Technology
Arts Sciences
TH Köln

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

Abbildungsverzeichnis

Kapitel 1

Einleitung

Die nachfolgende Arbeit beschäftigt sich mit der Entwicklung der Webanwendung *25Knots* zu einem Markfähigen Produkt, basierend auf bereits im Praxisprojekt erarbeiteten Konzepten und einem Proof of Concept.
Ziel ist dabei nicht nur, die Verwendung durch die Community[1] 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

[1] Als Teil der *Community* wird hier jede Person gesehen, die ein Interessen 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 das Anwendungscodes). Weitere details zu diesem Thema werden im Kapitel *Hosting* auf Seite 47 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 *Medieninfromatik 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 Github[2]. 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 währen 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 Umsatzung 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 Themengebeite von Bedeutung sind findet sich in Abbildung 2.1. Für den weiteren Verlauf dieses Dokumentes sollen die jeweiligen Zielmedien als *scopes* bezeichnet werden.

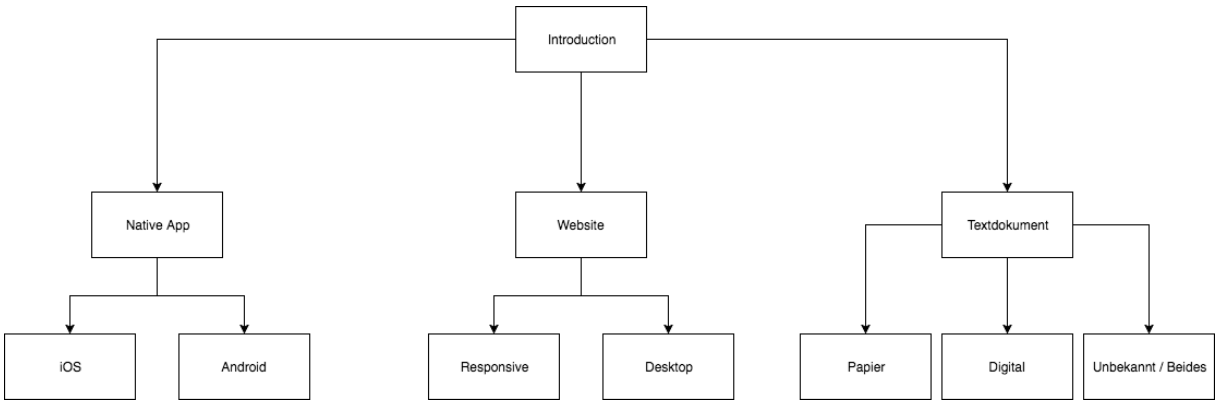

Abbildung 2.1: Zielmedien der Nutzer und deren Unterkategorien

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 Arbeit[1] 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 Wizard[2] durchlaufen, der maximal drei Auswahlmöglichkeiten zur gleichen Zeit darstellt. Zur weitern 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.

[1] Als 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]

