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.

Im Folgenden Kapitel sollen zunächst einige grundlegende Ziele und das generelle Vorgehen bei der Anfertigung der Arbeit erläutert werden

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

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

Die Motivation für diese Arbeit ergibt sich daher aus dem Glauben, dass eine umgesetzte Form dieser Konzepte einen realen Mehrwert hat, der über den Erhalt von *Credit Points* hinaus geht. Eine genauere Abhandlung der Relevanz dieser Arbeit sich Kapitel 1.3 auf Seite 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
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 Ausliefern auf einem Server an (im Gegensatz zu beispielsweise nur der Bereitstellung des Anwendungscodes, der lokal ausgeführt werden muss). Weitere details zu diesem Thema werden im Kapitel 5.1 erläutert.
Damit die Anwendung genutzt werden und potentiellen Nutzern helfen kann, müssen diese von ihrer Existenz wissen. Eine Vermarktung der Anwendung ist dabei nicht Ziel dieser Arbeit, jedoch muss die Anwendung in einer Form vorliegen, in der eine Vermarktung möglich ist.

Zuletzt sei auch die Möglichkeit zur 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 i Kapitel 5.2

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:

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?

[2] `https://github.com`

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 alle möglichen Szenarien seiner Benutzung abdeckt. 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 des Themengebietes wurde bereits im Praxisprojekt erläutert, daher soll hier nur eine Kurzfassung 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. [Pop16]

Hauptsächlich gestützt wurde dieser Gedanke von einer Studie von Lindgaard et al. [LFDB06]. 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, Abläufe und Entscheidungen.

1.4 Zielgruppe

Während der Konzeption 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

Der weitere Aufbau dieser Arbeit orientiert sich an den verschieden Tätigkeitsbereichen, die währen der Umsetzung der Anwendung relevant waren.
Im zweiten Kapitel werden zunächst die aus dem Praxisprojekt übernommenen Konzepte evaluiert, vervollständigt und gegebenenfalls erweitert. Das dritte Kapitel befasst sich mit den technischen Grundlagen der Anwendung. Hier findet sich eine Diskussion zur Wahl der Technologie, mit der die Anwendung umgesetzt wurde sowie ein Einstieg in diese. Darauf folgend wird in Kapitel vier auf die Entwicklung der Anwendung, deren Gestaltung und einige Entscheidungen im Aufbau der Architektur eingegangen. Abschließend wird im fünften Kapitel die Veröffentlichung der Anwendung und die damit verbundenen Entscheidungen thematisiert.

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

Aufgrund der Zielsetzung, innerhalb der Abschlussarbeit eine Marktfähige (und damit in ihren Features hochwertige) Anwendung zu erstellen und des gegebenen Zeitrahmens für deren Umsetzung, muss die Liste der zu behandelnden Gebiete eingeschränkt werden.
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.
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 Ein- 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 aufgrund der Verständlichkeit in *Layout & Grids* umbenannt):

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

2.2 Einstieg in die Anwendung

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

> Da sich dieses Tool nicht über die plattformspezifischen Richtlinien hinwegsetzen soll, sollte von Anfang an die Plattform, für die der Nutzer gestaltet, bekannt sein. Die Inhalte des Tools sollten sich dementsprechend anpassen. [Pop16]

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

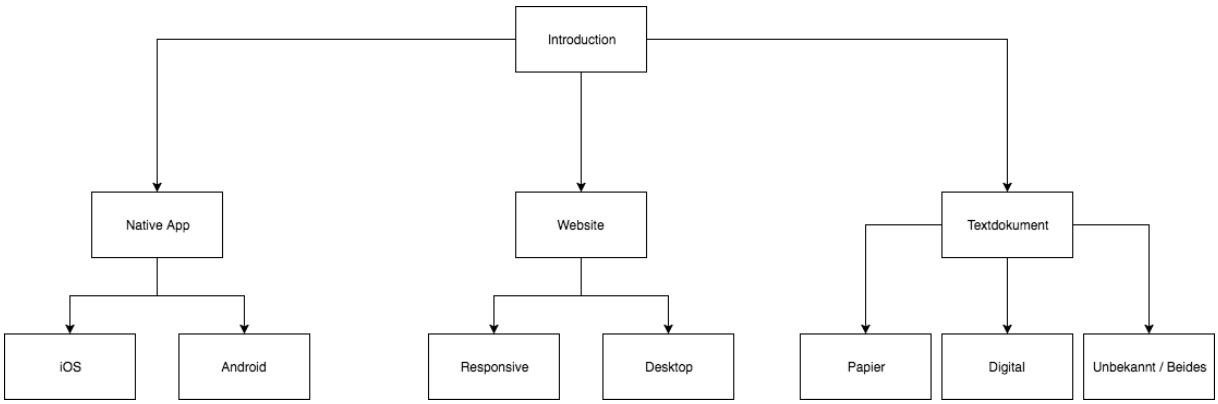

Abbildung 2.1: Zielmedien der Nutzer und deren Unterkategorien

Obwohl die Abgrenzung der Bereiche für die 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 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]

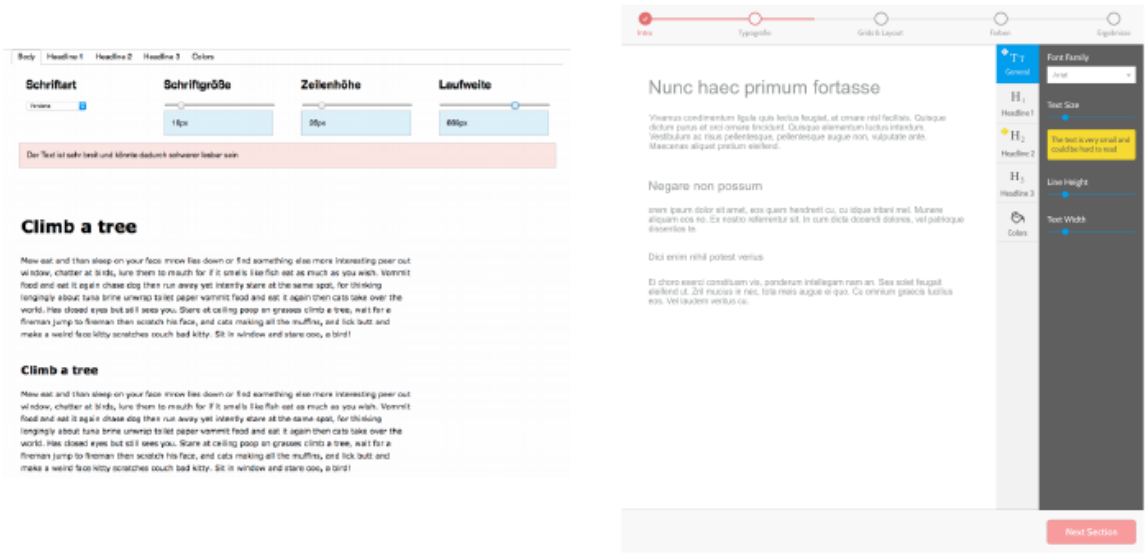

Abbildung 2.2: Der Bereich Typographie, im Praxisprojekt und der Abschlussarbeit

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 erforderlich wäre, 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 Werte verschiedener Bereiche 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 Praxisprojekts wurde auch hier bereits ein grundlegendes Konzept aufgestellt. Die wichtigsten Punkte, in wenigen Sätzen zusammen gefasst, sind:
Die Farbfindung geschieht in zwei Schritten, dem Finden der Grundfarbe und dem 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 Layout & Grid

Im Bereich Layout 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

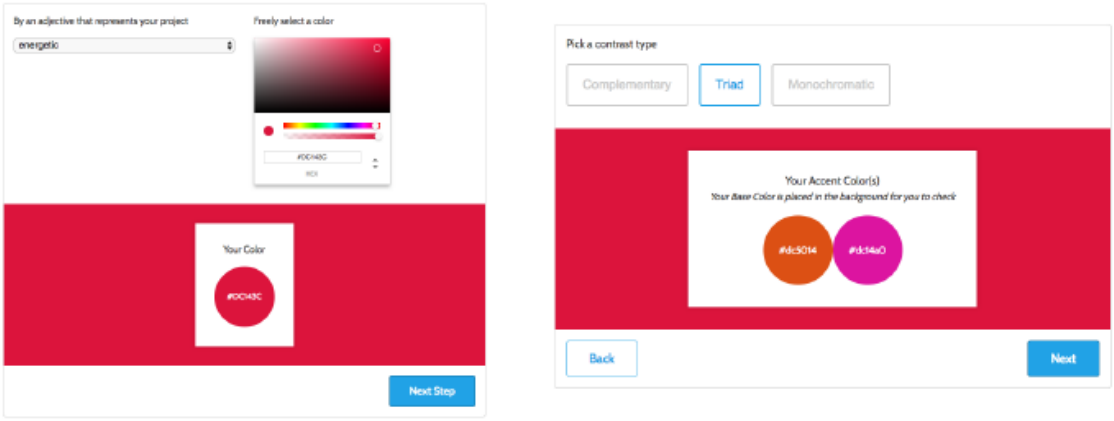

Abbildung 2.3: Vorschau der ausgewählten Farben in der Anwendung

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, es 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 Zugriff zu haben.

