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

Kapitel 1

Einleitung

Diese Arbeit beschäftigt sich mit der Umsetzung der Webanwendung *25knots*, die eine Verbesserung der gestalterischen Qualität von Artefakten im Hochschulkontext sichern soll. Das Konzept für die Anwendung wurde vorhergehend bereits im Praxisprojekt erarbeitet und beispielhaft in Form eines Proof of Concept umgesetzt. Im Rahmen der Abschlussarbeit soll nun, aufbauend auf den erarbeiteten Konzepten und dem Proof of Concept, ein marktfähiges Produkt erstellt werden. Dieses Produkt soll weiterhin von der Community[1]nicht nur verwendet, sondern auch aktiv weiterentwickelt werden können.

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

1.1 Motivation

Wie bereits erwähnt wurde das Konzept für die Anwendung bereits im Praxisprojekt erstellt. Dabei wurde viel Zeit und Energie investiert um sicher zu stellen, dass diese Konzepte auch

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

Umsetzbar sind. So wurden zum Beispiel bereits viele (grobe) Algorithmen entworfen, die in der Anwendung verwendet werden können. Des weiteren konnte in persönlichen Gesprächen mit verscheidenen Personen innerhalb und auch außerhalb des Hochscholkontextes festgestellt werden dass eine Umsetzung der erarbeiteten Konzepte durchaus eine Relevanz besitzt und von Personen verwendet werden würde.

Die Motivation für diese Arbeit ergibt sich daher aus dem Willen, aus diesem erarbeiteten Konzept einen Mehrwert schaffen zu wollen, der über das verdienen von Credit Points hinaus geht. Die generelle Motivation für eine Anwendung wie *25knots* lässt sich Kapitel *1.3: Relevanz des Themas* entnehmen

1.2 Zielsetzung

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

1. bietet eine befriedigende Nutzererfahrung
2. ist öffentlich 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 Anwendung wird sich mit den Umsetzungen dieser Bereiche beschäftigen.

Der einfachste Weg, um eine Webanwendung öffentlich zugänglich zu machen ist in der Regel, diese auf einem Server zu hosten. Weiter details dieser Entscheidung werden im Kapitel *Hosting* erläutert.

Damit diese genutzt werden kann, müssen potentielle Nutzer der Anwendung von dessen Existenz wissen. Mit Blick auf die Zielgruppe bietet sich zunächst eine Bewerbung direkt an der Hochschule, beispielsweise durch die Teilnahme am *Medieninfromatik Showcase* an. Aber auch

eine Bewerbung im größeren Rahmen, beispielsweise durch einen Talk beim *Webmontag Köln* ist denkbar.

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

Aus diesen Punkten kann eine zentrale Forschungsfrage für die Arbeit formuliert werden:
Wie kann der Community ein nutzbares und erweiterbares Produkt auf Basis eines Konzeptes und Prototypen bereitgestellt werden?

Diese lässt sich in zwei Unterfragen unterteilen, die es 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?

1.3 Relevanz des Themas

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

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

Gestützt wird dieser Gedanke durch Studien von [**?**], [**?**], und [**?**].

Unterstützend seien hier noch zwei weitere Quellen aufgeführt, die die Relevanz weiter unterstreichen: [**?**] 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 [**?**].

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

1.4 Abgrenzung

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, dass als fertig angesehen werden kann. Auch mit Blick auf die spätere Weiterentwicklung durch die Community muss es viel mehr das Ziel sein, eine hochwertige erste Version des Produktes, die als Grundlage für weitere Features dient, also ein *Minimum Viable Product* zu erstellen.

