch wurde mit blick auf die bereits bei simplen dingen auftretenden Probleme diese Möglichkeit zunächst als weniger geeignet eingestuft.

Für das Hosting wurde sich für den Service `Heroku` entschieden. `Heroku` ist ein `SaaS`, das ein `Deployment` und `hosting` ohne `Setup` und `Konfiguration` erlaubt. Ein `Deployment` auf `Heroku` läuft dabei über einen `remote branch` im `git repository`, kann also mit dem Befehl `git push heroku master` gestartet werden. `Heroku` erkennt die verwendete Sprache automatisch und stellt alles nötige automatisch ein. Jedoch unterstützt `Heroku` lediglich `Serverseitige Programmiersprachen` und bietet keinen `Support` für das `Hosting` von `statischen files`. Um diese Anwendung zu `hosten`, muss also ein `Server` geschrieben werden. Da dieser nur `statische Files` `hosten` muss, ist dieser sehr `simple` in `Node.js` und dem `Framework express` geschrieben:

```
1  const express = require('express')
2  const app = express()
3
4  app.use(express.static(__dirname + '/dist'))
5  app.listen(process.env.PORT || 8080)
```

Der `Server` `hostet` dabei den ordner `dist`, in dem sich die `gebuildeten Files` befinden (unter anderem auch die Datei `index.html`, auf die ein `Browser` automatisch zurück greift). Mit der Datei `Procfile` wird `Heroku` dann mitgeteilt, welche `Befehle` beim `Deployment` der Anwendung ausgeführt werden soll. Im Falles dieser Anwendung ist das die Datei `server.js`. Das `Procfile` besteht also lediglich aus der Zeile

```
1  web: node server.js
```

Da sowohl `Github Pages`, als auch `Heroku` von sich aus eine recht `kryptische URL` verwenden, wurde weiterhin die Domain `25knots.de` gekauft, um auch hier den `Einstieg` für `potentielle Nutzer` so einfach wie möglich zu gestalten.

5.2 Weiterentwicklung

Nachfolgend soll außerdem auf eine mögliche Weiterentwicklung der Anwendung nach dem Abschluss dieser Arbeit eingegangen werden. In diesem Zusammenhang stellen sich vor allem drei Fragen:

1. Wie können Personen aus der Community dazu gebracht werden, an einer Weiterentwicklung mitzuarbeiten?
2. Wie kann eine durchgängig hohe Qualität des Quellcodes garantiert werden, auch wenn viele verschiedene Menschen daran arbeiten?
3. Wie kann das Mitwirken für Interessenten möglichst einfach gestaltet werden?

Eine konkrete Beantwortung der ersten Frage gestaltet sich recht schwierig. Es lässt sich jedoch ein logischer Zusammenhang zwischen der Anzahl der Nutzer und der Anzahl der Mitwirkenden eines Projektes feststellen. Am Beispiel der Plattform Github können *stars* als relativ sichere Mindestzahl der Nutzer gesehen werden. Laut [BVHC15] besteht ein Zusammenhang zwischen *stars* und der Anzahl der Mitwirkenden an einem Projekt:

In git-based systems, forks are used to either propose changes to an application or as a starting point for a new project. In both cases, the number of forks can be seen as a proxy for the importance of a project in GitHub. [...] Two facts can be observed in this figure. First, there is a strong positive correlation between stars and forks (Spearman rank correlation coefficient = 0.55). Second, only a few systems have more forks than stars.

Somit ist eine Steigerung der Mitwirkenden also automatisch mit einer Steigerung der Nutzer der Anwendung verbunden. Der Inhalt dieses Abschnittes soll also vorrangig mit der Beantwortung der zweiten und dritten Frage beschäftigen.

5.2.1 Sicherung der Code-Qualität

Nach der Veröffentlichung der Anwendung ist es beabsichtigt, dass weitere Leute den Quellcode der Anwendung erweitern und bearbeiten. Hier kann nicht von einem festen Team ausgegangen werden, so ist es durchaus auch möglich, dass einzelne Personen nur

ein einziges Mal und an einem kleinen Teil der Anwendung arbeiten. Für die Anwendung ist es von Vorteil einen durchgängigen Code-Stil zu verwenden, um die Lesbarkeit durch die verschiedenen Bereiche der Anwendung zu verbessern. Personen, die allerdings nur für einen kurzen Moment Kontakt mit der Anwendung haben, kann jedoch nicht abverlangt werden, dass diese sich vorher eingehend mit den verschiedenen Code-Styles der Anwendung befassen haben. Aus diesem Grund wurde die Bibliothek `eslint` eingebunden, zur Unterstützung der Einhaltung von Coding-Styles innerhalb einer Anwendung dient. In der Datei `.eslintrc` wird dabei festgelegt, welche Conventions befolgt werden sollen. Durch das Einbinden der Bibliothek im Webpack-Build-Prozess können dann Warnung und error-messages ausgegeben werden, wenn gegen die Guidelines verstoßen wurde. Für einige Verstöße (wie zum Beispiel Fehler bei der Einrückung) lässt sich auch eine automatische Fehlerbehebung einstellen.