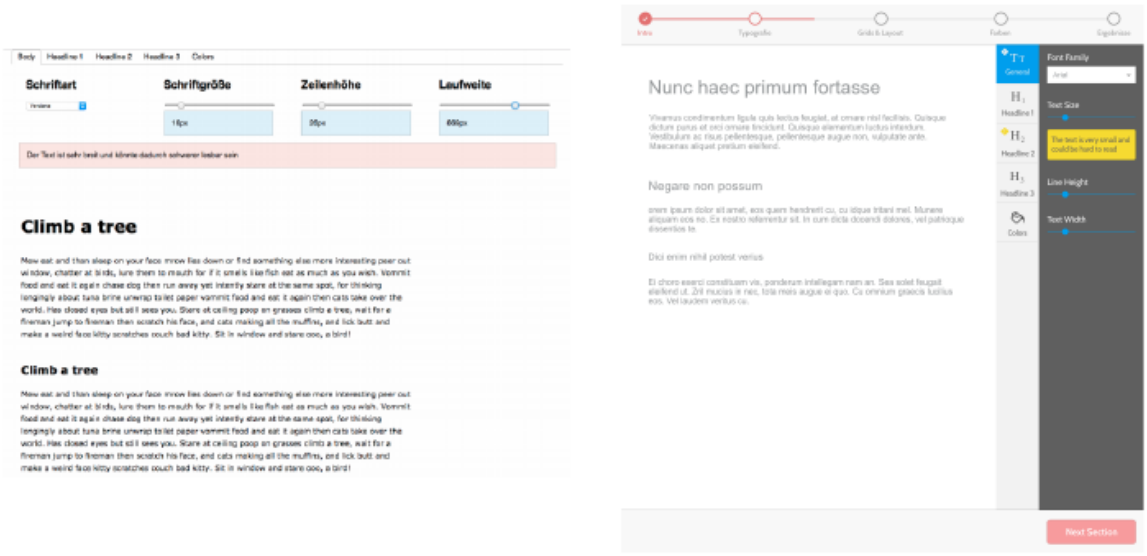

Abbildung 2.2: Der Bereich Typographie, im Praxisprojekt und der Abschlussarbeit

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 Schiebereglern, 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 zu 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, währen 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

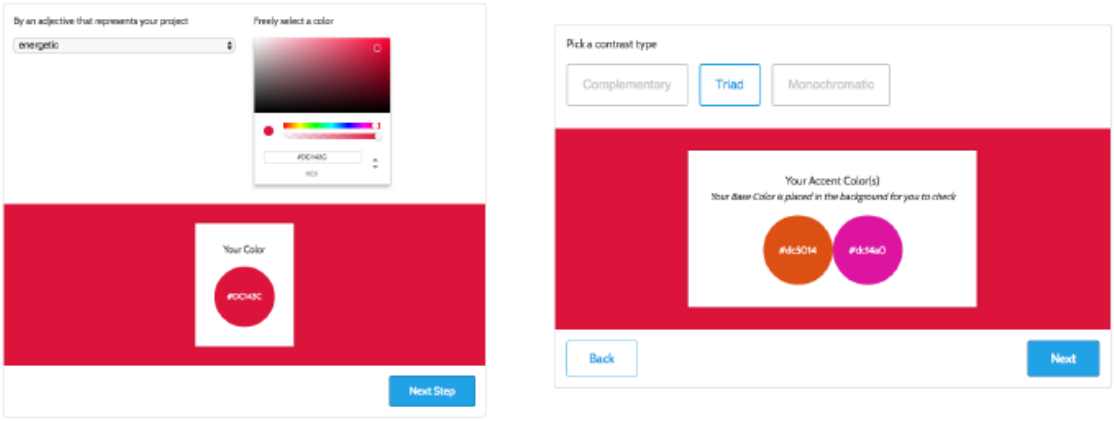

Abbildung 2.3: Vorschau der ausgewählten Farben in der Anwendung

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 Cookie[3] 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 Entwicklen eine Backends, das eine entsprechende Datenhaltung verwalten kann. Hier könnten auch mehrere Projekte eines Nutzer gespeichert

[3] Ein 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 verwendet[5].
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.

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

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

Kapitel 3

Theoretische Grundlagen

Das nachfolgende Kapitel soll die theoretischen Grundlagen erläutern, die für die Umsetzung der Anwendung von Bedeutung sind. Diese Grundlagen teilen sich in verschiedene Bereiche auf. Zum einen soll der Aufbau der Anwendung festgelegt werden. In diesem Beriech müssen außerdem einige Bereich noch Konzipiert werden. Auch dieser Prozess und dessen Ergebnisse sollen hier erläutert werden. Weiterhin gilt es, die verschiedenen technischen Möglichkeiten der Umsetzung zu diskutieren. Abschließend findet sich ein konzeptioneller Einstieg in React.js und in einige damit verbundene und wichtige Bibliotheken, sowie eine Darstellung der Design-Prozesse. **Die Einleitung stimmt nicht mehr**