Dieser Bereich wurde im Rahmen des Praxisprojekts 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 eines Backends, das entsprechende Daten verwalten kann. Hier könnten auch mehrere Projekte eines Nutzer gespeichert werden,

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

allerdings liegt der Entwicklungsaufwand für ein solches 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

Technischen Grundlagen

Das nachfolgende Kapitel soll die technischen Grundlagen erläutern, die für die Umsetzung der Anwendung von Bedeutung sind. Im Folgenden sollen verschiedene Möglichkeiten der Umsetzung diskutiert und der Prozess für die Wahl der verwendeten Technologien und Frameworks erläutert werden.
Weiterhin sollen im zweiten Teil dieses Kapitels die nötigen Grundlagen für ein Verständnis des Aufbaus der Anwendung und ihres Quellcodes geschaffen werden.

3.1 Diskussion verfügbarer Technologien

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

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

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

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

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

3.1.1 Rich Internet Applications

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

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

Um diese Interaktivität und Dynamik umzusetzen, wurden zunächst Plattformen mit eigenen Laufzeitumgebungen genutzt. Lawton nennt hier beispielsweise Microsoft Silverlight, Adobe Flash oder JavaFX [Law08]. Die Verwendung von eigenen Laufzeitumgebungen dieser Plattformen bedeutete für den Nutzer jedoch, dass dieser bestimmte Programme auf seinem Rechner oder Plugins in seinem Browser installieren musste, um Zugriff auf die so mögliche Interaktivität zu erhalten. Auf einen der größten Vorteile der Entwicklung von Webanwendungen, das Ausführen der Anwendung ohne vorherige Installation, musste somit verzichtet werden.

Durch das Verabschieden von neuen Web-Standards wie HTML5 oder ECMAScript 6 können viele Funktionen der oben genannten Plattformen nativ in einem Browser abgebildet werden. So empfiehlt Adobe beispielsweise, Flash nicht für Funktionen zu nutzen, die auch mit Hilfe von HTML5 abgebildet werden können[1].
Für die Realisierung dieser Interaktivität im Browser wird vor allem JavaScript verwendet.

3.1.2 Single Page Applications

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

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

ausgeliefert werden. Mikowski und Powell definieren SPAs als

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

Single Page Applications (SPAs) können somit als Untereinheit der *Rich Internet Applications* bezeichnet werden.
Eine der ältesten Möglichkeiten, eine Single Page Application zu realisieren stellt AJAX dar. AJAX erlaubt das Nachladen von Inhalten vom Server, ohne das erneute Laden der im Client geöffneten Seite zu erzwingen [Pau05].
Da eine Anforderung an die zu entwickelnde Anwendung aber die Absenz eines Backends ist, kann AJAX als Möglichkeit zur Umsetzung ausgeschlossen werden.

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

3.1.2.1 JavaScript Single Page Application Frameworks

Aus der Menge der verfügbaren JavaScript-Frameworks, die sich für die Umsetzung einer *SPA* eignen, wurden hier drei für eine Untersuchung auf die Tauglichkeit für die Umsetzung der konzipierten Anwendung hin untersucht. Die Auswahl der Frameworks erfolgte dabei nach Verbreitung und tatsächlicher Verwendung.
Da sich die genaue Messung der Verbreitung einer Software schwierig gestaltet, wurde als Indikator für die Beliebtheit und Verbreitung die Zahl der *stars*[2], die den verschiedenen Frameworks auf der Plattform GitHub zugeordnet sind, verwendet.

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

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

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

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

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

Vue.js ist dabei (wie die anderen hier behandelten Libraries und Frameworks auch) Komponentenbasiert. Um das User Interface effektiv zu aktualisieren, verwendet es einen *Virtual DOM*. Dieser ist eine Abstraktion des DOMs, wie ihn beispielsweise das W3C spezifiziert:

> 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. [Rob17]

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

> 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. [JS17b]

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

Das von Google Entwickelte Framework **Angular.js** [Goo] ist das älteste der hier behandelten Frameworks, die erste Version wurde im Oktober 2010 veröffentlicht.
Insgesamt ist Angular deutlich komplexer als Vue.js und React.js. Viele Erweiterungen, wie zum Beispiel das Routing, die in Vue.js und React.js durch externe Bibliotheken implementiert werden müssen, sind beispielsweise in Angular.js bereits enthalten. Auch weisen Angular-Anwendungen eine deutlich Komplexere Architektur auf, die unter anderem aus Teilen wie Modulen, Komponenten und Templates besteht. Auch hier wird für die Darstellung von Inhalten eine Templating-Engine verwendet.

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

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

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

3.2 Einstieg in React.js

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

3.2.1 Komponenten

Die komponentenbasierte Entwicklung ist einer der Hauptbestandteile der Architektur einer React-Anwendung [Gac15, S. 28].
Dieser Abschnitt beschäftigt sich mit dem Aufbau, der Verwendung, den verschiedenen Formen und den Besonderheiten von Komponenten in React.
Bevor auf die Besonderheiten von Komponenten in React.js eingegangen wird, soll im

Folgenden zunächst ein Überblick über das Konzept der Komponente an sich und im Kontext der Software-Entwicklung gegeben werden.

3.2.1.1 Was ist eine Komponente?

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

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

Und De Alfredo und Henzinger schreiben über Komponenten:

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

Für den Rahmen dieser Arbeit können daraus für Komponenten also folgende Eigenschaften gefolgert werden:
Eine Komponente ist nur ein einzelner Teil der ganzen Anwendung. Dieser Teil sollte (nach Möglichkeit) auch an anderen Stellen in der Anwendung wiederverwendbar sein und sollte weiterhin keine Abhängigkeit von seiner Umwelt besitzen.

3.2.1.2 Komponenten in React.js

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

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

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

drei Zuständen [Inc16c]:

1. Mounting
2. Updating
3. Unmounting

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

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

Für das Arbeiten mit Komponenten in React ist weiterhin das Konzept der *props* und des *state* relevant [Inc16a].
Dabei bezeichnen *props* JavaScript-Objekte, die von einer Elternkomponente zur Kindkomponente übergeben werden können. Besonders interessant ist die Möglichkeit, Referenzen auf Funktionen übergeben zu können. Somit kann in einer Elternkomponente auf die Interaktion mit einer Kindkomponente reagiert werden.
Der *state* bezeichnet im weitesten Sinne den Zustand einer Komponente. Dieser Zustand ist ein Attribut einer Komponente, in dem Werte gespeichert werden, die für die Anzeige des Zustandes der Komponente relevant sind (zum Beispiel, welche Kindkomponente als aktiv dargestellt werden soll).

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

3.2.1.3 Zustandslose Komponenten

Nicht jede Komponente in React.js muss einen Zustand verwalten, viele Komponenten innerhalb einer Anwendung dienen nur der Darstellung von Inhalten, die ihnen von ihren Elternkomponenten übergeben werden. Vor den in Kapitel 3.2.1.1 definierten Anforderungen an eine Komponente ist es Sinnvoll, viele in sich abgeschlossene und in ihren Funktionen limitierte Komponenten zu implementieren, die dann von einigen wenigen Komponenten kontrolliert werden.
Da zustandslose Komponenten keine der im vorhergehenden Kapitel erwähnten Methoden des Lebenszyklus implementieren, wurde mit dem Release von React 0.14[3] eine simplifizierte Schreibweise für diese Komponenten eingeführt (Listing 3.1 zeigt diese). Hier wird zum Einen das unnötige Schreiben von Code vermeiden, zum Anderen kann während der Entwicklung auf den ersten Blick festgestellt werden, ob eine Komponente einen Zustand verwaltet oder nicht.

Listing 3.1: Simplifizierte Schreibweise für Komponenten ohne *state*

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

Ein Beispiel für die Verwendung von Komponenten mit und ohne *state* innerhalb der hier geschriebenen Anwendung findet sich in Kapitel 4.4.1.

3.2.2 JSX

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

JSX ist dabei lediglich eine syntaktische Verschönerung der Funktion `React.createElement(componer` [Inc16b], die für die tatsächliche Definition von Komponenten verwendet wird. Obwohl die

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

JSX-Syntax der HTML-Syntax ähnlich sieht, gibt es hier einige Besonderheiten, die beachtet werden müssen. Zum Einen werden die Deklarationen von React-Komponenten und regulären HTML-Tags durch unterschiedliche Schreibweisen getrennt (React-Komponenten werden Groß geschrieben). Zum Anderen können hier keine Schlüsselwörter verwendet werden, die durch die ECMAScript-Spezifikation geschützt sind[4]. Dies betrifft in erste Linie das `class`-Attribut, das in HTML für das Vergeben von Klassen verwendet wird. Für die Vergabe von CSS-Klassen muss in JSX auf `className` zurück gegriffen werden.

