MASTERARBEIT

PORTIERUNG VON WEBTECHNOLOGIEN

im Frontend-Bereich

ausgeführt am

Studiengang

Informationstechnologien und Wirtschaftsinformatik

Von: Markos Giakoumis

Personenkennzeichen: 1910320003

Graz, am 19. November 2020 .....

Unterschrift

EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

.....

Unterschrift

DANKSAGUNG

An dieser Stelle möchte ich mich bei allen bedanken, die mir beim Schreiben dieser Arbeit geholfen haben.

Zuerst möchte ich mich bei meinem Betreuer Herrn Dipl.-Ing. (FH) Günther Zwetti bedanken, der mich stets unterstützt hat. Ohne seinen Input bereits bei der Themenwahl wäre diese Arbeit nicht zustande gekommen. Sein konstruktives Feedback und seine zahlreichen Anregungen haben mir stets beim Schreiben dieser Arbeit weitergeholfen und mich auf den richtigen Weg gebracht.

Besonders möchte ich mich auch bei meiner Freundin bedanken, die stets ein offenes Ohr für mich hatte und immer für mich da war.

Abschließend möchte ich mich bei meiner Mutter für die bedingungslose Unterstützung bedanken.

KURZFASSUNG

Das Technologieumfeld der Webentwicklung hat sich in den letzten fünf Jahren stark verändert. So ist die Beliebtheit der weitverbreiteten JavaScript Library jQuery im Verlauf der Jahre gesunken. Die bekanntesten Webtechnologien Angular.js, React.js und Vue.js haben stattdessen an Popularität gewonnen und haben ihre Konkurrenten längst überholt. Da Webtechnologien sich immer schneller weiterentwickeln und ständig neue Technologien auf den Markt kommen, beschäftigt sich diese Arbeit mit der Portierbarkeit sowie der Zukunftssicherheit von Angular.js, React.js und Vue.js.

Dabei wird überprüft, ob die vorgestellten Technologien in die jeweils anderen portierbar sind oder ob bei einem Technologiewechsel eine Neuentwicklung notwendig ist. Als Ziele dieser Arbeit kann neben der Gewinnung wichtiger Erkenntnisse in Bezug auf die Portierungen die Erarbeitung von Hilfestellungen und Richtlinien für Portierungen, sowie eine Bewertung und Empfehlung der ausgewählten Webtechnologien genannt werden.

Um die Portierungen durchzuführen, werden im ersten Schritt einheitliche Anwendungen in Angular.js, React.js und Vue.js nach zuvor definierten Anforderungen entwickelt. Die Anwendungen fokussieren sich auf Formularinhalte. Dabei werden Daten von einem Server geladen, im Frontend dargestellt, bearbeitbar gemacht und wieder an den Server geschickt, um persistent gespeichert zu werden. Jede dieser Anwendungen wird danach in die beiden anderen Technologien portiert und die daraus entstandenen Erkenntnisse dokumentiert. Die abschließende Bewertung der miteinander verglichenen Technologien erfolgt nach den Kriterien: Popularität, Entwicklungsaufwand, Flexibilität, Portierbarkeit und Zukunftssicherheit.

ABSTRACT

The technological environment of web development has changed in the past five years. The popularity of the widely used JavaScript library jQuery has declined of the years. The best-known web technologies Angular.js, React.js, and Vue.js have instead gained popularity and have overtaken their competitors. Since web technologies are developing faster and faster and new technologies are constantly coming onto the market, this thesis focuses on the portability and the future security of Angular.js, React.js, and Vue.js.

For each technology is checked whether the technology can be ported to the others or whether a new development is necessary in case of a technology change. In addition to gaining important knowledge of the portability, the goals of this work include the development of guidelines for porting, as well as an evaluation and recommendation of the selected web technologies.

In order to port the technologies, uniform applications in Angular.js, React.js, and Vue.js are developed with predefined requirements. The applications are focusing on form content. Data is loaded from a server, displayed in the frontend to be edited, and sent back to the server to be saved persistently. Each of these applications is then ported to the other two technologies. The evaluation of the compared technologies is based on the criteria: popularity, development effort, flexibility, portability, and future security.

INHALTSVERZEICHNIS

1 EINLEITUNG

"Was man lernen muss, um es zu tun, das lernt man, indem man es tut." – Aristoteles von Stageira

Die Trends der Softwareentwicklung werden immer schnelllebiger, besonders im Bereich der Webtechnologien. So sind im Verlauf der letzten Jahre eine Vielzahl neuer Frameworks, Libraries und weiterer Webtechnologien auf den Markt gekommen. Dabei zählen Angular/Angular.js, React.js sowie seit 2019 auch Vue.js zu den etabliertesten Technologien bei Entwicklern und Unternehmen (Stack Overflow, 2019).

Die ständige Entwicklung neuer Technologien wirkt sich dabei auf die Lebensdauer der Webanwendungen aus. Was geschieht jedoch, wenn die Technologie gewechselt werden muss? Oftmals bedeutet ein Technologiewechsel in der Softwareentwicklung eine Neuentwicklung der Anwendung oder ein aufwendiges Portieren von Legacy Code (Jha & Maheshwari, 2005).

1.1 Hypothesen und Forschungsfrage

Aufgrund der verkürzten Lebenszyklen von Webanwendungen werden Faktoren wie Wiederverwendbarkeit von Code und Zukunftssicherheit der Technologien immer wichtiger. Die vorliegende Masterarbeit beschäftigt sich mit genau diesen Faktoren und behandelt die folgende Forschungsfrage:

- Welche der Webtechnologien Angular.js, React.js und Vue.js bieten Zukunftssicherheit hinsichtlich Portierbarkeit und Wiederverwendbarkeit?

Es werden die nachfolgenden Hypothesen aufgestellt, die im Rahmen dieser Masterarbeit bearbeitet werden. Dabei wird für jede Webtechnologie eine Hypothese formuliert. An dieser Stelle wird angemerkt, dass grundsätzliche jede Webtechnologie in Form einer Neuentwicklung in andere portiert werden kann. Diese Arbeit beschäftigt sich ausschließlich mit der Wiederverwendbarkeit von Teilen des Programmcodes und richtet sich somit primär an Softwareentwickler, die Webtechnologien portieren möchten, sowie an Personen, die eine Entscheidung über einen Technologiewechsel treffen müssen.

Falls eine Portierung nicht in absehbarer Zeit möglich ist, beispielsweise durch einen anderen Aufbau der zu portierenden Technologie, zählt dies als nicht gelungene Portierung. Auch für den Fall, dass die Portierung nur durch eine Neuentwicklung möglich ist, bei der die bestehende Geschäftslogik in einer anderen Technologie neu aufgebaut wird, zählt dies als nicht gelungene Portierung. Dies wird in den nachfolgenden Hypothesen als nicht portierbar definiert.

Hypothese 1

H0: Angular/Angular.js ist nicht in React.js portierbar
H1: Angular/Angular.js ist in React.js portierbar

Hypothese 2

H0: React.js ist nicht in Vue.js portierbar
H1: React.js ist in Vue.js portierbar

Hypothese 3

H0: Vue.js ist nicht in Angular/Angular.js portierbar
H1: Vue.js ist in Angular/Angular.js portierbar

1.2 Zielsetzung

Diese Masterarbeit hat die folgenden Ziele:

1. Erkenntnisse der Portierungen

   Hier steht im Vordergrund, ob es grundsätzlich möglich ist, die ausgewählten Technologien in absehbarer Zeit und ohne komplette Neuentwicklung zu portieren. Zusätzlich wird festgehalten, ob es signifikante Unterschiede in Bezug auf Aufwände und Komplexität gibt.