3.1 Diskussion verfügbarer Technologien

Im Nachfolgenden Kapitel sollen mögliche Technologien diskutiert werden, die für die Umsetzung der Abschlussarbeit genutzt werden können. Für die Umsetzung einer Webanwendung bieten sich eine Vielzahl von Programmiersprachen und Frameworks an. Diese können durch die geplante Struktur der Anwendung jedoch bereits weiter eingegrenzt werden. Da die Anwendung keine persistente Datenhaltung implementiert und auch keine übermäßig aufwendigen Berechnungen durchgeführt werden müssen, kann auch ein klassisches Client-Server-Modell verzichtet werden. Aufgrund des hohen Grades der Interaktivität der Anwendung kann außerdem davon ausgegangen werden, dass in jedem Falle auf JavaScript zurück gegriffen werden muss.

In den letzten Jahren wurden viele JavaScript-Frontend-Frameworks veröffentlicht, die genau auf die Anforderung der gestiegenen Interaktivität im Browser reagieren. Im Praxisprojekt wurde bereits versucht, den Proof of Concept nur mithilfe des Frameworks jQuery umzusetzen. Hier wurde schnell deutlich, dass die Komplexität der Anwendung den Rahmen von jQuery übersteigt.

Im folgenden sollen also einige der bekanntesten JavaScript-Frontend-Frameworks verglichen werden. Die Auswahl dieser Frameworks erfolgt nach Beliebtheit auf GitHub.com. Weiterhin wurden Frameworks ausgeschlossen, die ein komplettes MCV-Pattern implementieren, da zum aktuellen Stand dieser Anwendung keine Models angemacht sind.

3.1.1 Vue.js

Zuerst wurde Vue.js im Dezember 2013 veröffentlicht. Mittlerweile ist Vue.js in der zweiten Version zugänglich. Vue beschreibt sich selbst in der Einführung der Documentation wie folgt:

> Vue [...] is a progressive framework for building user interfaces. Unlike other 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 eine Virtual DOM. Dieser ist eine Abstraktion des Normalen DOMs, wie ihn 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.

Der DOM im klassischen Sinn ist also eine Baumdarstellung der Elemente, die sich beispielsweise in einer HTML-Datei wiederfinden. Abb. XYZ zeigt eine beispielhafte Darstellung einer HTML-Datei im Browser, die Struktur ihrer Elemente und die Darstellung als DOM-Baum. Der Virtual DOM wird von Frameworks wie Vue.js dafür verwendet, Änderungen am DOM performances durchführen zu können. Die genaue Funktionsweise

sei im Rahmen dieser Arbeit nicht erläutert, jedoch beschreibt die Vue.js Dokumentation die 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.

3.1.2 Angular.js

Das von Google Entwickelte Angular.js ist das älteste der hier behandelten Frameworks. Es wurde zuerst im Oktober 2010 veröffentlicht und ist ebenfalls in der 2. Version angelangt.
Insgesamt ist Angular deutlich komplexer als Vue.js und React. Dies zeigt sich zum Einen beispielsweise durch die inkludierung von Routing (wenn auch nicht im angular core package), zum Anderen aber auch die die deutlich Komplexere Architektur, die unter anderem aus Teilen wie Modulen, Komponenten und Templates besteht. Auch hier wird also eine Templating-Engine verwendet.

3.1.3 React.js

React.js wurde im Juli 2013 von Facebook veröffentlicht.
Wie Vue.js verwendet auch React.js einen Virtual DOM. Auch wenn die Implementierung sich hier unterscheiden sollte, ist das Prinzip das Gleiche. Anders als Vue.js und Angluar.js verwendet React keine Templating Engine, React Applikationen sind daher als komplett in JavaScript geschrieben. Hier wird also HTML in JavaScript geschrieben, und nicht JavaScript in HTML, wie es im klassischen Sinne geschieht. Ähnlich wie Vue.js ist auch React.js in seiner Funktionalität sehr komprimiert, Funktionen wie Routing oder Application State müssen also über externe Bibliotheken eingebunden werden.