3.3 Einstieg in Redux

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

Um den Zustand einer Anwendung an einem zentralen Ort verwalten und diesen strukturiert verändern zu können, kann die Library Redux.js[5] verwendet werden. Die Dokumentation der Library definiert deren Ziel wie folgt:

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

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

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

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

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

[5] `http://redux.js.org/`, zuletzt abgerufen am 11.08.2017

Die Implementierung von Redux in der hier zu entwickelnden Anwendung wird in Kapitel 4.2 beschrieben.

Kapitel 4

Umsetzung der Anwendung

Im folgenden soll die Umsetzung der Anwendung thematisiert werden. Umsetzung meint hier sowohl die Programmierung der Anwendung und alle Bereiche, die mit dieser zusammen hängen, als auch die visuelle Gestaltung. Dieses Kapitel geht auf Konzepte ein, die währen der Gestaltung der Anwendung eine zentrale Rolle spielten und schildert das generelle Vorgehen. Es werden einige strukturelle Entscheidungen erläutert, die währen der Umsetzung von Relevanz waren und einige konkrete Codebeispiele gezeigt, die als besonders interessant erachtet wurden.

4.1 Gestaltung

Bereits in Kapitel 1.3 wird deutlich, welche zentrale Rolle die Gestaltung in jedwedem Projekt einnimmt. Auch das hier bearbeitete Projekt ist davon nicht ausgenommen. Um eine gute Benutzbarkeit des Tools zu gewährleisten, ist eine solide Gestaltung unabdingbar.
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]

Insgesamt ist *25knots* eine sehr interaktive Anwendung, in der Informationen eher Grafisch und durch Interaktionen des Benutzers übertragen werden, als Beispielswiese durch Texte. Daher wurde bei der Gestaltung großer Wert darauf gelegt, klar zu kommunizieren, welche Bereiche interaktiv sind und welche Auswirkungen eine Interaktion mit diesen Bereichen hat.

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

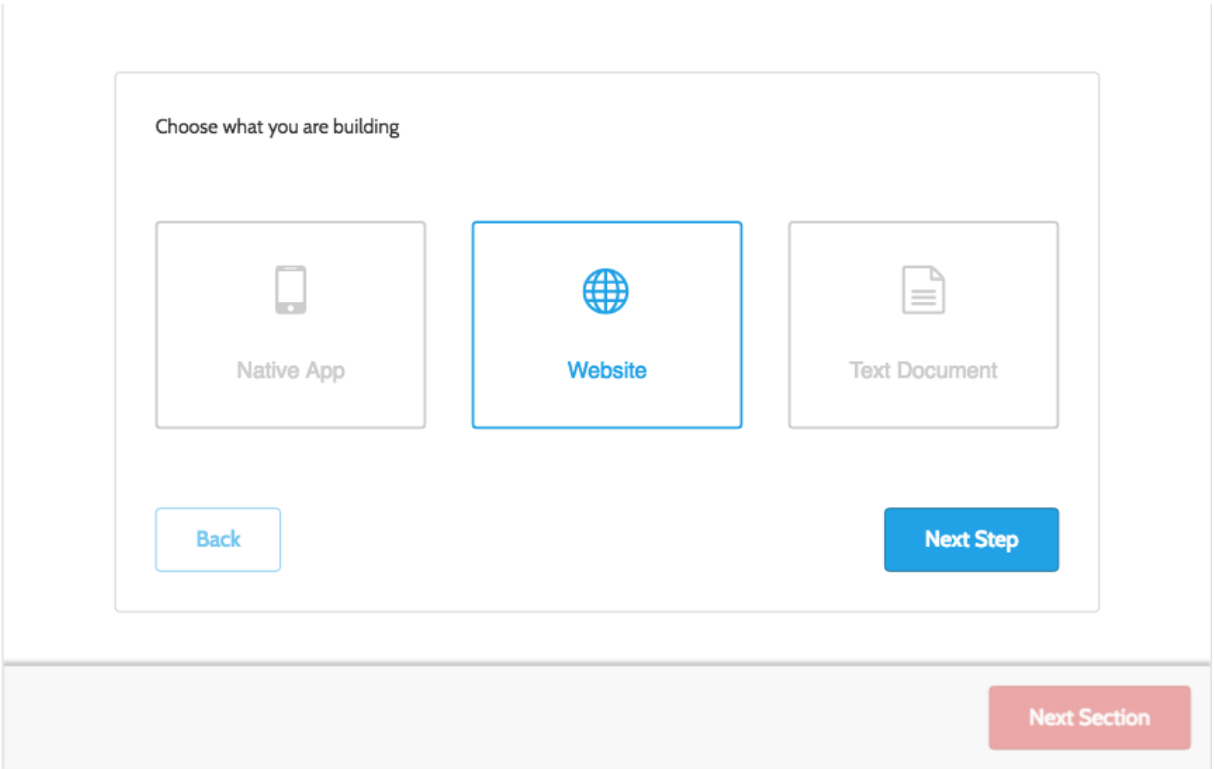

Abbildung 4.1: Farben verschiedener interaktiver Elemente

Weiterhin gilt es zu beachten, dass der Nutzer die Anwendung verwendet, um eine Gestaltung für sein Projekt zu erstellen. Der Fokus der Anwendung sollte daher darauf liegen, dem Nutzer die von ihm erarbeitete Gestaltung zu zeigen. Die Anwendung selbst 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 Experimente zu verzichten. Der größte Teil der Anwendung ist in Weiß- und Grautönen gehalten, um 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 für diesen bestimmten Anwendungsfall (das hervorheben von Fehlern) verwendet wird und dem Nutzer dadurch schnell auffällt (siehe Abbildung 4.2). Zum Anderen zeigt die Auswahl von Farben (vor allem in den *scopes* Android und iOS) gut, wie Interaktive Elemente in der Anwendung bereits die Gestaltung des Nutzers zeigen können. Die Buttons zur Auswahl einer Farbe (siehe Abbildung 4.3) bestehen hier lediglich aus der Farbe selbst, die Anwendung und deren Gestaltung wird hier als visuelle Zwischenebene komplett entfernt.

Abbildung 4.2: Fehlermeldungen im Bereich Typographie

4.1.1 Vorgehen

Das Vorgehen während der Gestaltung lässt sich als *agil* bezeichnen. Diese Agilität drückt sich auf verschiedene Weise aus.
Der Entwicklungsprozess (gemeint ist hier die tatsächliche Programmierung) und der Gestaltungsprozess (das Erstellen von Mock-Ups in einem entsprechenden Grafikprogramm)

Abbildung 4.3: Buttons zum Auswählen einer Akzentfarbe

waren nicht strikt von einander getrennt, sondern überschnitten sich. Diese Überschneidung zeigt sich sowohl in einer langfristigen, als auch in einer kurzfristigen Betrachtung. So gab es nicht einen Gestaltungsprozess, nach dessen Fertigstellung die Entwicklung startete, sondern vielmehr einen Gestaltungsprozess pro Schritt der Anwendung, der umgesetzt wurde.

Auch innerhalb eines Schrittes war das Vorgehen nicht linear, häufig wurden während der Entwicklung Aspekte in der Gestaltung verändert. Die Gründe hierfür liegen vor allem in den verbesserten Möglichkeiten zum testen und validieren von Interaktionen und Konzepten in einer prototypischen Realisierung im Vergleich zu statischen Mock-Ups. Durch die in Kapitel 4.1.2 und 4.1.3 erläuterten Aspekte konnten außerdem problemlos kleinere Änderungen in der Gestaltung während der Entwicklung vorgenommen werden, ohne, dass dabei ein vorheriges Anpassen der Mock-Ups nötig war.

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 entwerfen. 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.

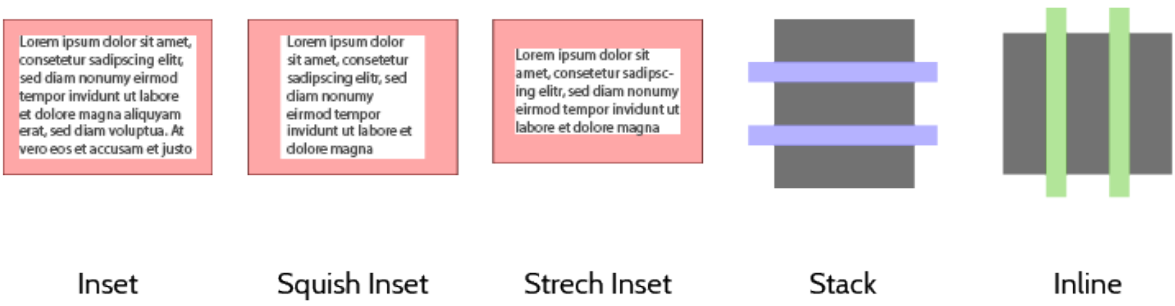

Abbildung 4.4: Die 5 implementierten Arten von Spacing nach [Cur16]

Curtis definiert 6 verschieden Arten von Abständen, von denen 5 innerhalb der Anwendung als eigenständige Komponenten definiert wurden (siehe Abbildung 4.4).
Die Komponenten nehmen die Größe des Abstandes als `prop` in einer der definierten Kleidergrößen entgegen und erzeugen ein `div` Element, das 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 eine ausschließlich visuelle Funktion haben. So könnte deren Funktion auch innerhalb der CSS-Regeln von anderen Komponenten definiert und so ein übersichtlicheres Markup geschaffen werden.
Während der Arbeit stellte sich heraus, dass die Definition er Abstände als eigene Komponenten ein sehr einfaches Entwickeln von Interfaces ermöglichte. Durch die eingegrenzten Möglichkeiten ist auch 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 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 wird lokal im Browser ausgeführt 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. Die Bibliothek bietet dadurch außerdem den Vorteil, dass Komponenten zunächst alleinstehend entwickelt werden können, ohne dass diese in die eigentliche Anwendung eingebunden werden müssen.