2. Hilfestellungen und Richtlinien für Portierungen

   Nach den durchgeführten Portierungen werden mögliche Hilfestellungen und Richtlinien erarbeitet, die einerseits tiefergehende Forschungsansätze aufdecken, sowie andererseits als Hilfestellung für Portierungen der ausgewählten Webtechnologien dienen.

3. Bewertung der Webtechnologien

   Abschließend wird eine Bewertung durchgeführt, um vor allem den Aspekt der Zukunftssicherheit zu behandeln. Dabei wird basierend auf den Ergebnissen eine Empfehlung abgegeben, welche der Webtechnologien Vorteile hinsichtlich der Portierbarkeit aufzeigt.

1.3 Methoden und Aufbau

Zur Überprüfung der Portierbarkeit und der Wiederverwendbarkeit werden einheitliche Webanwendungen in den ausgewählten Webtechnologien entwickelt. Diese Anwendungen werden danach in jeweils die anderen Technologien portiert, mit dem Ziel, soviel Code wie möglich wiederzuverwenden.

Jede Anwendung hat den gleichen Umfang, die gleiche Funktionalität und unterscheiden sich optisch nicht voneinander. Die Portierungen selbst werden als Experiment gesehen. Es wird eine einheitliche Vorgehensweise definiert welche Schritte bei den Portierungen durchgeführt werden.

Um entsprechende Theorie für diese Vorgehensweise zu erarbeiten, wird Literaturrecherche verwendet. Diese Informationen werden im theoretischen Teil dieser Arbeit angeführt.

Im praxisnahen Teil wird auf die genaue Vorgehensweise, das Ergebnis der Portierungen und auf die Bewertung der Ergebnisse eingegangen.

Zu Beginn dieser Arbeit werden aktuelle Trends angeführt, um einen Überblick über die Webentwicklung zu geben, sowie die Entscheidung der gewählten Webtechnologien verständlich zu machen.

2 AKTUELLE TRENDS IN DER WEBENTWICKLUNG

In diesem Kapitel wird auf die technologische Entwicklung der Frameworks und Libraries im Webbereich der letzten Jahre eingegangen. Da diese Arbeit auf Frontend-Technologien basiert, werden nur diese betrachtet. Zudem wird die Bedeutung der ausgewählten Technologien dargestellt.

2.1 Aktuelle Technologien

Die folgende Auflistung dient zur Übersicht von aktuellen Webtechnologien. Es wurde eine Unterscheidung zwischen Frontend- und Backend-Technologie getroffen. Die nachfolgende Auflistung bezieht sich nur auf Frontend-Technologien. Frameworks, die für Frontend- als auch für die Backend-Entwicklung konzipiert wurden, werden nicht angeführt. Der Überblick ist nicht vollständig, soll jedoch ein grobes Verständnis vermitteln. Im nächsten Abschnitt wird darauf eingegangen, warum diese gewählt wurden. Alle der angeführten Libraries und Frameworks basieren auf JavaScript. An dieser Stelle ist JavaScript rein als Programmiersprache zu sehen, auf den genauen Aufbau wird in Abschnitt 4.1 JavaScript eingegangen.

- jQuery

  Die JavaScript Library jQuery wurde im Jahr 2006 veröffentlicht. Die Library hat das Ziel, den Umgang mit JavaScript zu vereinfachen, indem Event Handling, Animationen und Ajax über die Library ausgeführt werden können (jQuery, o. D.).

- React.js

  React.js ist eine JavaScript Library, die von Facebook entwickelt wurde. React.js ist für die Entwicklung von User Interfaces (UI) ausgelegt und wurde erstmals 2013 der Öffentlichkeit zur Verfügung gestellt (React, o. D.).

- Angular/Angular.js

  Angular/Angular.js wurde von Google entwickelt und wurde im Jahr 2010 veröffentlicht. Im Gegensatz zu React.js und jQuery handelt es sich bei Angular/Angular.js um ein Framework (Angular, o. D.). Auf den Unterschied zwischen Angular und Angular.js wird im späteren Verlauf dieser Arbeit eingegangen.

- Vue.js

  Die JavaScript Library Vue.js wurde erstmals 2014 öffentlich verfügbar gemacht. Das Ziel von Vue.js ist es, zugänglich und performant zu sein (Vue.js, o. D.).

- Ember.js

  Die Open-Source Library Ember.js wurde 2011 der Öffentlichkeit zur Verfügung gestellt. Wie React.js ist auch Ember.js auf die Entwicklung von User Interfaces ausgelegt. Ember.js hat dabei vor allem die Entwicklung von skalierbaren Anwendungen als Fokus (Ember, o. D.).

2.2 Umfragen und Statistiken

Im Verlauf des letzten Jahrzehnts hat sich die Landschaft der Frameworks und Libraries in der Webentwicklung stetig verändert und weiterentwickelt. Die Plattform Stack Overflow (Stack Overflow, o. D.) gilt als eine der größten online Gemeinschaften für Softwareentwickler und erstellt jährliche Umfragen zu Beliebtheit von Technologien und Programmiersprachen.

Diese Umfragen werden für diese Arbeit verwendet, um die drei etabliertesten Webtechnologien für die Portierung und Bewertung auszuwählen. Zusätzlich wird Google Trends (Google, o. D.) als weitere Quelle verwendet, um Informationen über die Anzahl der Suchaufrufe zu gewinnen und somit Rückschlüsse auf die Popularität zu ziehen.

2.2.1 Ergebnis der Umfrage von Stack Overflow

Die Umfrage aus dem Jahr 2019 zeigt, dass JavaScript das siebte Jahr in Folge die meistverwendete Programmiersprache der Welt ist. Dies spiegelt sich auch in der Auswahl der verwendeten Webtechnologien bei professionellen Entwicklern wider. Dabei befinden sich jQuery, Angular/Angular.js und React.js auf den ersten drei Plätzen bei der Verwendung im professionellen Umfeld aller Webtechnologien. Betrachtet man nur Webtechnologien aus dem Frontend-Bereich, so befindet sich Vue.js an vierter Stelle hinter React.js.

Bei den Ergebnissen der Beliebtheit der Webtechnologien bei Entwicklern sieht das Ergebnis anders aus. Wie in Abbildung 1 dargestellt, führt React.js mit 21.5%, gefolgt von Vue.js mit 16.1% und Angular/Angular.js mit 12.2%. Die Library jQuery ist dabei mit 5% an fünfter Stelle zu finden (Stack Overflow, 2019).

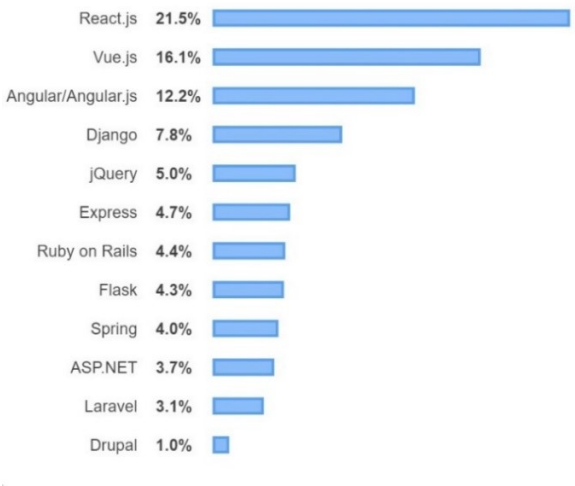

Abbildung 1: Ergebnisse der Umfrage der Beliebtheit von Webtechnologien von Stack Overflow, 2019

2.2.2 Stack Overflow Trends

Stack Overflow bietet einen weiteren Indikator, um die Relevanz von Technologien einzuschätzen. Die sogenannten Trends lassen mittels Schlagworte ein Diagramm ermitteln, das die prozentualen Stack Overflow Fragen des Monats im Laufe der Zeit darstellt. Somit lassen sich Rückschlüsse auf die Verwendung und die Lebenszyklen der Webtechnologien schließen (Stack Overflow, 2020).