3.2 Einstieg in React.js

Der Folgende Abschnitt soll einen überblick über wichtige Konzepte und Vorgehensweisen bei der Entwicklung mit React.js geben. Ziel soll es dabei sein, genug Wissen zu vermitteln, um die später in dieser Arbeit gezeigten Codebeispiele verstehen zu können.
Dieser Abschnitt wird auf das Konzept von Komponenten im Allgemeinen sowie deren Verwendung in und in React.js eingehen und einige Grundlagen in der Auszeichnungssprache JSX vermitteln. Weiterhin werden die Übertragung von Daten und Zuständen innerhalb von React Komponenten behandelt.

3.2.1 Komponenten

Der Komponentenbasierte Aufbau ist eines der Hauptaugenmerke von React.js. Komponenten werden auf der offiziellen React.sj-Website [1]als der wichtigsten Merkmale aufgeführt und Gackenheimer [Gac15, S. 28] nennt sie einen der Hauptbestandteile einer React-Anwendung.
Dieser Abschnitt beschäftigt sich mit dem Aufbau, der Verwendung, den verschiedenen Formen und den Besonderheiten von Komponenten in React.
Natürlich kann und soll es nicht Ziel sein, einen umfassenden überblick über alle Informationen zu geben, die mit Komponenten in React in Verbindung stehen. Daher soll im Folgenden vorrangig auf in der Umsetzung als am relevantesten empfundene Aspekte eingegangen werden.

3.2.1.1 Was ist eine Komponente?

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

Das Oxford Dictionary definiert eine Komponente als

> A part or element of a larger whole [...]

Gerade im Bezug auf Softwareentwicklung werden Komponenten aber noch einige weitere

[1] `https://facebook.github.io/react/`

Eigenschaften zugeschrieben. So schreiben [Szy02] über Komponenten in der Softwareentwicklung:

> 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.

Das Hauptaugenmerk bei der Entwicklung einer Komponente in der Softwareentwicklung sollte also auf der Möglichkeit der Wiederverwendbarkeit innerhalb der Anwendung liegen. Dabei sollte der Ort für die Wiederverwendung keine Rolle spielen. Eine Komponente sollte also unabhängig von ihrer Umgebung die gleichen Ergebnisse liefern [DAH01]:

> 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 $z = x/y$"

3.2.1.2 Komponenten in React.js

Jede Komponente in React ist eine Subklasse der Basisklasse `React.Component` [Inc16b]. Konzeptuell besitzt jede Komponente in React einen Lifecycle. Die offizielle Dokumentation [Inc16b] beschreibt in diesem Lifecycle drei Zustände:

1. Mounting
2. Updating
3. Unmounting

Mounting beschreibt dabei den Vorgang des erstellen der Komponente und des einfügens in den DOM. Funktionen, die beim Mounting aufgerufen werden, werden also nur ein einziges mal im gesamten Lifecycle aufgerufen (die Ausnahme bildet dabei die Funktion `render()`, die auch beim Update Ereignis aufgerufen wird).

Ein **Update** wird durch die Veränderung von `props` oder `state` herbeigeführt. Der Auslöser für ein Update kann dabei also sowohl von der Komponente selbst, als auch von einer anderen Komponente kommen, die diese Komponente aufgerufen hat.

Unmounting wird unmittelbar vor dem entfernen der Komponente aus dem DOM aufgerufen.

Abbildung XYZ zeigt eine Schematische Darstellung des Lifecycle einer React Komponente.

Für jeden dieser verschiedenen Zustände bietet die Basisklasse `React.Component` verschiedene Funktionen an, die überschreiben werden können um diese zu verwenden. Die Funktionen werden dabei entweder vor oder nach dem jeweiligen Ereignis im Lifecycle aufgerufen. Die einzelnen Funktionen sollen im Folgenden nicht im einzelnen erläutert werden, da diese ausreichend Dokumentiert sind.