4.2 Redux

Im laufe der Anwendung müssen bestimmte Informationen über die gesamte Anwendung hinweg für bestimmte Komponenten abrufbar sein. Dies betrifft vor allem, aber nicht ausschließlich, die vom Nutzer zu beginn der Interaktion mit der Anwendung definierten Entwicklungsziele, die in jedem Bereich der Anwendung für die korrekte und auf das jeweilige Ziel angepasste Darstellung der Inhalte überprüft werden.
Ein weiteres Beispiel stellt die Zusammenfassung am Ende der Interaktion mit der Anwendung dar, für die ein Zugriff auf alle vom Nutzer definierten Werte nötig ist. Der Redux-Store für die Anwendung umfasst daher Werte aus den Bereichen *Intro*, *Typographie* und *Farben*.

Um die Arbeit mit dem Store zu simplifizieren und eine übermäßige Verschachtelung des Store-Objekts zu vermeiden, wurde für jeden der oben genannten Bereiche ein eigener *Reducer* geschrieben, der ausschließlich für die Bearbeitung der diesem Bereich zugehörigen Werte verantwortlich ist.
In der Datei `ApplicationState.js` werden diese mit Hilfe der Funktion `combineReducers`, die von Redux zur Verfügung gestellt wird, dann zu einem Objekt zusammengefügt (siehe Listing 4.2).

Listing 4.2: Zusammenfügen der dedizierten Reducer zu einem Objekt

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

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

4.2.1 Beispielhaftes Verändern des Redux-Store

Das Verändern des Redux-Store soll im Folgenden an einem konkreten Beispiel verdeutlicht werden. Das Szenario, das der Nutzer durchläuft, ist dabei das Auswählen einer Grundfarbe. In diesem Szenario hat der Nutzer bereits eine Wahl über seine Grundfarbe getroffen und möchte nun durch den Klick auf den *Next Step* Button seine Auswahl bestätigen und zum nächsten Schritt übergehen (vergleiche hierzu Abbildung 2.3 auf Seite 16).
Für die Anwendung bedeutet diese Interaktion: Die aktuell gewählte Grundfarbe muss in den Redux-Store geschrieben[2] und der nächste Schritt für die Farbfindung angezeigt werden. Dieses Beispiel soll sich dabei auf das schreiben der Grundfarbe in den Redux-Store konzentrieren.

Um diese Veränderung im Redux-Store möglich zu machen, muss zunächst eine *action* definiert werden (Listing 4.3), auf deren Aufrufen hin der *reducer* (Listing 4.4) den Redux-Store aktualisiert.

Listing 4.3: Definition der *action* zum setzen der Grundfarben

```
1  export const setBaseColors = (colors) => {
2    return {
3      type: SET_BASE_COLORS,
4      colors
5    }
6  }
```

Listing 4.4: Veränderung des Redux-Store beim Aufrauf der *action* `setBaseColors`

```
1  case SET_BASE_COLORS:
2    return Object.assign({}, state, {
3      baseColors: [
4        ...action.colors
5      ]
6    })
```

Die *action* kann innerhalb der Anwendung durch den Aufruf der Methode `dispatch(setBaseColors(c` ausgelöst werden. Der Aufruf dieser Methode ist theoretisch direkt in der Komponente die den Button definiert, auf den der Nutzer klickt, möglich. Dieses Vorgehen wäre allerdings nicht mit den in Kapitel 3.2.1.1 definierten Anforderungen an eine Komponenten konform. So könnte dieser Button nur an Stellen eingesetzt werden, an denen die Grundfarbe für

[2]Es ist dabei möglich, dass mehr als eine Grundfarbe gespeichert wird, da für das Zielmedium Android insgesamt drei Abstufungen einer Grundfarbe benötigt werden.

den Bereich Farben im Redux-Store gesetzt werden soll (realistisch betrachtet würde diese Komponente also an genau einer Stelle eingesetzt werden).
Um diesen Umstand zu vermeiden, wird die Methode als Referenz weiter gegeben (Abbildung 4.5 zeigt eine informelle Darstellung aller beteiligter Komponenten).

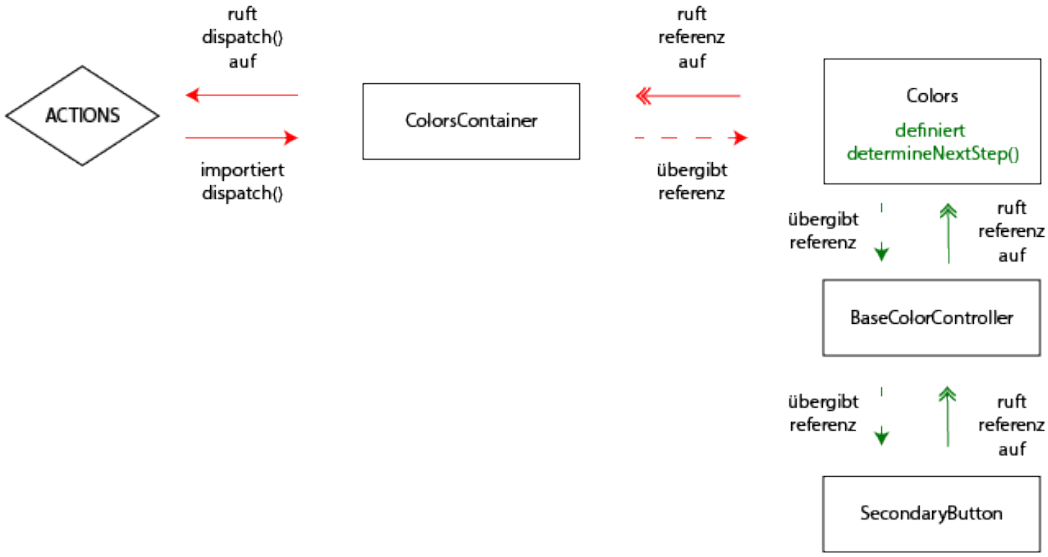

Abbildung 4.5: Übergabe der `dispatch()` Methode als Referenz

In der Komponente *ColorsContainer* wir die Referenz auf die `dispatch()` Methode als `prop` mit dem Namen `setBaseColors` an die Komponente *Colors* übergeben.

Die Komponente *Colors* implementiert die Funktion `determineNextStep`, die Feststellt, welcher Schritt in Abhängigkeit vom Entwicklungsziel des Nutzers der nächste ist. Die Referenz auf `setBaseColors` wird dabei auf jeden Fall aufgerufen. Die Komponente *Colors* gibt nun eine Referenz auf die Funktion `determineNextStep` an die Komponente *BaseColorController* weiter. Diese wiederum gibt diese Referenz an die Komponente *SecondaryButton* weiter, die auch die Komponente ist, mit der der Nutzer interagiert.

Die *SecondaryButton* Komponente führt auf einen Klick die ihr als `prop` übergebene Referenz auf eine Funktion aus (siehe Listing 4.5) und hat selbst kein Wissen darüber, welche Funktion innerhalb der Anwendung sie ausführt. Hierdurch werden die übergebenen Referenzen in rückläufiger Reihenfolge wieder aufgerufen und so der `dispatch()` ausgelöst.

Listing 4.5: Aufruf der übergebenen Funktion in der Komponente *SecondaryButton*

```
function SecondaryButton(props) {
  return (
    <button
      className={setStyles(props.variant, props.inactive)}
      onClick={props.inactive ? '' : props.onClick}
    >
      <SpacingSquishedInset size='l'>
        {props.children}
      </SpacingSquishedInset>
    </button>
  )
}
```

4.3 CSS-Architektur

In der Entwicklung von Webanwendungen wird es als gutes Vorgehen angesehen, Inhalte und deren Gestaltung voneinander zu trennen [Goo02, S. 56]. Diese Trennung erfolgt in der Regel durch das definieren von dedizierten HTML- und CSS-Dateien, die sich nur mit der Struktur von Inhalten bzw. deren Aussehen befassen. Eine der aktuell verbreitetsten Möglichkeiten, Regeln für die Darstellung von Inhalten zu definieren, ist das vergeben von Klassen über das `class`-attribut, die dann in der CSS-Datei über einen Selektor (zum Beispiel `.myClass`) aufgerufen werden können.