Auch wenn die Fehlermeldungen Webpack daran hindern, die Anwendung zu kompilieren, ist es dennoch möglich, Code auf GitHub zu pushen, der gegen die Conventions verstößt. Um hier eine weitere Sicherung einzubauen, wurden die beiden Branches `master` und `development`, die jeweils in Verbindung mit der Staging- und Production-Umgebung stehen, geschützt. In diese Branches kann also nicht mehr direkt committed werden, sondern nur noch über Pull Requests.

Für die Überprüfung von neuen Pull Request wurde der Service Travis eingebunden. Dieser service kann bei jedem neuen Pull Request bestimmte Programme ablaufen lassen und nur wenn diese keine Fehler ausgeben, kann der Pull Request gemerged werden (s. Abb ZXC). Dies bietet sich zum Beispiel für das Ausführen von Tests an, aber auch für das Testen der Code-Styles mit `eslint`.

5.2.2 Dokumentation

Grober Aufbau für diese Sektion:

1. Was zeichnet eine gute Dokumentation aus?
2. Dokumentation innerhalb des Codes
3. Dokumentation außerhalb des Codes

5.2.3 Contribution Guidelines

zahlen, Gliederungen, Literaturhinweisen etc. bietet sich der Rückgriff auf moderne Textverarbeitungsprogramme an. Nutzen Sie diese zur besseren Lesbarkeit und Strukturierung des Textes, aber vermeiden Sie überflüssige Spielereien. Da besonders bei Textdokumenten mit eingebundenen Objekten wie Bildern, Formeln

5.2.4 Tests

zahlen, Gliederungen, Literaturhinweisen etc. bietet sich der Rückgriff auf moderne Textverarbeitungsprogramme an. Nutzen Sie diese zur besseren Lesbarkeit und Strukturierung des Textes, aber vermeiden Sie überflüssige Spielereien. Da besonders bei Textdokumenten mit eingebundenen Objekten wie Bildern, Formeln

5.3 Vermarktung

Bei der Textgestaltung und automatischen Änderung von Abbildungsnummern, Querverweisen, Seitenzahlen, Gliederungen, Literaturhinweisen etc. bietet sich der Rückgriff auf moderne Textverarbeitungsprogramme an. Nutzen Sie diese zur besseren Lesbarkeit und Strukturierung des Textes, aber vermeiden Sie überflüssige Spielereien. Da besonders bei Textdokumenten mit eingebundenen Objekten wie Bildern, Formeln

Kapitel 6

Fazit & Ausblick

Bei der Textgestaltung und automatischen Änderung von Abbildungsnummern, Querverweisen, Seitenzahlen, Gliederungen, Literaturhinweisen etc. bietet sich der Rückgriff auf moderne Textverarbeitungsprogramme an. Nutzen Sie diese zur besseren Lesbarkeit und Strukturierung des Textes, aber vermeiden Sie überflüssige Spielereien. Da besonders bei Textdokumenten mit eingebundenen Objekten wie Bildern, Formeln

6.1 Zielerreichungsgrad

Zunächst lässt sich feststellen, dass im Rahmen der vorliegenden Arbeit eine Anwendung Konzipiert, Gestaltet und Umgesetzt werden konnte, die nach den in Kapitel 1 definierten Maßstäben eine Marktfähige Anwendung ist. Die Anwendung biete, sowohl durch ihre Gestaltung, als auch durch ihre Programmierung eine befriedigende Nutzererfahrung. Dem Nutzer gegenüber wird deutlich kommuniziert, an welchem Punkt der Anwendung er sich befindet und er erhält einen Mehrwert aus der Verwendung der Anwendung. Durch das Hosting und die damit verbunden Domain ist die Anwendung weiterhin für Nutzer leicht zugänglich. Außerdem ist sie auch für Nutzer, die an einer Mitarbeit an der Anwendung interessiert sind durch das öffentliche Repository und die darin erläuterten Wege zur Mitarbeit leicht zugänglich. Weiterhin ist die Anwendung eine ersten Gruppe von Personen bekannt. Hier fällt es schwer, den Grad der Zielerreichung festzulegen, da es sich zum Einen schwierig gestaltet, die Bekanntheit zu messen, die Bekanntmachung der Anwendung zum Anderen aber auch ein andauernder Prozess ist.