Für das Arbeiten mit Komponenten in React ist weiterhin das Konzept der `props` und des `state` relevant. `props` sind Daten, die von einer Komponente zur anderen übergeben werden können. Diese können beliebige JavaScript Objekte oder Primitive sein, somit können auch Referenzen auf Funktionen and Kind-Komponenten übergeben werden. Die Kind-Komponente kann dann via `this.props.propName` auf die ihr mitgegebenen props zugreifen. Listing XZY verdeutlich dieses Prinzip

```
1  class Parent extends REact.Component {
2    calculateSomething() {
3      // Claculates something
4      return result
5    }
6
7    render() {
8      let result = this.calculateSomething()
9
10     return (
11       <Child value={result} />
12     )
13   }
14 }
15
16 class Child extends React.Component {
17   render() {
18     return (
19       <div>{this.props.value}</div>
20     )
21   }
22 }
```

Das Prinzip des `state` wird im Kapitel Stateless & Stateful auf Seite 25 näher erläutert.

3.2.1.3 Stateful & Stateless

Komponenten in React.js können entweder Stateful oder Stateless sein. Wie der Name bereits vermuten lässt, haben diese Komponenten entweder einen state, oder nicht.
Der state in React.js beschreibt den Zustand eine Komponente. Hier findet sich auch eines der Hauptfeatures von React.js: Ändert sich der state einer Komponente, so wird

diese Komponente neu gerendert, also auch das User Interface aktualisiert um den neuen Zustand darzustellen. Der state kann (und sollte) dabei durch das Aufrufen der Funktion `setState()` geändert werden. Das nachfolgende Codebeispiel zeigt eine simple Komponente, die clicks auf einen Button zählt:

```
1  class ClickCounter extends React.Component{
2    constructor(props) {
3      super(props)
4
5      this.state = {clicks: 0}
6    }
7
8    handleClick() {
9      this.setState((prevState) => {
10         return {clicks: prevState.clicks + 1};
11     });
12   }
13
14   render() {
15     return (
16       <button onClick={this.handleClick}>Click me!</button>
17       {this.state.clicks}
18     )
19   }
20 }
```

Hier passiert folgendes: Im Konstruktor der Komponente (der nur ein mal, beim initialen rendern aufgerufen wird) wird die initiale Anzahl der Clicks im state auf 0 gesetzt. Die Komponente rendert ein `<buttton>` element und die Anzahl der gezahlten Klicks. Wird der Button geklickt, wird die Funktion `handleClick()` in der Komponente aufgerufen. Diese Funktion erhöht die Anzahl der Klicks im state um eins. Durch den aktualisierten state wird auch die Komponente neu gerendert, so dass nun auch die neue Klickzahl angezeigt wird.

Die Komponente im obigen Beispiel ist also Stateful. Wie schon früher in diesem Kapitel erwähnt, ist es aber Ziel der Komponentenbasierten Softwareentwicklung, wiederverwendbare Komponenten zu schreiben.
Diese Komponente ließe sich an jeder Stelle der Anwendung wiederverwenden, an der Klicks über einen Button gezählt werden müssen, jedoch ist die Wahrscheinlichkeit hoch, dass diese Anwendungsfall eher selten vorkommt. Außerdem ist die Wahrscheinlichkeit recht hoch, dass ein Button oder ein Element zur Darstellung von Werten in der Anwendung häufiger verwendet werden (natürlich würde eine einfache Darstellung wie im obigen Beispiel kein Sinn ergeben, für dieses Beispiel soll daher eine komplexere Darstellung der Daten angenommen werde, beispielsweise durch eine Progress-Bar). Eigentlich liegen hier

also drei Komponenten vor: Eine Komponente, die Logik enthält und zwei weitere, die lediglich Daten darstellen, somit also stateless sind.