Abbildung 2 zeigt in einem Diagramm die Trends der Technologien jQuery, React.js, Angular/Angular.js, Vue.js und Ember.js. Im Diagramm wird deutlich, dass jQuery immer mehr an Bedeutung verliert und React.js im Laufe der letzten Jahre Angular/Angular.js überholt hat. Ember.js hat hingegen in Vergleich zu den anderen Webtechnologien im Jahr 2019 kaum Relevanz.

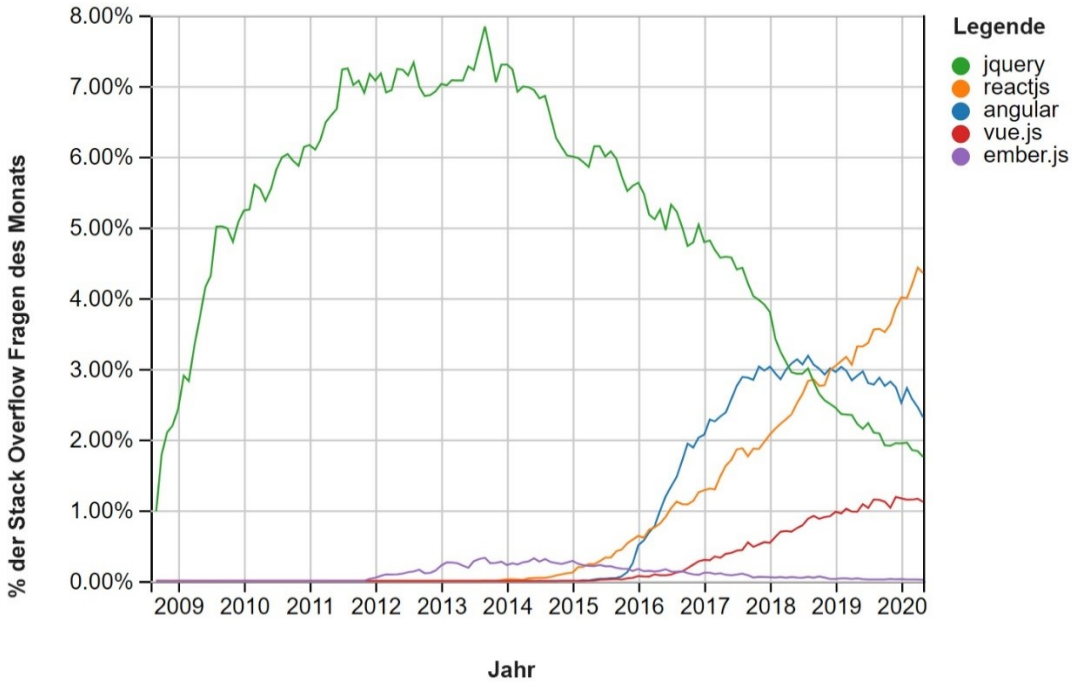

Abbildung 2: Diagramm der Stack Overflow Trends von Frontend-Technologien von Stack Overflow, 2020

2.2.3 Google Trends

Die Plattform Google Trends gibt Auskunft über die Anzahl von Suchanfragen für eine ausgewählte Zeitperiode (Google, o. D.).

In dieser Arbeit wird Google Trends verwendet, um zu überprüfen, ob sich die Ergebnisse der Stack Overflow Trends auch in den Suchanfragen von Google widerspiegeln. Aus diesen Daten werden Rückschlüsse auf das allgemeine Interesse der Technologien gezogen.

Abbildung 3 stellt die prozentualen relativen Suchanfragen in einem Diagramm dar. Wie in Abbildung 2 wird hier der gleiche Zeitraum von Jänner 2009 bis Juni 2020 betrachtet, sowie die Technologien jQuery, React.js, Angular/Angular.js, Vue.js und Ember.js gegenübergestellt.

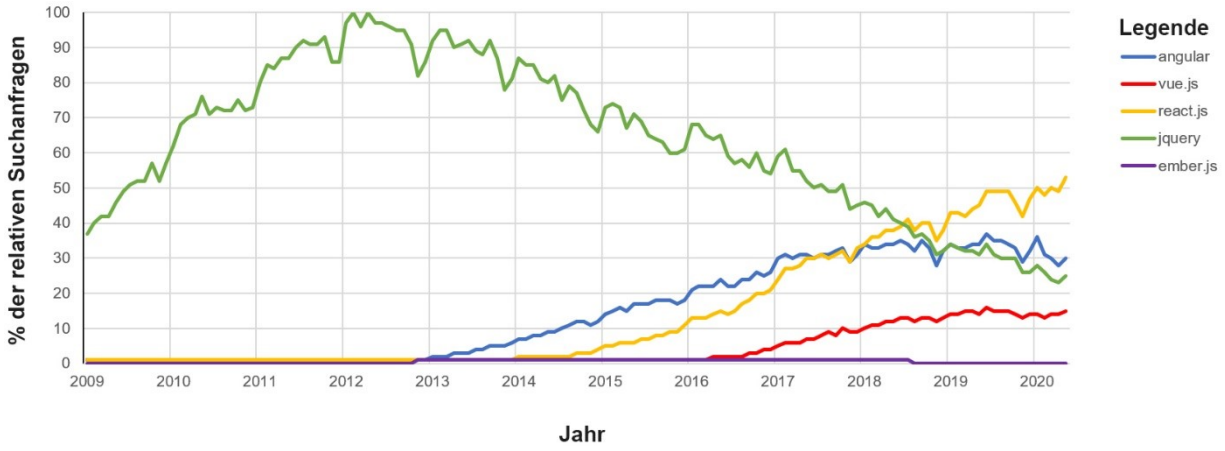

Abbildung 3: Vergleich der Suchanfragen der Webtechnologien auf Google Trends in Anlehnung an Google, o. D.

Die Tendenzen der Stack Overflow Trends spiegeln sich in den Google Trends wider. So zeigt Abbildung 3, dass jQuery im Verlauf der letzten Jahre immer mehr an Bedeutung verliert. Die Suchanfragen für React.js haben im Zeitraum von 2017 bis 2019 die Anfragen von Angular/Angular.js überholt. Die Suchanfragen für Vue.js sind seit 2016 kontinuierlich gestiegen, ähnlich wie in den Stack Overflow Trends.

2.2.4 Auswahl der Technologien

Die Erkenntnisse aus den Umfragen, sowie aus den Trends von Stack Overflow bieten die Basis für die Auswahl der Technologien React.js, Angular/Angular.js und Vue.js dieser Arbeit. Zusätzlich bestätigt die Auswertung von Google Trends die Relevanz der ausgewählten Technologien. Alle drei Webtechnologien sind mit Abstand die relevantesten im Frontend-Bereich. Zudem bauen alle auf JavaScript auf und bieten somit eine einheitliche Basistechnologie für die Portierungen. Da die Verwendung und das Interesse der Library jQuery seit 2014 sinkend ist, wird diese Technologie nicht bewertet.

3 ARCHITEKTUR VON JAVASCRIPT WEBANWENDUNGEN

Dieses Kapitel behandelt die Architektur von JavaScript Webanwendungen, um einen Überblick über den Aufbau zu geben. Da alle drei verwendeten Webtechnologien dieser Arbeit auf JavaScript aufbauen, wird speziell nur auf die Komponenten von JavaScript Webanwendungen eingegangen. Zusätzlich dient dieses Kapitel zur Verdeutlichung, mit welchen Teilen einer Webanwendung sich diese Arbeit beschäftigt.