Auch die Mitarbeit an der Anwendung durch die Community lässt sich zu diesem Zeitpunkt noch nicht beurteilen, da die Anwendung etwa Zeitgleich mit Fertigstellung dieser Arbeit veröffentlicht wurde. Es lässt sich jedoch feststellen, dass die Grundsteine für eine mögliche Mitarbeit von anderen gelegt wurden.

An dieser Stelle sei außerdem noch einmal der geplante Umfang der Anwendung angesprochen. Hier wurde der Bereich “Layouts & Grid” zwar zu Beginn als wünschenswert definiert, jedoch konnte dieser in der ersten Version der Anwendung (wie in Kapitel 3.6 ausgeführt) nicht implementiert werden. Dies ist vor dem Gedanken, dass hier ein MVP erstellt werden sollte jedoch hinnehmbar.

6.2 Ausblick

Wie bei der Entwicklung jeder Anwendung ist es auch hier schwer, an einem bestimmten Punkt von einer “fertigen Anwendung” zu sprechen. Es konnte eine Anwendung erstellt werden, die gemessen am Rahmen der Arbeit und dem vorgegebenen Arbeitsaufwand als fertig bezeichnet werden kann. Für sich betrachtet, bieten sich noch viele Möglichkeiten, die Anwendung zu erweitern und in ihrem aktuellen Stand zu verbessern. Einige potentielle Verbesserungen des aktuellen Standes wurden bereits in den zugehörigen Kapiteln angesprochen. Diese wurden als Issues mit in der Github Repository übernommen, um so bereits erste Schritte für die Weiterentwicklung zu gehen.

Weiterhin bieten sich Erweiterungen der Anwendung in Form von neuen Themengebieten an. Diese können die in Kapitel 1 definierten sein, jedoch können durchaus auch komplett neue, noch gar nicht bedachte Themengebiete auftreten. Ein weiterer, interessanter Punkt ist die intensivere Nutzung der Scopes. Diese haben momentan nur sehr subtile Auswirkungen, für jeden Anwendungsfall durchläuft der Nutzer jedoch die gleiche Anwendung, die sich nur in Details unterscheidet. Mit weiteren Anwendungsgebieten kann auch hier eine erhöhte Diversität der verschiedenen Scopes angenommen werden.

Generell lässt sich die Wahl von React.js zur Umsetzung der Anwendung auch weiterhin vertreten. Die Library bietet zum einen die nötigen Möglichkeiten zur Erweiterung der Anwendung, zum anderen aber auch das Potential, auf Dauer als interessantes Projekt zu wirken.

6.3 Fazit

Für mich persönlich war die Arbeit an dem Projekt im Rahmen dieser Arbeit sehr spannend. Hier konnte ich viele Rollen bekleiden, die in einem größeren Projekt wahrscheinlich von verschiedenen Personen bekleidet werden und so die verschiedenen Aufgabenbereiche kennen lernen. Das Ökosystem um React.js war dabei ein sehr spannendes. Man merkt diesem das Junge Alter der Library dabei an. Viele Best Practices ändern sich häufig und sind nicht so sehr verinnerlicht wie in anderen, älteren Libraries wie beispielsweise Ruby on Rails. Häufig kann man dabei mitverfolgen, wie sich Best Practices im Rahmen von Diskussionen auf Seiten wie Github herausstellen. Weiterhin werden recht häufig neue Versionen von Bibliotheken veröffentlicht, die dafür sorgen, dass das Ökosystem sehr lebendig bleibt.

Neu war für mich persönlich auch die Situation, ein Produkt mit dem Hintergedanken zu entwickeln, dass dieses nicht nur jemand nutzen soll, sondern auch jemand komplett fremdes Einblick in den Code bekommen und an diesem mitarbeiten könnte.

Insgesamt war die Arbeit an diesem Projekt eine sehr spannende, die sicherlich auch außerhalb des Rahmens dieser Arbeit noch weiter gehen wird.

[HIER VIELLEICHT NOCH COOLES ZITAT]

Literaturverzeichnis

- [Agg17] Arnav Aggarwal. Explaining value vs. reference in javascript, 2017. <https://codeburst.io/explaining-value-vs-reference-in-javascript-647a975e12a0>, zuletzt abgerufen am 10.08.2017.
- [Bar11] A. Barth. Http state management mechanism, 2011. <https://tools.ietf.org/html/rfc6265#page-20>.
- [BVHC15] Hudson Borges, Marco Tulio Valente, Andre Hora, and Jailton Coelho. On the popularity of github applications: A preliminary note. *arXiv preprint arXiv:1507.00604*, 2015.
- [CP96] Arnold Campbell and Susan Pisterman. A fitting approach to interactive service design: The importance of emotional needs. *Design Management Journal (Former Series)*, 7(4):10–14, 1996.
- [Cur16] Nathan Curtis. Space in design systems, 2016. <https://medium.com/eightshapes-llc/space-in-design-systems-188bcbae0d62> Abgerufen am 08.08.2017.
- [DAH01] Luca De Alfaro and Thomas A Henzinger. Interface theories for component-based design. In *EMSOFT*, volume 1, pages 148–165. Springer, 2001.
- [Dud06] Wissenschaftlicher Rat Dudenredaktion, editor. *Der Duden in 12 Bänden. Das Standardwerk zur deutschen Sprache / Die deutsche Rechtschreibung: Das umfassende Standardwerk auf der Grundlage der neuen amtlichen Regeln*. Bibliographisches Institut, 24., völlig neu bearb. u. erw. edition, 2006.