Mit dem release von React 0.14[2]wurde eine simplifizierte Schreibweise für stateless components eingeführt. Die beiden Komponente aus dem Beispiel könnten somit wie folgt definiert werden:

```
export default function Button = (props) => {
  return (
    <button onClick={props.onClick}>Click me!</button>
  )
}
```

und

```
export default function ValueDisplay = (props) => {
  return (
    <div>{props.value}</div>
  )
}
```

Die Werte für diese Komponenten werden ihnen in der Oberkomponente als `props` übergeben:

```
class ClickCounter extends React.Component{
  constructor(props) {
    super(props)

    this.state = {clicks: 0}
  }

  handleClick() {
    this.setState((prevState) => {
        return {clicks: prevState.clicks + 1};
    });
  }

  render() {
    return (
      <Button onClick={this.handleClick} />
      <ValueDisplay value={this.state.clicks} />
    )
  }
}
```

Ziel muss es also sein, so viele Komponenten wie möglich so klein wie möglich, optimaler Weise mit nur einer einzigen Aufgabe zu schreiben, um die Wiederverwendbarkeit zu erhöhen.

Hier irgendwo eine Referenz auf Beispielhafte Verwendung von Komponenten in der Anwendung

[2] https://facebook.github.io/react/blog/2015/09/10/react-v0.14-rc1.html

3.2.2 JSX

Hier soll nicht tiefer auf die Kompilierung und Prozess von JSX eigegangen werden, jedoch ist JSX ein elementarer Teil von React-Anwendung und sei der Vollständigkeit halber hier erwähnt. Im folgenden soll vorrangig auf die Unterschiede zwischen JSX und HTML und die Besonderheiten und Pitfalls während der Entwicklung mit JSX eingegangen werden.

JSX ist eine (eigens für die Verwendung mit React entwickelte) Syntax-Erweiterung für JavaScript, die es Erlaubt, HTML-Ähnliche Tags in der `render()`-Methode von React Komponenten zu verwenden (der Einsatz von JSX konnte bereits in den vorhergehenden Beispielen beobachtet werden). JSX ist dabei lediglich eine syntaktische Verschönerung der Funktion `React.createElement(component, props, ...children)` [Inc16a], auch wenn die Syntax der HTML-Syntax ähnlich sieht, werden alle JSX-Elemente in die genannte JavaScript-Funktion kompiliert.

JSX unterstütz die Deklaration von regulären HTML-Elementen, als auch die von React-Komponenten. Unterschieden werden die beiden Varianten dabei nach Groß- und Kleinschreibung, wobei React-Komponenten groß- und reguläre HTML-Elemente klein geschrieben sind. React-Komponenten können dabei, wie oben bereits erwähnt, `props` mitgegeben werden. Der Schritt von HTML zu JSX stellt in der Entwicklung keine große Herausforderung dar. Der wichtigste Punkt ist die Verwendung von `class`. Auch bei der Entwicklung mit React werden für DOM-Elemente häufig Klassen vergeben. Da JSX-Code aber in JavaScript kompiliert wird, kann das in JavaScript reservierte Wort[3] `class` nicht verwendet werden. Es muss stattdessen auf `className` zurück gegriffen werden.

3.3 Einstieg in Redux

Redux[4]erlaubt die zentrale Verwaltung des state bzw. des Zustandes der Anwendung im sogenannten redux store. Dabei ersetzt der redux store nicht zwangsweise den state von einzelnen Komponenten. Es gibt keine genauen Richtlinien dafür, wann ein state in der Komponente und wann im redux store verwaltet werden sollte, jedoch bietet es sich als Richtlinie an, nur Zustände im redux store zu verwalten, die von mehr als nur einer ein-

[3] `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar`

zigen Komponente verwendet werden. Redux besteh dabei aus drei Grundbestandteilen: Dem *Store*, den *Actions* und den *Reducern*.

Wie bereits erwähnt wird im Store der Zustand der Anwendung verwaltet. Der *Store* an sich kann dabei ein beliebiges JavaSCript Objekt (Funktionen ausgenommen) oder Primitiv sein. Für diesen *Store* hat die Anwendung nur Leserecht, er darf nicht direkt manipuliert werden. Für eine Änderung des Stores werden die anderen beiden Bestandteile benötigt.