1.5 Zielgruppe

Während der Konzeption im Praxisprojekt wurden Studenten der Technischen Hochschule Köln im Studiengang Medieninformatik als Zielgruppe festgelegt. Es wurde aber bereits im Praxisprojekt deutlich, dass diese Zielgruppe leicht erweiterbar ist. Somit kann jede Person einen Mehrwert aus diesem Tool ziehen, der ein Artefakt erstellen muss, dessen Hauptaugenmerk eigentlich nicht auf der Gestaltung, sondern auf einer bestimmten Funktion liegt. Das kann im Studium an einer mangelnden Bewertung oder in der Wirtschaft an einem mangelndem Budget liegen, jedoch entstehen häufig Situationen, in denen eine grundlegend solide Gestaltung nicht als wichtig erachtet wird, jedoch 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.

1.6 Struktur der vorliegenden Arbeit

Außerhalb dieser Einleitung Teilt sich die Arbeit in drei weitere Kapitel. Im zweiten Kapitel werden zunächst einige Theoretische Grundlagen erläutert und Entscheidungen auf einer taktischen Ebene erläutert. Im dritten Kapitel werden einige Aspekte der Umsetzung der Anwendung erläutert, während im vierten Kapitel einige Fragen bezüglich der Veröffentlichung der Anwendung beantwortet werden.

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

Kapitel 2

Theoretische Grundlagen

Hier fehlt noch eine generelle Einleitung zum Kapitel

2.1 Struktur der Anwendung

In diesem Abschnitt soll zunächst die generelle Struktur der Anwendung definiert werden. Weite Teile der Struktur können dabei aus dem Praxisprojekt übernommen werden. Im Praxisprojekt wurden die folgenden Themengebiete behandelt:

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

Nach einer erneuten evaluation der Ergebnisse des Praxisprojektes konnten die in der Abschlussarbeit zu behandelnden Themengebiete auf zunächst drei eingegrenzt werden (der Bereich *Layout & Struktur* wurde dabei in *Layout & Grids* umbenannt):

- Typographie

- Layout & Grids
- Farben

Diese Abgrenzung begründet sich auf verschiedene Weisen. Im Fall des Bereiches *Whitespace* konnte im Rahmen des Praxisprojektes kein zufriedenstellendes Konzept erarbeitet werden, wodurch sich dieser Bereich per se nicht für eine Umsetzung eignet. Weiterhin muss es Ziel dieser Arbeit sein, am Schluss eine vollständige Anwendung zu erhalten. Um dieses Ziel im zeitlichen Rahmen erreichen zu können mussten weitere Themengebiete vernachlässigt werden. Hier boten sich die Bereiche *Bilder* und *Interaktive Elemente* an, da diese für die gestalterische Grundqualität eine vergleichsweise niedrige Rolle spielen. **NOTE: Vielleicht muss das hier noch belegt werden, wie ein Brötchen**. Diese Bereiche wurden aber ausreichend konzeptioniert und bieten sich als erste Erweiterungen für die Anwendung nach beenden der Arbeit an.

Außerdem müssen für die Anwendung jeweils ein nutzerfreundlicher Einstieg und Ausstieg gefunden werden. Diese wurden im Praxisprojekt nicht explizit ausgearbeitet und fallen somit auch Konzeptionell in den Bereich der Abschlussarbeit und werden später in diesem Kapitel behandelt.

Die finale Struktur der Anwendung für den Rahmen dieser Arbeit sieht also wie folgt aus:

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

2.1.1 Einstieg in die Anwendung

Bereits im Praxisprojekt wurde festgestellt, dass es sinnvoll ist, das Zielmedium des Nutzers zu kennen. Mit Blick auf die Zielgruppe wurden hier drei mögliche Bereiche definiert: Native App, Website und Textdokument. **NOTE: Hier verweis auf Untersuchung im PP**. Diese Bereiche können jedoch auch in sich verschiedene Eigenarten aufweisen, so kann ein Textdokument beispielsweise für das Lesen an einem Bildschirm oder das Lesen in gedruckter Form entworfen

werden. Eine komplette Auflistung der möglichen Bereiche oder *scopes* der Anwendung findet sich in Abbildung 2.1 auf Seite 10.