- [Eva] Evan You. Vue.js. Version 2.4.2, abgerufen über <https://vuejs.org/> am 11.08.2017.
- [Fac] Facebook Inc. React.js. Version 15.6.1, abgerufen über <https://facebook.github.io/react/> am 11.08.2017.
- [Gac15] Cory Gackenheimer. *Introduction to React*. Apress, 1st ed. edition, 9 2015.
- [Goo] Google Inc. Angular.js. Version 4.3.4, abgerufen über <https://angularjs.org/> am 11.08.2017.
- [Goo09] Kim Goodwin. *Designing for the Digital Age: How to Create Human-Centered Products and Services*. John Wiley & Sons, 1 edition, 3 2009.
- [Inc16a] Facebook Inc. Components and props, 2016. <https://facebook.github.io/react/docs/components-and-props.html>, zuletzt abgerufen am 11.08.2017.
- [Inc16b] Facebook Inc. Jsx in depth, 2016. <https://facebook.github.io/react/docs/jsx-in-depth.html>.
- [Inc16c] Facebook Inc. React.component, 2016. <https://facebook.github.io/react/docs/react-component.html>.
- [Jaz07] Mehdi Jazayeri. Some trends in web application development. In *2007 Future of Software Engineering*, FOSE '07, pages 199–213, Washington, DC, USA, 2007. IEEE Computer Society.
- [JS17a] Vue JS. Introduction - vue.js, 2017. <https://vuejs.org/v2/guide/>.
- [JS17b] Vue JS. Template syntax - vue.js, 2017. <https://vuejs.org/v2/guide/syntax.html>.
- [Kru14] Steve Krug. *Don't make me think!: Web Usability: Das intuitive Web (mitp Business)*. mitp, 3., überarbeitete auflage 2014 edition, 10 2014.
- [Law08] George Lawton. New ways to build rich internet applications. *Computer*, 41(8), 2008.
- [LFDB06] Gitte Lindgaard, Gary Fernandes, Cathy Dudek, and Judith Brown. Attention web designers: You have 50 milliseconds to make a good first impression! *Behaviour & information technology*, 25(2):115–126, 2006.

- [MP13] Michael Mikowski and Josh Powell. *Single Page Web Applications: JavaScript end-to-end*. Manning Publications, 1 edition, 9 2013. Abgerufen über <https://livebook.manning.com/#!/book/single-page-web-applications/chapter-1/> am 11.08.2017.
- [Nic98] Raymond S Nickerson. Confirmation bias: A ubiquitous phenomenon in many guises. *Review of general psychology*, 2(2):175, 1998.
- [Pau05] Linda Dailey Paulson. Building rich web applications with ajax. *Computer*, 38(10):14–17, 2005.
- [Pop16] Christian Poplawski. Ermittlung relevanter themengebiete für die entwicklung eines tools zur unterstützung beim erstellen von gestaltungslösungen im hochschulkontext. 2016.
- [Red16a] Redux. Core concepts - redux, 2016. <http://redux.js.org/docs/introduction/CoreConcepts.html>, zuletzt abgerufen am 11.08.2017.
- [Red16b] Redux. Motivation - redux, 2016. <http://redux.js.org/docs/introduction/Motivation.html>, zuletzt abgerufen am 11.08.2017.
- [Rob17] Jonathan Robie. What is the document object model?, 2017. <https://www.w3.org/TR/WD-DOM/introduction.html>.
- [Szy02] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming (2nd Edition)*. Addison-Wesley Professional, 2 edition, 11 2002.
- [TCKS06] Noam Tractinsky, Avivit Cokhavi, Moti Kirschenbaum, and Tal Sharfi. Evaluating the consistency of immediate aesthetic perceptions of web pages. *International journal of human-computer studies*, 64(11):1071–1083, 2006.
- [TKI00] Noam Tractinsky, Adi S Katz, and Dror Ikar. What is beautiful is usable. *Interacting with computers*, 13(2):127–145, 2000.
- [Wri17] Angela Wright. Psychological properties of colours, 2017. <http://www.colour-affects.co.uk/psychological-properties-of-colours>, zuletzt Besucht am 10.08.2017.