Actions beschreiben Aktionen und sind JavaScript Objekte. Sie müssen mindestens einen `type` Parameter haben, der beschreibt, welche Aktion ausgeführt werden soll. Weiterhin können sie Daten definieren, die für die Änderung des stores von Bedeutung sind. Das absenden dieser actions ist der einzige weg, mit dem die Anwendung den *Store* verändern kann.

Reducer definieren, wie der Zustand der Anwendung je nach erhaltener Aktion verändert werden muss. Ist beispielsweise der `type` einer *Action* `INCREMENT_DOWNLOADS_COUNT` kann im *Reducer* definiert sein, dass beim erhalten dieser action das Feld `downloadsCount` im *Store* um eins erhöht werden muss.

Ein *Provider* macht den *Store* dann für Container-Komponenten verfügbar. Diese verteilen den *Store* und die actions als props an representative Komponenten. Ändert sich der *Store* erhalten die entsprechen betroffenen Komponenten also neue *props* und werden somit neu gerendert. Abb XYZ zeigt einen schematischen Ablauf einer Aktualisierung des redux stores.

Detailliertere Erläuterungen zur Implementation von redux im Rahmen der Anwendung finden sich in Kapitel ABC.

[4] `http://redux.js.org/`

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 Beispielswiese 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 ZXC 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 erlaub das wechseln zum nächsten Abschnitt in der Anwendung.

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 (s. Abb. ZXY). Zum Anderen zeig die Auswahl von Farben gut, wie Interaktive Elemente der Anwendung gleichzeitig auch die Gestaltung des Nutzer zeigen können. Die Buttons zur Auswahl einer Farbe (s. Abb. ASH) bestehen hier lediglich aus der Farbe selbst, die Gestaltung der Anwendung wird hier als visuelle Zwischenebene also komplett entfernt.

4.1.1 Vorgehen

TBD

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 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.l`). 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='l' >
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 ei-

ne 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 währen 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 Zeitpunk 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 Storybook[1] 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.

[1] `https://github.com/storybooks/storybook`

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

- Trennung in verschiedene Sub-Stores
- Container
- Kein direktes dispatch in Components mehr

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 seine hier nur als die populärsten Varianten beispielhaft genannt). Auch die Verwendung von CSS-Präsporzessoren 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 simplifizierte 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
```

```
  </ReactComponent>
```

Und könnte im Stylesheet über

```
.myClass {
  color: red
}
```

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

```
const styles = {
  backgroundColor: 'red',
  fontSize: '12px'
}

export default function myComponent(props) {
  return (
    <div styles={{styles}} >
    {props.children}
    </div>
  )
}
```

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 (abge-

sehen 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 Bespiel 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 wir 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 4.2 zu sehen ist[2], 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(ke` 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 4.2: Die Komponente `SetupProgress` in gekürzter Form

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

[2]Teile 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='l'>
87              <span>Choose what you are building</span>
88              <SpacingInset size='l' />
89              <div className={css(styles.buttonWrapperStyles)}>
90                {this.generateContent()}
91              </div>
92              <SpacingInset size='l' />
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 Sate, 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 Länger der verschiedenen Äste des Baumes bei der Verwendung des Arrays eine weniger große Rolle, während bei der Verwendung eines Objektes immer auch geprüft werden muss, ob der key im store überhaupt vorhanden ist. Daher wurde an dieser Stelle ein Array verwendet. [HIER SPÄTER NOCH MAL SCHAUEN, OB DAS NOCH STIMMT]

4.3.3 Anzeige von Inhalten nach Scope

Ein Hauptaugenmerk währen der Entwicklung lag auf der Anwendung der vom Nutzer gewählten *scope* in der Anwendung. In Abhängigkeit dieser Scopes muss die Anwendung verschiedene Daten präsentieren. Dabei muss die Möglichkeit bestehen, diese Daten zu erweitern oder zu verändern, ohne dass die Anwendung selbst dafür umstrukturiert werden muss. Diese Datenhaltung soll hier am Beispiel der verschiedenen Schriftfamilien, die im Bereich Typographie Verwendung finden können, gezeigt werden.

Da die Anwendung keine Datenbank implementiert, werden diese Daten in eigenen Dateien als JavaScript-Objekte gespeichert. Listing 4.3 zeigt das Objekt, in dem die verschiedenen Schriftarten der *scopes* als Arrays gespeichert sind. Bei Betrachtung des Objektes fällt auf, dass sich Daten teilweise wiederholen. Obwohl hier gegen das Prinzip *Don't Repeat Yourself* verstoßen wird, ist eine solche Struktur mit Blick auf eine Weiterentwicklung der Anwendung nötig, um ein möglichst einfaches Verändern eines einzelnen *scopes* gewährleisten zu können.

Listing 4.3: 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* entsprechen[3], ist ein einfaches Ermitteln der benötigten Schriftfamilien, wie es in Listing 4.4 gezeigt wird, möglich.

Listing 4.4: 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. Währen 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'}
```