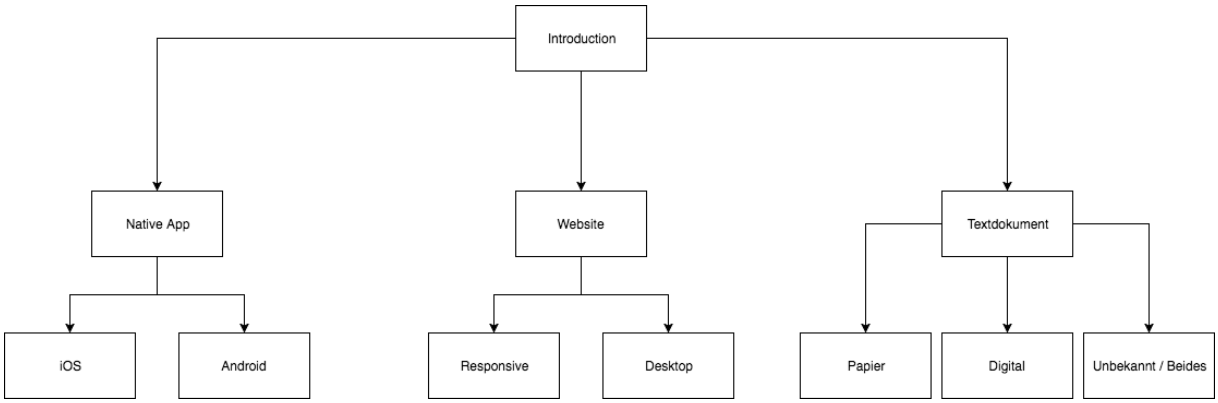

Abbildung 2.1: Mögliche Entscheidungen im Einstieg der Anwendung

Obwohl die Abgrenzung der Bereiche für die hier definierte Zielgruppe ausreichend ist, lassen sich bereits jetzt 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 *Responnsive* und *Desktop* für ein Echtwelt-Szenario unter Umständen zu allgemein gehalten.

Um den kognitiven Aufwand **Beleg** für den Nutzer möglichst gering zu halten, bietet es sich an, ihn Schrittweise durch das Festlegen des für ihn passenden Bereiches zu führen.

2.1.2 Ergebnisse der Benutzung

Eines der Ziele der Anwendung ist es, dem Nutzer währen der Nutzung auf interaktive weise Wissen zu vermitteln. Da der Nutzer jedoch während der Nutzung auch konkrete Ergebnisse erarbeitet wäre es hier kontraproduktiv, ihm diese Ergebnisse nicht am Ende der Anwendung noch einmal explizit zukommen zu lassen (zusätzlich zum implizit gesammelten Wissen).

Dieser Bereich wurde im Rahmen des Praxisprojektes nicht ausdefiniert, ist aber für die Wahrnehmung der Anwendung als fertiges Produkt durchaus wichtig. Eine gute Darstellung der Ergebnisse des Nutzers definieren einen ausschlaggebenden Teil der Nutzungserfahrung. Hier sollen also mögliche Darstellungen der Ergebnisse diskutiert werden und vorrangig zwei Fragen beantwortet werden:

1. Welche Darstellung der Ergebnisse ist für den Nutzer am Vorteilhaftesten?

2. Welche Darstellung bietet den besten Kompromiss aus Umsetzbarkeit und Mehrwert für den Nutzer?

Die einfachste Darstellung ist eine Transistente Darstellung innerhalb der Anwendung am Ende der Verwendung. Eine Darstellung ist wegen der Haltung der ermittelten Werte im Redux-Store der Anwendung einfach. Obwohl einfach umzusetzen, ist diese Lösung nicht optimal: Nachdem der Nutzer die Anwendung schließt sind die erarbeiteten Daten verloren, da diese nicht persistent gespeichert werden.

Ein naheliegender Schritt ist also eine Persitierung des Wissens für den Nutzer. Hier bieten sich verschiedene Möglichkeiten, wie zum Beispiel das Speichern in Cookies oder das Entwicklen eines Backends, an. Ein Kompromiss zwischen Usability und Entwicklungsaufwand wäre hierbei die persistieren des Wissen in einer herunterladbarer Datei, beispielsweise als PDF.