3.1 Komponenten

In den folgenden Abschnitten wird auf die einzelnen Komponenten von JavaScript Web-Applikationen eingegangen.

3.1.1 Client und Frontend

Der Client ist die Interaktionsschnittstelle zu den Benutzerinnen und Benutzern. Hierbei handelt es sich bei JavaScript Webanwendungen um einen Browser, der beim Aufruf einer Website Hypertext Markup Language (HTML)-Dateien, Cascading Style Sheets (CSS)-Dateien und JavaScript-Code von einem Webserver lädt. Der Browser kann am Desktop PC oder auf mobilen Geräten ausgeführt werden. Dieser kann über das User Interface Input von den Benutzerinnen und Benutzer annehmen, um den entsprechenden Code auszuführen. Der Begriff Frontend, mit dem sich diese Arbeit beschäftigt, umfasst die gesamte clientseitige Anwendung, somit das gesamte UI als auch die Kommunikation zu den anderen Komponenten (Elliot, 2014).

3.1.2 Server, Backend und Services

Wie im Abschnitt zuvor angemerkt, dient ein Server bei JavaScript Webanwendungen zur Auslieferung von HTML-, CSS-, JavaScript-Dateien, sowie anderer Ressourcen wie beispielsweise Bilder (Wagner & Diaconescu, 2017).

Um jedoch Daten persistent speichern zu können, muss eine Schicht die Daten vom Client annehmen und diese in einem Datenspeicher ablegen. Bei JavaScript Anwendungen spricht man hier von sogenannten Webservices, die für die Geschäftslogik und somit auch für die Interaktion zwischen Client und Daten zuständig sind (Brown, 2016).

Der Begriff Backend umfasst dabei das gesamte Konzept von Server, Service und Datenschicht (Elliot, 2014). Auf letztere wird im nächsten Abschnitt eingegangen.

Für diese Arbeit wird das Backend als Black-Box betrachtet, die das Erstellen, das Laden, ein Update und das Löschen von Daten ermöglicht.

3.1.3 Daten

Bei einem Datenspeicher handelt es sich um einen persistenten Speicher, auf dem Daten aus der Anwendung abgelegt werden. Abhängig von der Anwendung können dafür SQL- oder NoSQL-Datenbanken, aber auch reine Textdateien verwendet werden (Brown, 2016). Da für diese Arbeit auch die Datenschicht als Black-Box betrachtet wird, werden an dieser Stelle keine weiteren Informationen angeführt.

3.1.4 Kommunikation der Komponenten

Um den Ablauf und die Interaktion zu verdeutlichen, zeigt Abbildung 4 die Kommunikation zwischen den Komponenten. An dieser Stelle wird angemerkt, dass die Kommunikation von JavaScript Webanwendungen über einen HTTP-Client mittels HTTP-Protokoll geschieht. Als Datenformat hat sich im Laufe der letzten Jahre die JavaScript Object Notation (JSON) im Webbereich durchgesetzt (Elliot, 2014).

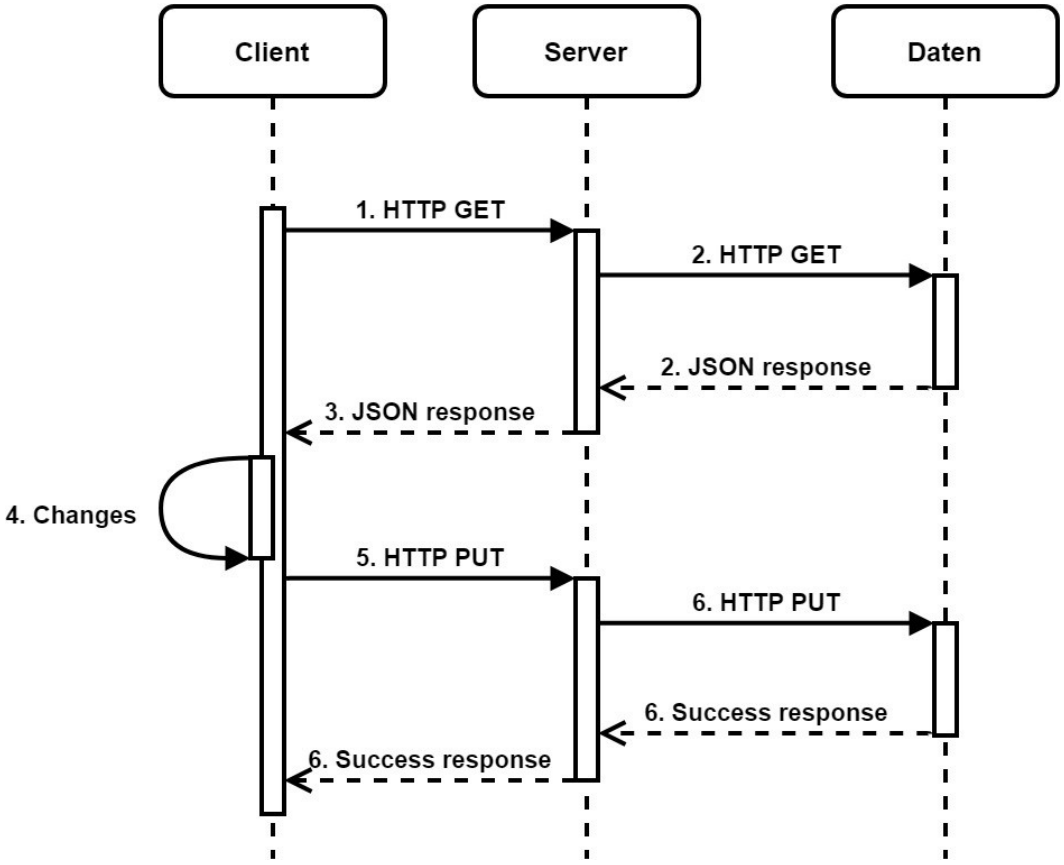

Abbildung 4: Kommunikation der Komponenten von Javascript Webanwendungen in Anlehnung an Elliot, 2014

Abbildung 4 zeigt, wie zu Beginn der Client mittels HTTP GET eine Anfrage an den Server schickt. Der Server nimmt den Request an, kommuniziert mit der Datenschicht und bekommt eine entsprechende Antwort im JSON Format, die danach an den Client weitergegeben wird. Daraufhin werden die Daten im User Interface des Clients verändert und mittels HTTP PUT an den Server weitergeleitet, der wieder den Request an die Datenschicht weitergibt. Diese führt die

Änderung an den entsprechenden Daten durch und gibt die Success Rückmeldung an den Server, der wieder die Meldung an den Client weitergibt (Elliot, 2014).