[3] Auch 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 4.5 zeigt die Funktion `calculateComplemntary`, die diese Berechnung ausführt.

Listing 4.5: Berechnung eines Komplementär-Kontrastes

```
1   export function calculateComplemntary(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 erzeugen[5].

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

[5] Objekte 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 weitern, veränderten Farben).

Für jede Farbe müssen hier also verschiedene Entscheidungen getroffen werden. Zunächst muss entscheiden 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 4.6 zeigt, zufällig getroffen.

Listing 4.6: 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 4.7).

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

Listing 4.7: Setzen der HSL-Werte

```
changeValuesOfColor(values, color) {
  let manipulatedColor = Object.assign({}, color)
  // Switch case
  // Do the changes that need to be DropdownController
  switch (values) {
    case 0:
      // change lightness
      manipulatedColor.l = Math.random().toFixed(2)
      break
    case 1:
      // change Saturation
      manipulatedColor.s = Math.random().toFixed(2)
      break
    case 2:
      // change lightning and saturation
      manipulatedColor.l = Math.random().toFixed(2)
      manipulatedColor.s = Math.random().toFixed(2)
      break
    default:
      throw new 'Oops, seems like the randomizer messed something up.'
  }

  return manipulatedColor
}
```

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 4.8 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 4.8: Überprüfen der Ähnlichkeit zweier Farben

```
checkForSimilarColors(colors, candidate) {
  let similarValues = false
  for (var j = 0; j < colors.length; j++) {
    let saturationDifference = Math.abs(colors[j].s - candidate.s)
    let lightnessDifference = Math.abs(colors[j].l - candidate.l)

    if (saturationDifference < 0.1 && lightnessDifference < 0.1) {
      similarValues = true
    }
  }

  return similarValues
}
```

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 file 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 veröffentlich 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 Anwenudung

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 Progrmmiersprachen und beitet keinen Support für das Hosting von statischen files. Um diese Anwendung zu hosten, muss also ein Server geschreiben 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 alsolediglich 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 diese 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 er 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 befasst haben. Uns 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 des 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 weiter Sicherung einzubauen, wurden die beiden Branchen 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, kenn 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 Punk 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 From 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 diveristät 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 spannendens. 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.

[Gac15] Cory Gackenheimer. *Introduction to React*. Apress, 1st ed. edition, 9 2015.

[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. Jsx in depth, 2016. https://facebook.github.io/react/docs/jsx-in-depth.html.

[Inc16b] Facebook Inc. React.component, 2016. https://facebook.github.io/react/docs/react-component.html.

[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.

[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.

[Nic98] Raymond S Nickerson. Confirmation bias: A ubiquitous phenomenon in many guises. *Review of general psychology*, 2(2):175, 1998.

[Pop16] Christian Poplawski. Ermittlung relevanter themengebiete für die entwicklung eines tools zur unterstützung beim erstellen von gestaltungslösungen im hochschulkontext. 2016.

[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.