Eine weitere Frage beschäftigt sich mit dem Aufbau dieser Datei. Auch hier wäre der einfachste Ansatz, die Ergebnisse des Nutzers einfach aufzulisten. Optimal wäre eine Aufbereitung der Daten, sodass der Nutzer diese möglichst ohne weitere Manipulation in seinen Workflow übernehmen kann. Obwohl das Zielmedium des Nutzers bekannt ist, zeigt sich hier das Problem, dass innerhalb dieser Zielmedien weiterhin verschieden Tools verwendet werden können, die eine unterschiedliche Aufbereitung der Daten erfordern. Beispielsweise kann bekannt sein, dass der Nutzer eine Webandwendung entwickelt und das Styling für seine texte in CSS vornimmt. Trotzdem kann der Nutzer zum Beispiel verschieden Preprozessoren wie SCSS, SASS oder LESS verwenden, die alle eine unterschiedliche Syntax verwenden. Es liegt dabei durchaus im Rahmen des Möglichen, diese Informationen vom Nutzer zu erhalten und die Daten entsprechend aufzubereiten, jedoch liegen diese Anforderungen außerhalb des Zeitlichen Rahmens dieser Abschlussarbeit.

Hier wird dem Nutzer daher zunächst eine PDF zur Verfügung gestellt. Ein exemplarischer Aufbau kann ABB. XYZ entnommen werden.

2.2 Diskussion verfügbarer Technologien

Im Nachfolgenden Kapitel sollen mögliche Technologien diskutiert werden, die für die Umsetzung der Abschlussarbeit genutzt werden können. Für die Umsetzung einer Webanwendung

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

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

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

2.2.1 Vue.js

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

2.2.2 Angular.js

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

2.2.3 React.js

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

2.3 Einstieg in React.js

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

2.3.1 Komponenten

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

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

2.3.1.1 Was ist eine Komponente?

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

Das Oxford Dictionary definiert eine Komponente als

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

Gerade im Bezug auf Softwareentwicklung werden Komponenten aber noch einige weitere Eigenschaften zugeschrieben. So schreiben [Szy02] über Komponenten in der Softwareentwicklung:

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

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

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

2.3.1.2 Komponenten in React.js

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

1. Mounting
2. Updating
3. Unmounting

Mounting beschreibt dabei den Vorgang des erstellen der Komponente und des einfügens in

den DOM. Funktionen, die beim Mounting aufgerufen werden, werden also nur ein einziges mal im gesamten Lifecycle aufgerufen (die Ausnahme bildet dabei die Funktion `render()`, die auch beim Update Ereignis aufgerufen wird).

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

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

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

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

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

```
class Parent extends REact.Component {
        calculateSomething() {
                // Claculates something
                return result
        }

        render() {
                let result = this.calculateSomething()

                return (
                        <Child value={result} />
```

```
            )
        }
    }

    class Child extends React.Component {
        render() {
            return (
                <div>{this.props.value}</div>
            )
        }
    }
```

Das Prinzip des `state` wird im Kapitel Stateless & Stateful auf Seite 16 näher erläutert.

2.3.1.3 Stateful & Stateless

Komponenten in React.js können entweder Stateful oder Stateless sein. Wie der Name bereits vermuten lässt, haben diese Komponenten entweder einen state, oder nicht.
Der state in React.js beschreibt den Zustand eine Komponente. Hier findet sich auch eines der Hauptfeatures von React.js: Ändert sich der state einer Komponente, so wird diese Komponente neu gerendert, also auch das User Interface aktualisiert um den neuen Zustand darzustellen. Der state kann (und sollte) dabei durch das Aufrufen der Funktion `setState()` geändert werden. Das nachfolgende Codebeispiel zeigt eine simple Komponente, die clicks auf einen Button zählt:

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

        this.state = {clicks: 0}
    }

  handleClick() {
        this.setState((prevState) => {
```

```
                return {clicks: prevState.clicks + 1};
            });
    }

    render() {
        return (
            <button onClick={this.handleClick}>Click me!</b
            {this.state.clicks}
        )
    }
}
```

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

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

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

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

und

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

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

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

                this.state = {clicks: 0}
        }

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

        render() {
```

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

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

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

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

2.3.2 JSX

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

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

JSX unterstütz die Deklaration von regulären HTML-Elementen, als auch die von React-Komponenten. Unterschieden werden die beiden Varianten dabei nach Groß- und Kleinschreibung, wobei React-Komponenten groß- und reguläre HTML-Elemente klein geschrieben sind. React-Komponenten können dabei, wie oben bereits erwähnt, `props` mitgegeben werden. Der Schritt von HTML zu JSX stellt in der Entwicklung keine große Herausforderung dar. Der wichtigste Punkt ist die Verwendung von `class`. Auch bei der Entwicklung mit React werden für DOM-Elemente

häufig Klassen vergeben. Da JSX-Code aber in JavaScript kompiliert wird, kann das in JavaScript reservierte Wort[3] `class` nicht verwendet werden. Es muss stattdessen auf `className` zurück gegriffen werden.

2.4 Einstieg in Redux

Redux[4] erlaubt die zentrale Verwaltung des state bzw. des Zustandes der Anwendung im sogenannten redux store. Dabei ersetzt der redux store nicht zwangsweise den state von einzelnen Komponenten. Es gibt keine genauen Richtlinien dafür, wann ein state in der Komponente und wann im redux store verwaltet werden sollte, jedoch bietet es sich als Richtlinie an, nur Zustände im redux store zu verwalten, die von mehr als nur einer einzigen Komponente verwendet werden. Redux besteh dabei aus drei Grundbestandteilen: Dem *Store*, den *Actions* und den *Reducern*.

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

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

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

Ein *Provider* macht den *Store* dann für Container-Komponenten verfügbar. Diese verteilen den *Store* und die actions als props an representative Komponenten. Ändert sich der *Store* erhalten die entsprechen betroffenen Komponenten also neue *props* und werden somit neu gerendert.

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

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

Abb XYZ zeigt einen schematischen Ablauf einer Aktualisierung des redux stores.

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

2.5 Komponentenbasierte Gestaltung

Bei einer so stark komponentenbasierten Entwicklung ergibt auch eine Anpassung des Design-Prozesse an diese Strategie Sinn. Dabei ist jedoch ein rein komponentenbasierter Designprozess nicht die Optimale Lösung, denn Aspekte wie Interaktion im Gesamtkontext der Anwendung und auch gestalterisches Zusammenspiel der verschiedenen Komponenten müssen getestet werden. Daher werden zunächst Wireframes mit Hilfe des Tool UXPin[5]erstellt, um die allgemeine Struktur der Anwendung zu entwerfen. Auf basis dieser Wireframes können dann bereits einzelne Komponenten definiert werden, die anschließend Gestaltet werden können. Jedoch muss die endgültige Gestaltung dieser Komponenten im Gesamtverbund einer Seite erfolgen, um garantieren zu können, dass die Komponenten untereinander harmonieren.

Bei einer Komponentenbasierten Gestaltung (und gerade auch mit Blick auf die Weiterentwicklung durch die Community) bietet sich außerdem immer ein Styleguide an, der einen überblick über die verschiedenen Komponenten gibt. Dieser Styleguide soll in diesem Projekt in Form von Storybook[6]abgebildet werden. Storybook bietet dabei nicht nur die Möglichkeit, einen überblick über die vorhandenen Komponenten zu bilden, sonder erlaubt es auch, den Code direkt aus der Anwendung einzubinden. Das heißt, es muss nicht extra für den Styleguide doppelter Code beschreiben werden. Weiterhin können Komponenten zunächst in Storybook entwickelt werden, der Fokus kann also rein auf der zu entwickelnden Komponente liegen und es muss nicht auf die umliegende Anwendung geachtet werden.