Auch React.js bietet die Möglichkeit, diese Architektur abzubilden. In Kapitel ZXC wurde bereits erwähnt, dass hier wegen der Verwendung von JavaScript in Komponenten das Attribut `className` verwendet werden muss.

Durch die Verwendung einer solchen Architektur werden für Komponenten jedoch Abhängigkeiten geschaffen, da für die korrekte Darstellung der Komponente auch immer die entsprechenden Regeln in der CSS-Datei verfügbar sein müssen. Diese Abhängig von ihrer Umwelt ist jedoch nicht Konform mit den in Kapitel AZS definierten Anforderungen an eine Komponente innerhalb dieser Arbeit.

Um das Aussehen innerhalb von HTML-Elementen zu verändern, kann das `style`-attribut verwendet werden. Auch dieses akzeptiert CSS-Syntax, die das Aussehen des jeweiligen Elements definiert. React.js ermöglicht die Verwendung dieses Attributes innerhalb von Komponenten und somit auch die Deklaration von Inhalt und Aussehen innerhalb einer Komponenten, ohne die Notwendigkeit weiterer Abhängigkeiten.

Die Verwendung des `style`-attributes zum Festlegen des Aussehens birgt jedoch einige

Nachteile. So können über das Attribut keine Pseudo-Klassen, wie zum Beispiel `:hover` oder `:before` angesprochen werden. [TC13] Für die Entwicklung der Anwendung sind die Pseudo-Klassen jedoch notwendig. Um dieses Problem zu lösen, wurden im Bereich der *JavaScript-SPAs* viele Bibliotheken entwickelt. Im Rahmen dieses Projektes wurde die Bibliothek Aphrodite[3] verwendet. Diese Bibliothek erlaubt eine Notation der CSS-Regeln wie sie React.js auch nativ ermöglicht, unterstützt aber beispielsweise Pseudo-Klassen (Listing 4.6 zeigt ein simples Beispiel).

Listing 4.6: Beipspielhafte Verwendung der Bibliothek Aphrodite

```
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       ':hover': {
13         color: 'red'
14     }
15   })
16 }
```

Während der Entwicklung wurde nicht jegliche Art von Gestaltung innerhalb von Komponenten realisiert. Verschiedene native HTML-Elemente, die über die Anwendung hinweg verwendet werden, wurden in einer globalen CSS-Datei definiert. Dies hat den Vorteil, dass zum Beispiel für die Verwendung einer Überschrift keine eigene Komponente geschrieben werden muss, die funktional equivalent zu einem nativen HTML-Element (beispielsweise `<h1>`) ist, nur um dessen Aussehen anzupassen.

4.4 Interessante Aspekte in der Entwicklung

Nachfolgend sollen einige konkrete Beispiele aus der Entwicklung der Anwendung erläutert werden. Die verschiedenen Beispiele wurden dabei aus unterschiedlichen Gründen Gewählt: Kapitel 4.4.1 zeigt einen zentralen Aspekt der Umsetzung einer komponentenbasierten Anwendung mit React.js, Kapitel 4.4.2 eine der zentralen Arbeitsweisen der Anwendung. Die Kapitel 4.4.3 und 4.4.4 zeigen die Lösung von Problemen, die für die

[3] `https://github.com/Khan/aphrodite`, zuletzt abgerufen am 12.08.2017

Funktionalität der Anwendung eine hohe Relevanz besitzen.

4.4.1 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 inwiefern sich der Zustand einer Komponente vom Zustand der gesamten Anwendung unterscheidet. 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 (siehe dazu Abbildung 4.6), von denen die meisten *stateless* sind, lediglich die Komponente `SetupProgress`, die in Listing 4.7 zu sehen ist[4], verwaltet einen Zustand.

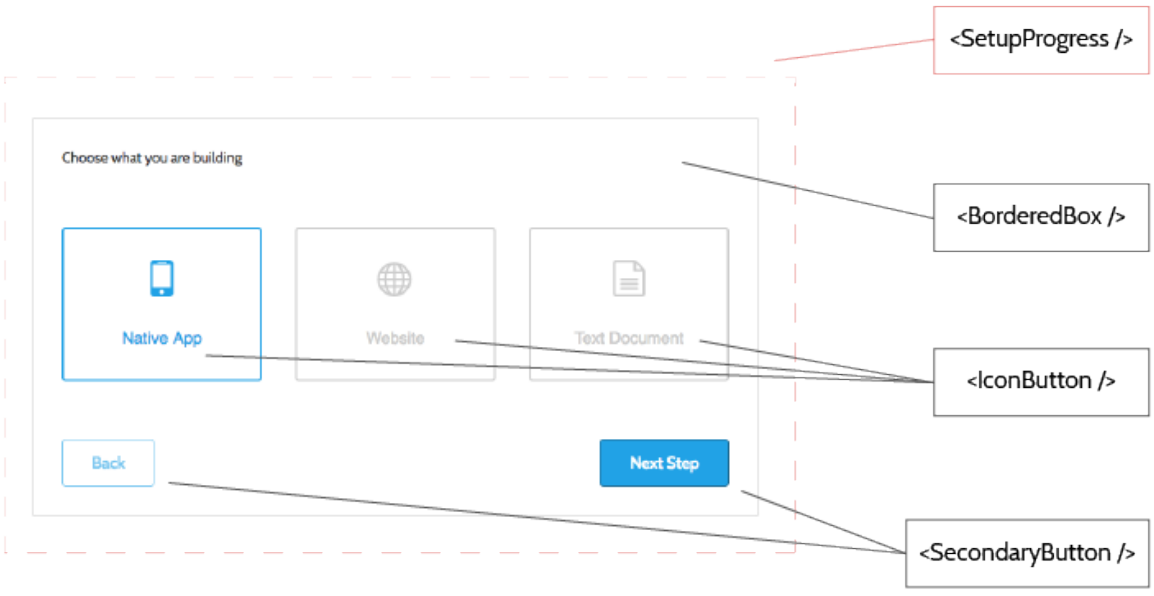

Abbildung 4.6: Komponenten im der Festlegung des Zielmediums

[4]Teile des Quellcodes, die für diesen Anwendungsfall nicht relevant sind, wurden gekürzt. Die gesamte

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 4.7: Die Komponente `SetupProgress` in gekürzter Form

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

Datei kann der beiliegenden CD entnommen werden

```
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.

4.4.2 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 4.8 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 4.8: Aufbau des `FONTS` Objektes

```
export const FONTS = {
  DISPLAY: [
    'Verdana', 'Arial', 'Tahoma', 'TrebuchetMS'
  ],
  RESPONSIVE: [
    'Verdana', 'Arial', 'Tahoma', 'TrebuchetMS'
  ],
  NOT_RESPONSIVE: [
    'Verdana', 'Arial', 'Tahoma', 'TrebuchetMS'
  ],
  PAPER_DISPLAY: [
    'Verdana', 'Arial', 'Tahoma', 'TrebuchetMS', 'Times New Roman', 'Georgia', 'Palatino'
  ],
  PAPER: [
    'Times New Roman', 'Georgia', 'Palatino'
  ],
  ANDROID: [
    'Roboto', 'Noto'
  ],
  IOS: [
    'San Francisco'
  ]
}
```

Da die *keys* im `FONTS` Objekt dabei exakt den möglichen *scopes* entsprechen[5], ist ein

einfaches Ermitteln der benötigten Schriftfamilien, wie es in Listing 4.9 gezeigt wird, möglich.

Listing 4.9: Zugriff auf Werte des `FONTS` Objektes

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

4.4.3 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[6] 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. Die Bibliothek selbst bietet auch einige Funktionen zum Erstellen von Farbkontrasten, die Ergebnisse dieser Funktionen wurden für den Rahmen dieser Arbeit jedoch als nicht geeignet befunden.

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

[5] Auch die verschiedenen *scopes* sind Konstanten, die in einem Objekt gespeichert werden.

[6] `https://github.com/bgrins/TinyColor`, zuletzt abgerufen am 10.8.2017

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 4.10 zeigt, zufällig getroffen.

Listing 4.10: Berechnung eines Monochromatischen Farbschemas