Das folgende Listing zeigt eine aufgezeichnete Kommunikation des HTTP-Protokolls zwischen einem Frontend und einem Backend, wie in Abbildung 4 dargestellt. Das Frontend hat die Adresse localhost:3001, das Backend localhost:3000. In Listing 1 ist im ersten Absatz der Header der GET Anfrage zu sehen, dabei wird entsprechend definiert an welchen Host der Aufruf geht, welche Datenformate akzeptiert werden sowie weitere Inhalte, die für diese Arbeit jedoch keine Relevanz haben. Im zweiten Absatz ist die Antwort des Servers zu sehen, in diesem Fall eine Success Rückmeldung mit einem Objekt mit dem Eintrag `data` und dem Wert `testString` (Richardson, 2007).

```
GET /input HTTP/1.1
Host: localhost:3000
Connection: keep-alive
Accept: application/json, text/plain, */*
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36
DNT: 1
Origin: http://localhost:3001
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3001/
Accept-Encoding: gzip, deflate, br
Accept-Language: de-DE,de;q=0.9,en-US;q=0.8,en;q=0.7
If-None-Match: W/"1d-fUKNwszO9cg1ykvqjGMes0XHB3c"

HTTP/1.1 200 OK
Content-Type: application/json
{
  "data": "testString"
}
```

Listing 1: HTTP Header und Response eines GET Aufrufs in Anlehnung an Richardson, 2007

3.2 Single Page Applications

Diese Arbeit beschäftigt sich ausschließlich mit sogenannten Single Page Applications (SPA). Bei SPAs handelt es sich um einen Trend in der Webentwicklung, der mit modernen Frontend Libraries und Frameworks populär geworden ist. Der Unterschied zu einer normalen Webanwendung ist, dass bei einem Klick auf einen Link die Seite nicht neugeladen wird. Die gesamte Interaktion mit der Anwendung wird durch ein JavaScript Frontend verarbeitet. Einer der Vorteile von SPAs ist, dass die Logik der UI auf den Client ausgelagert wird und somit Last vom

Netzwerk und Server genommen wird, da diese nicht ständig die Inhalte der Anwendung vom Server laden müssen (Fink & Flatow, 2014).

Da im Rahmen dieser Arbeit nur SPAs entwickelt werden, wird im praktischen Teil dieser Arbeit der Begriff Anwendung einer SPA gleichgesetzt.

3.3 Design Patterns

Dieser Abschnitt geht auf Design Patterns ein, die von den ausgewählten JavaScript Libraries und Frameworks verwendet werden. Design Patterns sind wiederverwendbare Lösungen für Probleme, die oft im Software-Design auftreten. Dabei sind Design Patterns jedoch keine exakte Lösung, sondern dienen als Vorlage dafür wie Probleme gelöst werden können (Gamma et al., 1994).

In Bezug auf die ausgewählten JavaScript Libraries und Frameworks werden die folgenden Design Patterns verwendet, um die Darstellungslogik von der Geschäftslogik zu trennen. Ausgenommen dabei ist Dependency Injection, das zur Entkoppelung verwendet wird (Osmani, 2012).

3.3.1 Model-View-Controller

Das Design Pattern Model-View-Controller (MVC) hat das Ziel die Struktur einer Anwendung zu verbessern. MVC erzwingt die Isolierung der Geschäftsdaten, den sogenannten Models, von User Interfaces, die als Views bezeichnet werden, mittels einer dritten Komponente, den Controller. Dieser verwaltet die Logik, verarbeitet Benutzereingaben und koordiniert die Models und Views (Gamma et al., 1994).

Die folgenden Definitionen der MVC Komponenten beziehen sich auf Osmani (2012).

Model

Models repräsentieren die Daten einer Anwendung. Sie befassen sich nicht mit der UI oder mit dem User Input. Wenn sich das Model ändert, gibt es einen Mechanismus, der die beobachtenden Views benachrichtigt, so dass sich diese aktualisieren können.

View

Eine View ist eine visuelle Darstellung von Models, die deren aktuellen Zustand abbilden. Wie zuvor beschrieben beobachtet die View das Model und wird benachrichtigt, sobald sich das Model ändert. Somit kann sich die View entsprechend aktualisieren. Die View ist auch der Interaktionspunkt zu den Benutzerinnen und Benutzer, die beispielsweise Daten eines Models in der View bearbeiten können. Werden die Daten bearbeitet, so ist es die Aufgabe des Controllers das Model entsprechend zu aktualisieren. Die View von MVC wird als passiv bezeichnet, da diese keine Informationen des Controllers hat und nur durch diesen manipuliert wird.

Controller

Ein Controller ist ein Vermittler zwischen den Models und den Views und ist für zwei Aufgaben verantwortlich. Erstens für die Aktualisierung der View, wenn sich das Model ändert, sowie zweitens für die Aktualisierung des Models, wenn die Benutzerin oder der Benutzer die View bearbeitet.

3.3.2 Model-View-ViewModel

Das von Microsoft definierte Design Pattern Model-View-ViewModel (MVVM) basiert auf den Konzepten von MVC und dem sogenannten Model-View-Presenter (MVP) Pattern. Bei MVP handelt sich um eine Ableitung von MVC, welche sich auf die Verbesserung der Darstellungslogik fokussiert. MVVM versucht die Entwicklung der UI klarer von der Geschäftslogik und des Verhaltens einer Anwendung zu trennen (Kouraklis, 2016).

Die Beschreibung der drei MVVM Komponenten bezieht sich auf Osmani (2012).

Model

Wie im vorhergehenden Abschnitt beschrieben, repräsentiert auch bei MVVM das Model die Daten der Anwendung.

View

Analog zu MVC ist auch bei MVVM die View der Interaktionspunkt zu den Benutzerinnen und Benutzer. Anders als bei MVC wird jedoch nicht das Model dargestellt, sondern der aktuelle Status des ViewModels. Ebenso anders als bei MVC, handelt es sich bei der View von MVVM um eine aktive View. Das liegt daran, dass die View Data Bindings enthält und für das Event Handling verantwortlich ist.

ViewModel

Das ViewModel kann als spezialisierter Controller betrachtet werden, der für das Konvertieren von Daten verantwortlich ist. Er ändert die Informationen des Models in der View und übergibt Befehle von der View zum Model. Beispielsweise kann das Model ein Datum in Form eines UNIX Zeitstempels beinhalten. In der View selbst ist die formatierte Datumsansicht festgelegt. Das ViewModel dient als Vermittler zwischen den beiden Komponenten das die Logik verwaltet.

Die nachfolgende Abbildung stellt den Unterschied zwischen MVC und MVVM vereinfacht dar. Bei MVC stellt die View rein die Informationen des Models dar, welche vom Controller bearbeitet werden. Bei MVVM hat die View jedoch durch das Data Binding direkten Zugriff auf das ViewModel, welche die Daten aus dem Model konvertiert (Syromiatnikov & Weyns, 2014).

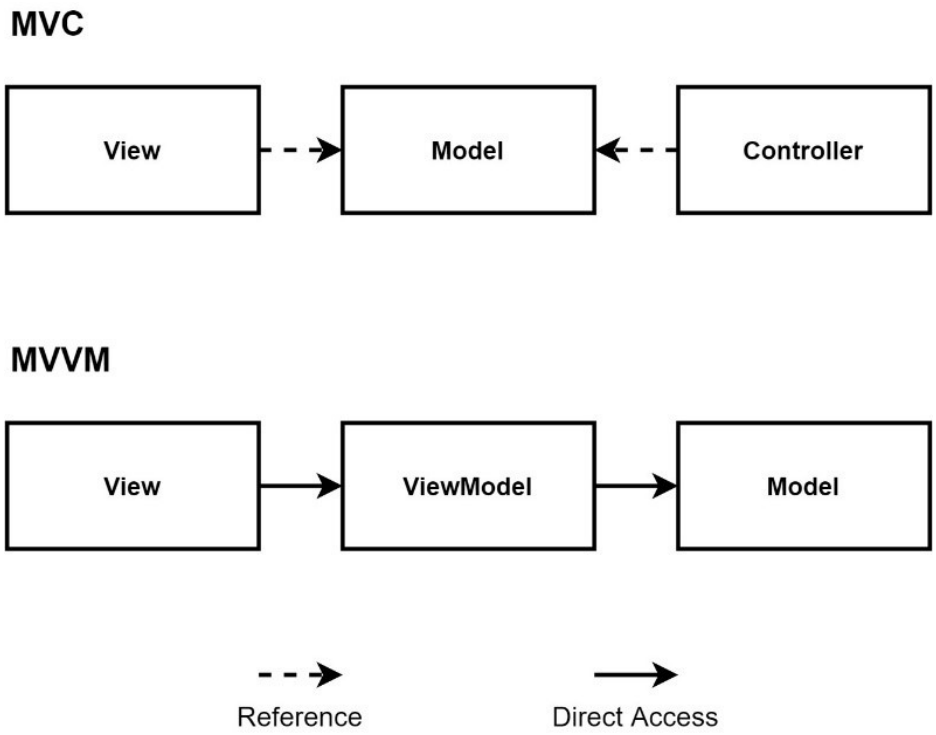

Abbildung 5: Unterschied zwischen MVC und MVVM in Anlehnung an Syromiatnikov & Weyns, 2014