[HIER NOCH BILD]

[5]urlhttps://www.uxpin.com/

[6]urlhttps://storybook.js.org/

Kapitel 3

Entwicklung der Anwendung

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

3.1 Gestaltung

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

3.1.1 Wireframes & Struktur

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

eingebundenen Objekten wie Bildern, Formeln

3.1.2 High Fidelity Mockups

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

3.2 Besonderhieten im Technologie-Stack

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

3.2.1 Redux

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

3.2.2 React Storybooks

Bei der Textgestaltung und automatischen Änderung von Abbildungsnummern, Querverweisen, Seitenzahlen, Gliederungen, Literaturhinweisen etc. bietet sich der Rückgriff auf moderne Textverarbeitungsprogramme an. Nutzen Sie diese zur besseren Lesbarkeit und Strukturierung

des Textes, aber vermeiden Sie überflüssige Spielereien. Da besonders bei Textdokumenten mit eingebundenen Objekten wie Bildern, Formeln

3.2.3 CSS-Architektur

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

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

```
<ReactComponent className='myClass'>
  Content
</ReactComponent>
```

Die kompilierte Dom-Node wäre dabei

```
<ReactComponent className='myClass'>
  Content
</ReactComponent>
```

Und könnte im Stylesheet über

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

Angesprochen werden. Auch die Verwendung von Methodiken wie BEM oder SMACCS ist mit diesem Ansatz ohne Probleme möglich. Für den Rahmen dieses Projektes wurde sich jedoch gegen diese Vorgehensweise entschieden, da bewusst ein Fokus auf eine komponentenbasierte

Architektur gelegt werden sollte. Auch mit einer komponentenbasierten CSS-Architektur ist eine Komponente immer noch auf zwei Orte aufgeteilt: Die Funktion und das Markup, und das Styling.

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

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

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

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

Auf den ersten Blick wirkt die Verwendung von inline-styles problematisch, vielleicht weil diese auf statischen Seiten viele Nachteile mit sich bringen. In einem Komponentenbasierten System wie React.js sind diese Nachteile jedoch nicht present. Durch die Komponentenbasierte Struktur und das damit einhergehende Ziel der Wiederverwendung von Komponenten, müssen Stylingänderungen auch hier nur an einer Stelle vorgenommen werden. Da jede Komponente nur für ihr eigenes Styling verantwortlich ist und nicht für das von Kindern, und auch kein CSS außerhalb der inline-styles verwendet wird (abgesehen von CSS-Reset und einigen globalen Regeln wie der Schriftart), kann es zu keinen Problemen mit der Spezifität von CSS-Regeln kommen.

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

[HIER NOCH VERWEIS AUF SPEZIFIKATION]

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

```
import React from 'react'
import { StyleSheet, css } from 'aphrodite'

function myComponent(props) {
      <div className={css(styles.componentStyles)} >
            {props.children}
      </div>

      const styles = StyleSheet.create({
            componentStyles: {
                  color: 'blue'
            }
      })
}
```

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

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

3.3 Algorythmen

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

3.4 Tests

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

Kapitel 4

Veröffentlichung der Anwendung

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

4.1 Hosting

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

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

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

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

```
const express = require('express')
const app = express()

app.use(express.static(__dirname + '/dist'))
app.listen(process.env.PORT || 8080)
```

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

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

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

4.2 Weiterentwicklung

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

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

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

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

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

4.2.1 Sicherung der Code-Qualität

Grober Aufbau für diese Sektion:

1. Travis-CI
2. es-lint

4.2.2 Dokumentation

Grober Aufbau für diese Sektion:

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

4.2.3 Contribution Guidelines

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

4.2.4 Tests

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

4.3 Vermarktung

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

Kapitel 5

Ausblick

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

Kapitel 6

Schluss

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

6.1 Fazit

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

Literaturverzeichnis

[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.

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

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

[Inc16a] Facebook Inc. Jsx in depth, 2016.

[Inc16b] Facebook Inc. React.component, 2016.

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