```
1  /**
2   * Calculates a color scheme of monochromatic colors based on a base color with a variable
        amount of colors.
3   * The colors returned by this function are random in saturation and lightness.
4   * The colors returned by this function are guaranteed to not be equal to either each other,
        nor the base color.
5   *
6   * NOTE: Since the colors cannot be similar to each other, the amount of options in limited.
        Therefor, the number of colors to be returned should not be too high (6 will probably
       still work fine, whereas 25 will cause the function to break.)
7   *
8   * @param amount The number of colors that should be returned
9   * @param baseColor A HEX-Value of a color that is the basis of the color scheme
10  * @returns An Array of HEX-Values that build a monochromatic color scheme
11  */
12 export function calculateMonochromaticColors(amount, baseColor) {
13
14   let hslColor = convertToHsl(baseColor)
15   let colors = []
16
17   for (var i = 0; i < amount; i++) {
18     let currentColor = Object.assign({}, hslColor)
19     let randomOption = CHANGABLE_COLOR_ATTRIBUTES[Math.floor(Math.random() * 3)]
20     let changedColor = changeValuesOfColor(randomOption, currentColor)
21
22     // Check, if the color is similar to the base color
23     let similarToBaseColor = colorsAreSimilar(hslColor, changedColor)
24     while (similarToBaseColor) {
25       changedColor = changeValuesOfColor(randomOption, changedColor)
26       similarToBaseColor = colorsAreSimilar(hslColor, changedColor)
27     }
28
29     if (colors.length < 1) {
30       colors.push(convertToHex(changedColor))
31     } else {
32       // Check, if the new color is similar to other colors that were calculated
33       let similarColorsPresent = colorInArrayIsSimilar(colors, changedColor)
34       while (similarColorsPresent) {
35         changedColor = changeValuesOfColor(randomOption, changedColor)
36         similarColorsPresent = colorsAreSimilar(colors, changedColor)
37       }
38       colors.push(convertToHex(changedColor))
39     }
40   }
41
42   return colors
43 }
```

Im nächsten Schritt müssen die Veränderungen in den bestimmten Werten vorgenommen werden, auch hier werden diese Werte zufällig gewählt. Um auszuschließen, dass die definierten Werte zu hell oder zu dunkel sind (also fast Schwarz oder fast Weiss und damit sehr wenig Farbe aufweisen), wurden die Wertebereiche, in denen *Lightness* und *Saturation* verändert werden können, begrenzt. Dieser Vorgang findet in der Funktion `changeValuesOfColor` statt (siehe Listing 4.11).

Listing 4.11: Setzen der HSL-Werte

```
1  /**
2   * Changes the lightness and/or saturation value of an HSL color object to a random value
        and returns a copy of that object.
3   * The random values are restricted to prevent them from beign colorless (i.e. almost black
        or almost white).
4   * A switch case determines, which attribute(s) of the color should be changed.
5   *
6   * @param attributeToChange The attribute on the color to be changed
7   * @param color An HSL color object on which's hue the new color will be based
8   * @returns An HSL color object with the manipulated attributes
9   */
10 function changeValuesOfColor(attributeToChange, color) {
11   let manipulatedColor = Object.assign({}, color)
12   let lightnessVal = Math.random() * 0.85 + 0.15
13   let saturationVal = Math.random() * 0.65 + 0.15
14
15   switch (attributeToChange) {
16     case 'LIGHTNESS':
17       manipulatedColor.l = lightnessVal.toFixed(2)
18       break
19     case 'SATURATION':
20       manipulatedColor.s = saturationVal.toFixed(2)
21       break
22     case 'LIGHTNESS_SATURATION':
23       manipulatedColor.l = lightnessVal.toFixed(2)
24       manipulatedColor.s = saturationVal.toFixed(2)
25       break
26     default:
27       throw new 'Oops, seems like the randomizer messed something up.'
28   }
29
30   return manipulatedColor
31 }
```

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 `colorsAreSimilar` in Listing 4.12 deutlich. Die Funktion gibt dabei `true` zurück, wenn die beiden übergebenen Werte zu ähnlich sind. Anstatt der betroffenen Farbe wird dann in der Funktion `calculateMonochromaticColors` eine neue generiert.

Listing 4.12: Überprüfen der Ähnlichkeit zweier Farben

```
/**
 * Checks if two HSL color objects are similar.
 * Similarity is defined as the Lightness and Saturation being less than 0.1 apart,
 * with the Hue being exactly the same.
 *
 * NOTE: The Hue of the colors is not taken into consideration.
 *
 * @param color The first HSL color object
 * @param candidate The second HSL color object
 * @returns true if the two colors are found to be similar
 */
function colorsAreSimilar(color, candidate) {
  let saturationDifference = Math.abs(color.s - candidate.s)
  let lightnessDifference = Math.abs(color.l - candidate.l)

  if (saturationDifference < 0.1 && lightnessDifference < 0.1) {
    return true
  }

  return false
}
```

4.4.4 Erstellen von PDF-Dateien

Wie in Kapitel 2.6 bereits angesprochen, soll dem Nutzer im letzten Schritt der Anwendung, neben der einfachen Darstellung, die Möglichkeit gegeben werden, seine Ergebnisse in Form einer PDF-Datei zu speichern.
Für die generierung einer PDF-Datei bieten sich verschiedene Möglichkeiten. Da in der Anwendung kein Backend enthalten ist, können Lösungen, die einer Serverseitige Generierung von PDF-Dateien implementieren, bereits zu Beginn ausgeschlossen werden.

Eine der simpelsten der Möglichkeiten, eine PDF-Datei Nutzerseitig zu erzeugen, ist ein Drucken als PDF Datei über das Betriebssystem des Nutzers. Hierbei müsste für die Seite lediglich ein entsprechendes Stylesheet hinterlegt werden, dass das Layout gegebenenfalls für den Druck anpasst. Diese Lösung weist allerdings eine beschränkte Verfügbarkeit auf: Das Betriebssystem macOS bieten den Druck als PDF beispielsweise nativ an, das Betriebssystem Windows aber erst seit der neusten Version, Windows 10. Somit könnte diese Funktion nicht von allen Nutzer verwendet werden.

Eine weitere Möglichkeit stellt die Bibliothek html2canvas[7] 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 in der Datei nicht markiert und kopiert werden, was einen erhöhten Arbeitsaufwand für den Nutzer bedeutet.

Die Entscheidung viel aus diesen Gründen hier auf die Bibliothek jsPDF[8]. Diese erlaubt das erstellen von PDF-Dateien im Browser und auch das einfügen von DOM-Elementen in PDF-Dateien. Das Einfügen von DOM-Elementen ist zwar Komfortabel, bedeutet aber auch einen (zumindest teilweisen) Verlust der Kontrolle über die Struktur und das Aussehen der PDF-Datei. Aus diesem Grund würde die Möglichkeit für die manuelle Erzeugung von PDF-Dateien genutzt, die die Bibliothek ebenfalls ermöglicht. Hierfür steht eine API zur Verfügung, mit der verschiedene Elemente (wie Text oder geometrische Formen), unter Angabe der Position auf der x- und y-Achse, in die Datei eingefügt werden können. Listing 4.13 zeigt eine gekürzte Version der Funktion, die die PDF-Datei erzeugt und speichert, Abbildung 4.7 eine beispielhafte PDF-Datei, die von der Anwendung erstellt wurde.

Listing 4.13: Beispilehafte Generierung einer PDF-Datei