3.3.3 Dependency Injection

Ein weiteres Design Pattern, dass in Angular/Angular.js verwendet wird, ist Dependency Injection (DI). An dieser Stelle wird angemerkt, dass Dependency Injection in der Literatur als Software-Design Prinzip, als auch als Design Pattern definiert wird. Das Ziel dieses Patterns besteht darin, dass Objekte und deren Abhängigkeiten entkoppelt werden. Um dies zu erreichen, werden die Abhängigkeiten nicht im Objekt selbst definiert, sondern stattdessen dem Objekt übergeben (van Deursen & Seemann, 2019).

Anwendungsbeispiele von Dependency Injection finden sich beim Testen von Software. Wenn Tests externe Komponenten benötigen, wie beispielsweise den Zugriff auf ein Netzwerk, kann bei einer Implementierung mittels Dependency Injection ein Objekt erstellt werden, dass den Netzwerkzugriff simuliert. Dieses wird beim Ausführen des Tests per DI übergeben. Aber auch in Frameworks findet Dependency Injection Anwendung, beispielsweise bei dem Aufruf und dem Übergeben von Services (van Deursen & Seemann, 2019).

4 JAVASCRIPT FRAMEWORKS UND LIBRARIES

Dieses Kapitel geht auf die relevanten Details von JavaScript, sowie auf die ausgewählten JavaScript Frameworks und Libraries der vorliegenden Arbeit ein.

4.1 JavaScript

JavaScript ist eine Skriptsprache, die vom Browser ausgeführt wird. Sie wurde ursprünglich im Jahre 1995 von Brendan Eich bei Netscape Communications entwickelt. Im Jahre 1996 wurde JavaScript bei Ecma eingereicht - einer privaten, internationalen Organisation für gemeinnützige Standards in der Technologie- und Kommunikationsbranche. Seither ist JavaScript eine Implementierung der Ecma Spezifikation ECMAScript (ES), die Features und Sprache kontinuierlich verbessert und erweitert (Brown, 2016).

4.1.1 JavaScript Object Notation

Wie bereits in Abschnitt 3.1.4 Kommunikation der Komponenten angemerkt, handelt es sich bei JSON um ein Dateiformat. Dieses wurde im Jahr 1999 in einer Ecma Spezifikation eingeführt. Mit JSON werden Objekte in JavaScript definiert, die entsprechend die Daten und Strukturen repräsentieren. Das Dateiformat ist so konzipiert, dass es einfach lesbar ist und wenig Speicher benötigt. Gespeichert wird das Format mit der Dateiendung .json (Jackson, 2016).

Das folgende Listing zeigt die Darstellung eines Objekts im JSON Format. Dabei wird ein Objekt mit ID, Name, Alter sowie zwei Hobbies dargestellt. Die ID und das Alter sind Integer-Werte und somit ohne Anführungszeichen geschrieben. Der Name wird als String angegeben und muss somit mit Anführungszeichen definiert werden. Die Hobbies sind ein Array aus Strings. Das ganze Objekt wird in den geschweiften Klammern zusammengefasst (Marrs, 2017).

```
{
   "ID": 1,
   "Name": "John",
   "Age": 35,
   "Hobbies": [
      "Baseball",
      "Cooking"
   ]
}
```

Listing 2: Darstellung eines Objects in JSON in Anlehnung an Marrs, 2017

4.1.2 Document Object Model

Das Document Object Model (DOM) beschreibt die Struktur eines HTML Dokuments in Form einer Baumstruktur. Jeder Knoten des Baumes ist dabei ein Objekt, das einen Teil des Dokuments repräsentiert und entweder keine oder mehrere untergeordnete Knoten hat (Marini, 2002).

Das DOM ist das Kernstück der Interaktion mit einem Browser. Durch das DOM sind nach Elliot (2014) folgende Interaktionen möglich:

- Auffinden von gesuchten Elementen
- Manipulieren bestehender Elemente
- Erstellen neuer DOM Objekte
- Veränderung des Style eines Elements
- Bearbeiten der Daten-Attribute
- Hinzufügen, entfernen von Events, sowie das Reagieren und Zuhören auf diese
- Asynchrone Aufrufe zur Kommunikation mit einem Server

Alle aufgezählten Interaktionsmöglichkeiten werden von JavaScript mittels Funktionen eines Application Programmer Interface (API) den Entwicklern zu Verfügung gestellt (Keith & Sambells, 2011).

Abbildung 6 visualisiert das Document Object Model. Dabei stellt das Dokument selbst die Wurzel der Baumstruktur dar. Darunter das erste `<html>` Element, dass in dieser Darstellung die Elemente `<head>` und `<body>` als untergeordnete Knoten hat. Die beiden Elemente selbst haben wiederum entsprechende untergeordnete Knoten. Im `<head>` werden dabei allgemeine Metadaten wie beispielsweise der Titel der Seite angegeben. Im `<body>` werden die Elemente definiert, welche die Darstellung der Website oder Webanwendung für die Ansicht der Benutzerinnen und Benutzer beinhalten (Brown, 2016).

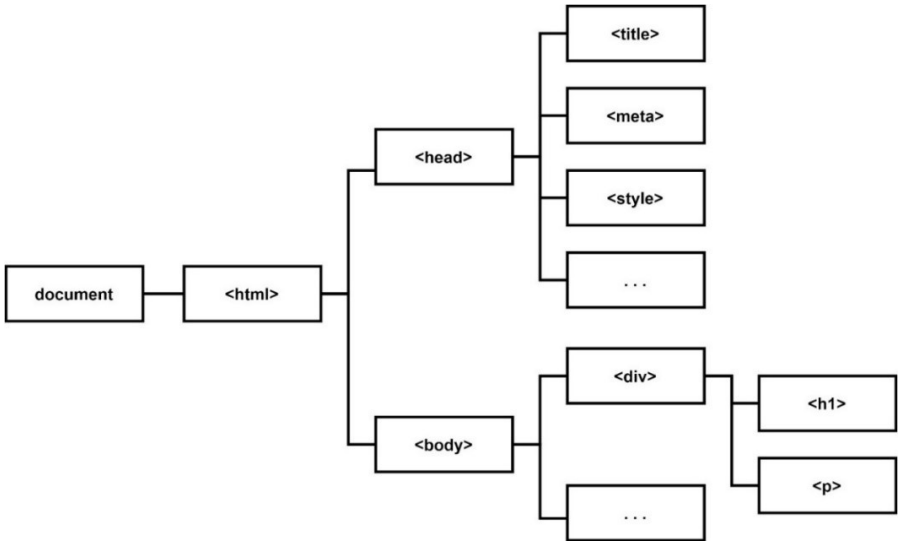

Abbildung 6: Grafische Darstellung eines Document Object Model in Anlehnung an Brown, 2016

In Abbildung 7 wird das DOM einer realen Anwendung dargestellt, das mit den Entwicklertools eines Browsers aufgezeichnet wurde. Es wird zuvor mittels `<!DOCTYPE html>` der HTML-Typ definiert und alle nachfolgenden Elemente werden in einem `<html>` Element zusammengefasst. Wie zuvor in Abbildung 6 zusehen, teilen sich die Inhalte in den `<head>` mit allgemeinen Metadaten und den `<body>`, in dem die Elemente der Website definiert sind. In dem angeführten Beispiel sind im `<body>` mehrere verschachtelte `<div>` Elemente, welche die Navigation und eine Formularfeld beinhalten.

```
<html lang="en">
▼<head>
    <meta charset="utf-8">
    <link rel="icon" href="/favicon.ico">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="theme-color" content="#000000">
    <link rel="apple-touch-icon" href="/logo192.png">
    <title>Base-React</title>
  ▶<style type="text/css">…</style>
  </head>
▼<body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
  ▼<div id="root">
    ▶<div id="nav">…</div>
    ▼<div id="content">
      ▼<div id="Input">
        ▼<form>
            <input type="text" placeholder="Input" value="Test">
          ▶<p>…</p>
            <button type="submit">Save</button>
          </form>
        </div>
      </div>
    </div>
    <script src="/static/js/bundle.js"></script>
    <script src="/static/js/0.chunk.js"></script>
    <script src="/static/js/main.chunk.js"></script>
  </body>
</html>
```

Abbildung 7: Aufzeichnung eines DOM einer Anwendung in Anlehnung an Duckett, 2010

4.1.3 Virtual DOM

Das Virtual DOM ist eine Kopie des DOM. Der Grundgedanke bei diesem Konzept ist, dass die virtuelle Repräsentation in-memory gehalten wird und mit dem tatsächlichen DOM synchronisiert wird. Dieses Konzept wird von JavaScript Libraries über die Browser-APIs implementiert (React, o. D.).

Durch die Verwendung des Virtual DOM wird das DOM nur aktualisiert, sobald eine Änderung im Virtual DOM geschieht. Das hat den Vorteil, dass es nicht notwendig ist die komplette Seite neu

zu rendern. Zusätzlich ist das Manipulieren des Virtual DOM schneller als beim DOM (Uzayr et al., 2019).

4.1.4 Style

Grundsätzlich wird das Aussehen von Websites über CSS-Dateien definiert. Mit JavaScript ist es zusätzlich möglich den Style von HTML-Elementen dynamisch über das DOM anzupassen. Jedes Element im DOM kann ein Style-Objekt besitzen, in dem die Eigenschaften mit den entsprechenden Werten festgehalten sind. Mittels JavaScript ist es möglich, dieses Style-Objekt zu manipulieren oder zu erstellen, falls es noch nicht vorhanden ist. Dafür kann entweder ein bestehender Wert einer CSS-Eigenschaft geändert werden oder eine neue Eigenschaft mit dem entsprechenden Wert hinzugefügt werden (Duckett, 2010).

Im Laufe der letzten Jahre wurden weitere Möglichkeiten entwickelt, um den Style einer Webanwendung zu definieren. Dabei handelt es sich um CSS-Präprozessoren, wie beispielsweise SCSS, LESS oder SASS, die es unter anderem ermöglichen Logiken und Variablen zu definieren. Alle davon bauen jedoch auf CSS auf, um die Entwicklung zu vereinfachen (Mazinanian & Tsantalis, 2016). Da sich die vorliegende Arbeit jedoch ausschließlich mit der Definition des Styles mittels CSS beschäftigt, wird an dieser Stelle keine tiefergehende Erklärung angeführt.

4.1.5 Frameworks und Libraries

Konzepte wie beispielsweise das Virtual DOM, die grundsätzlich für viele Webanwendungen wiederverwendbar und nützlich sind, haben unter anderem dazu geführt, dass Libraries und Frameworks entstanden sind. Bei einer Library handelt es sich um einen wiederverwendbaren Code-Abschnitt, der sich auf eine einzige Funktionalität beschränkt. Die Verwendung des Codes folgt durch ein API, das definierte Library-Funktionen zur Verfügung stellt. Ein Framework besteht auch aus Code-Abschnitten, gibt jedoch die gesamte Architektur einer Anwendung vor. Somit müssen bei der Entwicklung einer Anwendung mit einem Framework die vorgegebenen Entwurfsmethoden befolgt werden. Frameworks inkludieren im Normalfall Libraries, die für bestimmte Funktionen genutzt werden (Wilken, 2018).

4.2 Angular.js

Angular.js ist ein MVC Framework, das in und für JavaScript entwickelt wurde (Angular, o. D.). An dieser Stelle wird angemerkt, dass es zwei Versionen von Angular gibt, weshalb zuvor die Schreibweise Angular/Angular.js verwendet wurde. Während Angular.js auf JavaScript basiert, wurde Angular in TypeScript entwickelt, einer objektorientierten Erweiterung von JavaScript (Wilken, 2018).

Der Unterschied zwischen JavaScript und TypeScript hat für diese Arbeit keine Relevanz. Jedoch wird klargestellt, dass die vorliegende Arbeit nur Frameworks und Libraries auf Basis von

JavaScript behandelt und deshalb Angular.js und nicht Angular für die Portierungen verwendet. Zusätzlich wird angemerkt, dass ab dieser Stelle nur mehr der Begriff Angular.js anstelle von Angular/Angular.js verwendet wird.

Die folgenden Abschnitte geben einen Überblick über die grundlegenden Konzepte und Bestandteile von Angular.js.

4.2.1 Modules

Angular.js verfolgt einen modularen Aufbau mit sogenannten Modules. Ein Module kann dabei unterschiedliche Teile der Webanwendung beinhalten, wie beispielsweise einen Controller. Hintergrund für diesen Aufbau ist, dass Angular.js keine Main-Methode hat, die zu Beginn aufgerufen wird. Anstelle dessen wird durch die Modules angegeben wie die Anwendung gebootet werden soll (Angular.js, o. D.).

Das führt nach Angular.js (o. D.) zu den folgenden Vorteilen:

- Der Code kann in wiederverwendbare Modules verpackt werden.
- Die Modules können in beliebiger Reihenfolge oder parallel geladen werden.
- Die Unit-Tests müssen nur entsprechend relevante Modules laden, wodurch diese schneller werden.
- Tests können Modules verwenden, um beispielsweise Konfigurationen zu überschreiben.

4.2.2 Controller

Webanwendungen, die mit dem Framework Angular.js entwickelt sind, werden von einem Controller gesteuert. Der Controller selbst ist ein JavaScript Objekt und beinhalten die Geschäftslogik einer einzelnen View. Der Controller wird stets von Angular.js mit dem Scope Objekt aufgerufen, das im nachfolgenden Abschnitt beschrieben ist. Der Controller soll stets nur das Scope Objekt bearbeiten. Er soll nicht dazu verwendet werden, um das DOM zu manipulieren, Input zu formatieren, Output zu filtern oder Code und Daten mit anderen Controller zu teilen (Chandermani, 2015).

4.2.3 Scope

Das Scope Objekt ist ein JavaScript Objekt, das für die Kommunikation zwischen Controller und View genutzt wird. Das Scope Objekt bietet API-Funktionen zum Überwachen des Models und zum Verändern des Models durch den Controller. Dabei wird jedoch für jeden Controller ein eigenes Scope Objekt übergeben. Somit können Daten und Methoden des Scope Objekts eines Controllers nicht von anderen Controller aufgerufen werden (Gharat & Nehlsen Matthias, 2014).