```
1  export function generatePDF(typography, colors) {
2    var pdf = new pdfConverter('p', 'mm', 'a4')
3
4    // shortened for readability
5
6    pdf.addImage(imgData, 'PNG', 65, 10, 81, 30)
7    pdf.setFontSize(10)
8    pdf.text('These are the values you gathered with the help of 25knots', 10, 50)
9
10   pdf.setFontSize(20)
11   pdf.text('Typography', 10, 70)
12   pdf.setFontSize(10)
13   pdf.text('Font Family:', 10, 80)
14   pdf.text(typography.general.fontFamily, 50, 80)
15
16   // shortened for readability
17
18   let date = new Date()
19   let dateString = date.getFullYear() + '-' + (date.getMonth() + 1 ) + '-' + date.getDate()
         + '-' + date.getHours() + '-' + date.getMinutes()
20   pdf.save('25knots_' + dateString + '.pdf')
21 }
```

Für den aktuellen Umfang der Anwendung ist diese Lösung praktikabel, bei einer Erweiterung des Funktionsumfangs muss diese Methode jedoch erneut, auf das Verhältnis von Aufwand und Nutzen hin evaluiert werden

Um dem Nutzer eine spätere Zuordnung der Datei zu vereinfachen, wird diese im Namen mit dem aktuellen Datum und der Uhrzeit versehen. Eine bessere Möglichkeit wäre hier,

[7] https://github.com/niklasvh/html2canvas

[8] https://github.com/MrRio/jsPDF

es dem Nutzer zu ermöglichen, sein Projekt zu beginn der Benutzung der Anwendung namentlich zu benennen.

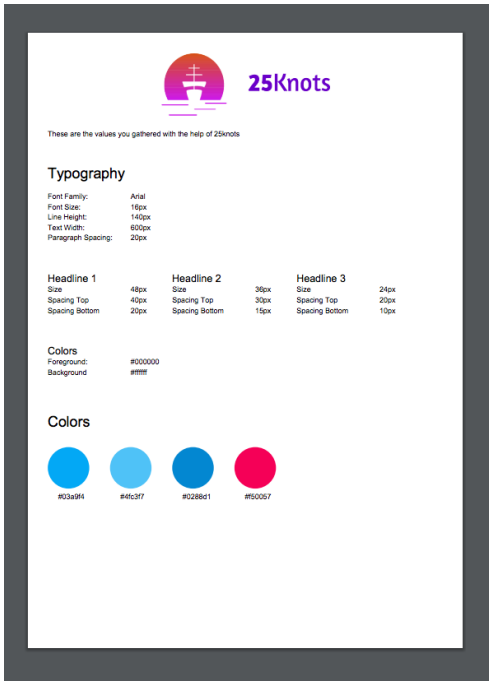

Abbildung 4.7: Beispiel einer generierten PDF-Datei

Kapitel 5

Veröffentlichung der Anwendung

Der letzte Schritt, der für die Entwicklung einer marktfähigen Webanwendung von Bedeutung ist, ist deren Veröffentlichung. Dabei müssen in diesem Fall zwei Aspekte angesprochen werden: Zum Einen muss die Anwendung für den Endnutzer verfügbar sein, zum Anderen ist es aber auch Ziel dieser Arbeit, eine spätere Weiterentwicklung der Anwendung durch die Community zu ermöglichen. Daher muss auch der Quellcode der Anwendung zugänglich und Möglichkeiten zur Mitarbeit definiert sein.

5.1 Ausliefern der Anwendung

Ein publizieren von Ressourcen und Anwendungen im World Wide Web suggeriert in der Regel die Verwendung eines Servers zum ausliefern dieser Ressourcen. Da es sich bei dieser Anwendung um eine *Single Page Application* handelt, sind die technischen Ansprüche an einen solchen Server äußerst gering. Dieser muss lediglich in der Lage sein, zwei statische Dateien auszuliefern. Hier sollen zunächst drei Lösungsmöglichkeiten näher betrachtet werden:

- das manuelle verwalten eines Servers
- die Verwendung von *Github Pages*
- die Verwendung des PaaS *heroku*

Ein manuell verwalteter Server bietet von allen Möglichkeiten die flachste Lernkurve. Es

müssen keine neuen Konzepte erlernt werden und aufgrund des niedrigen technischen Anspruches muss keine komplizierte Konfiguration des Servers erfolgen.
Von Nachteil ist hierbei jedoch die niedrige Flexibilität. Jede neue Version der Anwendung muss manuell auf dem Server abgelegt oder wahlweise ein Automatismus hierfür entwickelt werden.

Da der Quellcode der Anwendung auf Github veröffentlich wurde (siehe dazu auch Kapitel 5.2.1), ist die Verwendung von *Github Pages* eine naheliegende Option. Die Erweiterung der Plattform erlaubt es, den Quellcode eines bestimmten *Branches* innerhalb eines *Repository* statisch auszuliefern.
Die Veröffentlichung einer neuen Version der Anwendung ist dabei sehr einfach, außerdem muss keine weitere Plattform in den Ablauf integriert werden. Während der Tests mit *Github Pages* kam es jedoch zu Konflikten mit einer der verwendeten Bibliotheken. Auch wenn diese Konflikte lösbar waren, lassen diese doch auf mögliche Probleme in der Zukunft schließen.

Heroku erlaubt die Veröffentlichung von komplexen Anwendungen mit wenig Konfiguration. Der Service ist dabei auf dynamische Backends spezialisiert und eigentlich nicht für das Ausliefern von statischen Dateien gedacht. *Heroku* bietet jedoch von allen genannten Lösungen die einfachsten Möglichkeiten, Prozesse zu automatisieren und mit wachsender Größe der Anwendung zu skalieren.

Auch wenn alle hier angesprochenen Lösungen von einem technischen Standpunkt gesehen für die Auslieferung der Anwendung in Frage kommen würden, wird vor dem Hintergrund der möglichen Weiterentwicklung der Anwendung der Service *Heroku* verwendet.

Für eine leichte Zugänglichkeit ist außerdem eine prägnante URL von Vorteil. Aus diesem Grund wurde die Domain `25knots.de` registriert, unter der die Anwendung abgerufen werden kann.

5.2 Weiterentwicklung

Für eine mögliche Weiterentwicklung der Anwendung ist es Voraussetzung, dass Personen ein Interesse daran haben, die Anwendung zu verbessern. Zunächst stellt sich also die

Frage, wie Personen dazu gebracht werden können, sich an der Weiterentwicklung der Anwendung zu beteiligen.
Auch wenn eine konkrete Antwort auf diese Frage schwierig ist, liefern Borges, Valente, Hora und Coelho einen guten Ansatz:

> 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. [BVHC15]

Mit der Menge der Personen, die ein generelles Interesse an der Anwendung und deren Nutzung haben steigt also auch die Menge der Personen, die ein Interesse an deren Weiterentwicklung haben. Ein Weg, Personen für die Weiterentwicklung zu gewinnen ist also, die Anwendung möglichst vielen Personen bekannt zu machen.

Aus einer Mitarbeit einer unbekannt großen Gruppe von Personen ergeben sich außerdem weitere Anforderungen an die Anwendung:

- Der Arbeitsaufwand, um mit der tatsächlichen Entwicklung beginnen zu können, muss so gering wie möglich sein
- Der Quellcode muss in Qualität und Stil gleich bleiben

Diese Anforderungen sollen in den folgenden Kapiteln näher ausgeführt werden.

5.2.1 Vereinfachung der Mitarbeit

Ziel muss es sein, den Aufwand, der zwischen dem Moment liegt, in dem eine Person sich für die Mitarbeit an der Weiterentwicklung des Projektes entscheidet und dem Moment, an dem diese tatsächlich etwas zur Entwicklung beitragen kann, so gering wie möglich zu gestalten. Dies bedeutet in erste Linie, dass diese Person über das Wissen verfügen muss, *wo* und *wie* sie mit der Mitarbeit beginnen kann.

Bereits im vorhergehenden Abschnitt wurde deutlich, dass Personen die an einer Anwendung mitarbeiten zuvor häufig Personen sind, die die Anwendung auch benutzen. Um

diese Personengruppe ansprechen zu können, wird im letzten Schritt der Anwendung auf das Github-Repository verwiesen und erläutert, dass dort eine Mitarbeit an der Weiterentwicklung der Anwendung möglich ist. Jeder Nutzer der Anwendung verfügt somit über das Wissen, *wo* eine Mitarbeit möglich ist.

Um die Frage, *wie* eine Mitarbeit aussehen kann zu beantworten, bietet die Plattform Github selbst einige Hilfsmittel[1]. Eine erste Erklärung gibt die Datei `README.md`, in der sich generelle Informationen zum Projekt und dessen Ausführung finden, aber auch eine Erklärung, auf welche Arten eine Mitarbeit am Projekt erfolgen (zum Beispiel durch Programmierung, Arbeit konzeptioneller Art oder das Melden von Fehlern innerhalb der Anwendung) und an welcher Stelle der Prozess der Mitarbeit begonnen werden kann.

Außerdem wurden Vorlagen zum erstellen von Issues[2]. und Pull Requests geschrieben, die eine Hilfestellung zu den Inhalten bieten, die wünschenswert sind.
Die Inhalte dieser Dateien können der CD entnommen werden, die dieser Arbeit beiliegt.

5.2.2 Sicherung der Code-Qualität

Innerhalb von Programmiersprachen können Befehle auf unterschiedliche Arten formatiert werden. Da bei einer Weiterentwicklung der Anwendung verschiedene Personen mit verschiedenen Hintergründen beteiligt sein können ist anzunehmen, dass auch verschiedene Präferenzen für die Formatierung von Code bestehen werden. Häufige Stilwechsel können für die Lesbarkeit des Quellcodes, gerade für Personen, die sich zum ersten Mal mit diesem befassen, jedoch nachteilig sein. Ein einheitlicher Stil innerhalb des Quellcodes ist daher erstrebenswert. Jedoch kann nicht davon ausgegangen werden, dass sich alle Personen, die an der Mitarbeit der Anwendung beteiligt sind, lange mit dieser befassen. Unter umständen wollen diese nur einen kleinen Fehler beheben und sich nicht lange mit Richtlinien für die Formatierung des Codes auseinandersetzen.

Aus diesem Grund wurde ein *Linter* in die Anwendung integriert. Dieser analysiert den Quellcode auf die Einhaltung von vorher definierten Regeln. Zu diesen Regeln gehören

[1] Siehe hierzu `https://help.github.com/articles/helping-people-contribute-to-your-project/`, zuletzt abgerufen am 13.08.2017

[2] Ein *Issue* kann genutzt werden, um Probleme oder Verbesserungen an einer zentralen Stelle zu diskutieren

beispielsweise die Überprüfung der Einrückung oder die Schreibweise von Funktionen. Wird währen der Entwicklung gegen diese Regeln verstoßen, gibt der *Linter* eine Fehlermeldung aus. Kleinere Fehler, wie zum Beispiel das inkorrekte Einrücken werden dabei automatisch korrigiert.

Um den Zustand des Codes außerdem an einer zentralen Stelle überprüfen zu können, wurde der Service *Travis CI*[3] in das Repository eingebunden. Dieser erlaubt das ausführen von bestimmten Skripten bei erstellen eines neuen Pull Request und das anzeigen der Ergebnisse dieser Skripte. So kann die Einhaltung der definierten Regeln auf der Plattform Github automatisch überprüft werden.

5.3 Vermarktung

Im Kapitel Weiterentwicklung wurde die Rolle des Bekanntheitsgrades einer Anwendung bereits verdeutlicht. Obwohl die Bekanntmachung der Anwendung nicht explizit Teil dieser Arbeit ist, soll dieser Bereich trotzdem kurz angesprochen werden.
Aufgrund der Zielgruppe ist die naheliegendste und erfolgsversprechendste Strategie eine Bekanntmachung direkt an der Technischen Hochschule Köln, vorrangig am Campus Gummersbach. Hier bieten sich verschiedene Möglichkeiten im Internet an, aber auch Veranstaltungen am Campus selbst. Vor allem der *Medieninformatik Showcase* eignet sich hier. Im weiteren Rahmen eigenen sich aber auch Informelle Vorträge oder sogenannte *Meetups* für eine Vorstellung der Anwendung.

[3] `https://travis-ci.org/`

Kapitel 6

Fazit

In der vorliegenden Arbeit konnte die Umsetzung einer marktfähigen Anwendung auf Basis von bereits bestehenden Konzepten und einem Proof of Concept gezeigt werden. Dabei konnten die verschiedene theoretischen und praktischen Schritte, die für eine solche Umsetzung nötig sind, genauer betrachtet werden. Es konnte ein Einblick in das relativ junge Feld der JavaScript Single Page Applications und die Entwicklung mit diesen gegeben werden. Auf die Gestaltung einer Anwendung, die zur Gestaltung dient, konnte eingegangen werden, wie auch auf die nötigen Schritte, um eine Anwendung zur Nutzung und zur Weiterentwicklung verfügbar zu machen.

Der Zielerreichungsgrad für die Umsetzung der Anwendung lässt sich gut an den in Kapitel 1.2 definierten Zielsetzungen feststellen:

Die Anwendung bietet, sowohl durch ihre Gestaltung und Programmierung, als auch durch ihren Aufbau eine befriedigende Nutzererfahrung. Dem Nutzer ist es möglich, der Anwendung seine Zielmedium mitzuteilen und die Anwendung vermittelt auf dieses Medium optimierte Inhalte. Weiterhin erhält der Nutzer durch die Möglichkeit, seine Ergebnisse als PDF-Datei zu speichern einen langfristigen Mehrwert aus der Benutzung der Anwendung.

Durch die Auslieferung der Seite durch einen Server und die Verwendung einer prägnanten URL ist die Anwendung für ihre Nutzer einfach zugänglich. Dies gilt durch die Veröffentlichung des Quellendes der Anwendung weiterhin für Personen, die an der Weiterentwicklung der Anwendung interessiert sind.

Im Bezug auf die Bekanntheit der Anwendung gestaltet es sich schwierig, den Grad der Zielerreichung festzustellen, da die Anwendung etwa Zeitgleich mit Fertigstellung dieser Arbeit veröffentlicht wurde. Es lässt sich jedoch feststellen, dass die Grundlagen, sowohl für eine Bekanntmachung der Anwendung, also auch für eine Mitarbeit der Community an der Anwendung geschaffen wurden.

An dieser Stelle sei außerdem noch einmal der geplante Umfang der Anwendung angesprochen, der nicht komplett umgesetzt werden konnte. Wie in Kapitel 2.5.1 angesprochen wurde hier auf die Implementierung eines Teils der Anwendung verzichtet, der maximal einen prototypischen Stand hat, um die Qualität der Anwendung als ganzes auf einem hohen Niveau zu halten.

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, jedoch bieten sich hier noch einige Möglichkeiten zur Weiterentwicklung.

In den vergangenen Kapiteln wurden bereits Möglichkeiten angesprochen, mit denen der aktuelle Stand der Anwendung weiter verbessert werden kann (beispielsweise durch eine Zielgerichtetere Aufbereitung der PDF-Datei). Es bieten sich aber auch Erweiterungen in völlig neuen Themengebieten an. Dabei kann es sich sowohl um bereits konzipierte Themengebiete (siehe Kapitel 2.1) handeln, aber auch um solche, die bisher nicht bedacht wurden. Mit einer zunehmenden Zahl von Nutzern ist hier auch mit einer zunehmenden Zahl von neuen Anforderung zu rechnen.

Abschließend lässt sich also feststellen, dass das Ziel, eine marktfähige Webanwendung zu entwickeln, durchaus als erreicht angesehen werden kann. Konzepte, die theoretisch eine Nützlichkeit aufweisen konnten damit in eine Anwendung überführt werden, die einen tatsächlichen praktischen Nutzen aufweist.

Literaturverzeichnis

[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.

[Cur16] Nathan Curtis. Space in design systems, 2016. https://medium.com/eightshapes-llc/space-in-design-systems-188bcbae0d62 Abgerufen am 08.08.2017.

[DAH01] Luca De Alfaro and Thomas A Henzinger. Interface theories for component-based design. In *EMSOFT*, volume 1, pages 148–165. Springer, 2001.

[Dud06] Wissenschaftlicher Rat Dudenredaktion, editor. *Der Duden in 12 Bänden. Das Standardwerk zur deutschen Sprache / Die deutsche Rechtschreibung: Das umfassende Standardwerk auf der Grundlage der neuen amtlichen Regeln.* Bibliographisches Institut, 24., völlig neu bearb. u. erw. edition, 2006.

[Eva] Evan You. Vue.js. Version 2.4.2, abgerufen über https://vuejs.org/ am 11.08.2017.

[Fac] Facebook Inc. React.js. Version 15.6.1, abgerufen über https://facebook.github.io/react/ am 11.08.2017.

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

[Goo] Google Inc. Angular.js. Version 4.3.4, abgerufen über https://angularjs.org/ am 11.08.2017.

[Goo02] Danny Goodman. *Dynamic HTML: The Definitive Reference: A Comprehensive Resource for HTML, CSS, DOM & JavaScript*. Ö'Reilly Media, Inc.", 2002.

[Goo09] Kim Goodwin. *Designing for the Digital Age: How to Create Human-Centered Products and Services*. John Wiley & Sons, 1 edition, 3 2009.

[Inc16a] Facebook Inc. Components and props, 2016. `https://facebook.github.io/react/docs/components-and-props.html`, zuletzt abgerufen am 11.08.2017.

[Inc16b] Facebook Inc. Jsx in depth, 2016. `https://facebook.github.io/react/docs/jsx-in-depth.html`.

[Inc16c] Facebook Inc. React.component, 2016. `https://facebook.github.io/react/docs/react-component.html`.

[Jaz07] Mehdi Jazayeri. Some trends in web application development. In *2007 Future of Software Engineering*, FOSE '07, pages 199–213, Washington, DC, USA, 2007. IEEE Computer Society.

[JS17a] Vue JS. Introduction - vue.js, 2017. `https://vuejs.org/v2/guide/`.

[JS17b] Vue JS. Template syntax - vue.js, 2017. `https://vuejs.org/v2/guide/syntax.html`.

[Kru14] Steve Krug. *Don't make me think!: Web Usability: Das intuitive Web (mitp Business)*. mitp, 3., überarbeitete auflage 2014 edition, 10 2014.

[Law08] George Lawton. New ways to build rich internet applications. *Computer*, 41(8), 2008.

[LFDB06] Gitte Lindgaard, Gary Fernandes, Cathy Dudek, and Judith Brown. Attention web designers: You have 50 milliseconds to make a good first impression! *Behaviour & information technology*, 25(2):115–126, 2006.

[MP13] Michael Mikowski and Josh Powell. *Single Page Web Applications: JavaScript end-to-end*. Manning Publications, 1 edition, 9 2013. Abgerufen über `https://livebook.manning.com/#!/book/single-page-web-applications/chapter-1/` am 11.08.2017.

[Pau05] Linda Dailey Paulson. Building rich web applications with ajax. *Computer*, 38(10):14–17, 2005.

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

[Red16a] Redux. Core concepts - redux, 2016. `http://redux.js.org/docs/introduction/CoreConcepts.html`, zuletzt abgerufen am 11.08.2017.

[Red16b] Redux. Motivation - redux, 2016. `http://redux.js.org/docs/introduction/Motivation.html`, zuletzt abgerufen am 11.08.2017.

[Rob17] Jonathan Robie. What is the document object model?, 2017. `https://www.w3.org/TR/WD-DOM/introduction.html`.

[Szy02] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming (2nd Edition)*. Addison-Wesley Professional, 2 edition, 11 2002.

[TC13] Elika J. Etemad Tantek Celik. Css style attributes, 2013. `https://www.w3.org/TR/css-style-attr/#syntax`, zuletzt abgerufen am 12.08.2017.

[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.