Listing 3 stellt den Zusammenhang zwischen Scope Objekt und Controller dar. Im ersten Schritt wird in JavaScript die Variable `myApp` definiert, die ein Angular.js Module für dieses Beispiel ist. Im nächsten Schritt wird `GreetingController` erstellt und in dessen Scope Objekt die Variable

`greeting` mit `Hallo!` gesetzt. Diese Variable kann nun in einem Template verwendet werden (Angular.js, o. D.). Auf Templates wird in den nächsten Abschnitten eingegangen.

```
var myApp = angular.module('myApp',[]);

myApp.controller('GreetingController', ['$scope', function($scope) {
  $scope.greeting = 'Hallo!';
}]);
```

Listing 3: JavaScript Code eines Controllers in Angular.js in Anlehnung an Angular.js, o. D.

4.2.4 Services

Damit Code anwendungsübergreifend geteilt werden kann, bietet Angular.js sogenannte Services. Die Services selbst sind Singletons und somit nur einmalig instanziiert. Eine weitere Eigenschaft ist, dass die Services nur dann geladen werden, wenn diese von einem Teil der Anwendung benötigt werden. Ein beispielhaftes Service ist der `$http` Service, den Angular.js out-of-the-box bietet. Dieser Service ermöglicht die Kommunikation mit einem Backend und hat bereits alle notwendigen HTTP-Funktionen dafür implementiert (Panda, 2014).

4.2.5 Templates

Angular.js verfolgt den Ansatz von Templates. Das Template wird in HTML geschrieben und beinhaltet die entsprechenden HTML-Elemente mit den spezifischen Angular.js-Elementen und -Attributen. Angular.js bringt die Templates mit den Informationen des Models und des Controllers in Verbindung, um die dynamische View für den Benutzer im Browser zu rendern (Williamson, 2015).

Listing 4 zeigt ein Template, dass den zuvor definierten `GreetingController` verwendet. Das Template selbst besteht aus den einzelnen HTML-Elementen. Im `<div>` wird mittels einem sogenannten Directive der Controller für dieses Element gesetzt. Darunter wird im `<script>` eine JavaScript-Datei geladen, die für Angular.js notwendig ist (Angular.js, o. D.).

```
<html ng-app>
 <body>
    <div ng-controller="GreetingController">
          {{ greeting }}
    </div>
   <script src="angular.js"></script>
 </body>
</html>
```

Listing 4: Beispielhaftes Template in Angular.js in Anlehnung an Angular.js, o. D.

4.2.6 Directives

Die sogenannten Directives sind Markierungen eines Elements in DOM, zum Beispiel ein Attribut oder eine CSS-Klasse eines HTML-Elements. Mit diesen Markierungen kann Angular.js den DOM Elementen ein bestimmtes Verhalten zuweisen, wie beispielsweise einen Event Listener. Das Framework bietet eine Vielzahl an vorgefertigten Directives wie beispielsweise ngClass, zum dynamischen Setzen von CSS-Klassen oder ngController zum Setzen eines Controllers. Es bietet jedoch auch die Möglichkeit eigene Directives zu erstellen (Tarasiewicz & Böhm, 2014).

4.2.7 Expressions

Expressions sind Codeteile, die für das Data Binding eines HTML-Elements verwendet werden können, aber auch direkt in Directives eingebunden werden können. So kann beispielsweise die CSS-Klasse eines HTML-Elements per Expression übergeben werden und somit dynamisch verändert werden. In Listing 4 ist in Zeile vier eine Expression zu sehen. Die Variable `greeting` wird im Controller in dem Scope Objekt abgelegt und mittels Expression im Template dargestellt (Dayley, 2014).

4.2.8 Components

Angular.js bietet die Möglichkeit Templates und Controller in einer Component zusammenzufassen. Durch diesen Aufbau entsteht der Vorteil der Wiederverwendbarkeit, da die Components isoliert als Einheit betrachtet werden können. Technisch gesehen sind Components in Angular.js eine Art der Directives, die weniger Konfiguration benötigen (Angular.js, o. D.).

In Listing 5 ist die Definition einer Component in Angular.js dargestellt. Dabei wird zuvor das Module `testApp` erstellt. Mittelhilfe der `component()` Methode wird zuvor der Name der Component übergeben, in diesem Fall wird diese `testDetail` benannt. Danach werden die Eigenschaften als JavaScript Objekt definiert. In dem angeführten Listing wird ein Template per `templateUrl` übergeben, was in Angular.js aus Gründen der Übersicht empfohlen wird (Angular.js, o. D.).

```
angular.module('testApp')
    .component('testDetail', {
          templateUrl: 'testDetail.html',
          ...
});
```

Listing 5: Definition einer Component in Angular.js in Anlehnung an Angular.js, o. D.

4.2.9 Filters

Die Filter von Angular.js werden zur Formatierung einer Expression verwendet. Beispielsweise wird in einem HTML-Element der Preis eines Produkts dargestellt. Per Expression wird nur der unformatierte Wert des Preises übergeben. Zusätzlich dazu wird ein Filter angegeben, der die Zahl dann entsprechend in die beispielhafte Form von €12.9 umwandelt (Green & Seshadri, 2013).

Listing 6 zeigt zwei weitere Eigenschaften. Filter können verkettet werden, so dass der nachfolgende Filter als Input den Output des vorherigen verwendet. Eine weitere Möglichkeit ist, dass dem Filter entsprechende Argumente übergeben werden, beispielsweise bei einer Formatierung einer Dezimalzahl die Anzahl der Nachkommastellen (Angular.js, o. D.).

```
{{ expression | filter1 | filter2 | ... }}

{{ expression | filter:argument1:argument2:... }}
```

Listing 6: Verkettung von Filter und mehrere Argumente in Angular.js in Anlehnung an Angular.js, o. D.

4.2.10 Dependency Injection

Um Services einzubinden verwendet Angular.js das in Abschnitt 3.3.3 Dependency Injection angeführte Design Pattern. Die Implementierung von Dependency Injection in Angular.js hat dabei ein Injector Subsystem, dass für das Erstellen der Components zuständig ist. Dieses System übergibt den Components die angeforderten Abhängigkeiten. Bei Angular.js können neben den Services noch Directives, Filter und Animationen per Dependency Injection übergeben werden (Williamson, 2015).

Listing 7 stellt die Verwendungsweise von Dependency Injection in einem Controller dar. Dabei wird beim Erstellen des Controllers `MyController` das zuvor beschriebene Scope Objekt und der `$http` Service von Angular.js via Dependency Injection geladen und können somit im Controller verwendet werden (Angular.js, o. D.).

```
myModule.controller('MyController', ['$scope', '$http', function($scope,
$http) {
  ...
}]);
```

Listing 7: Dependency Injection in einem Controller in Angular.js in Anlehnung an Angular.js, o. D.

4.2.11 Data Binding

Angular.js verwendet einen Two-Way Data Binding Ansatz. Das bedeutet, dass im ersten Schritt ein Template mit den HTML- und Angular.js-Elementen und zusätzlichen Directives und Expressions erstellt wird, welches live vom Browser zu einer View kompiliert wird. Jegliche Änderung führt zu einer Änderung im Model, was wiederum zu einer sofortigen Änderung der View führt. Die nachfolgende Abbildung stellt diesen Ablauf grafisch dar. Das Template wird vom Browser zur View kompiliert. Sollten durch Interaktion der Benutzerinnen oder Benutzer Änderungen geschehen, so wird das Model verändert, wodurch eine Aktualisierung der View erfolgt. Aber auch Änderungen im Model führen zur Aktualisierung der View (Angular.js, o. D.).

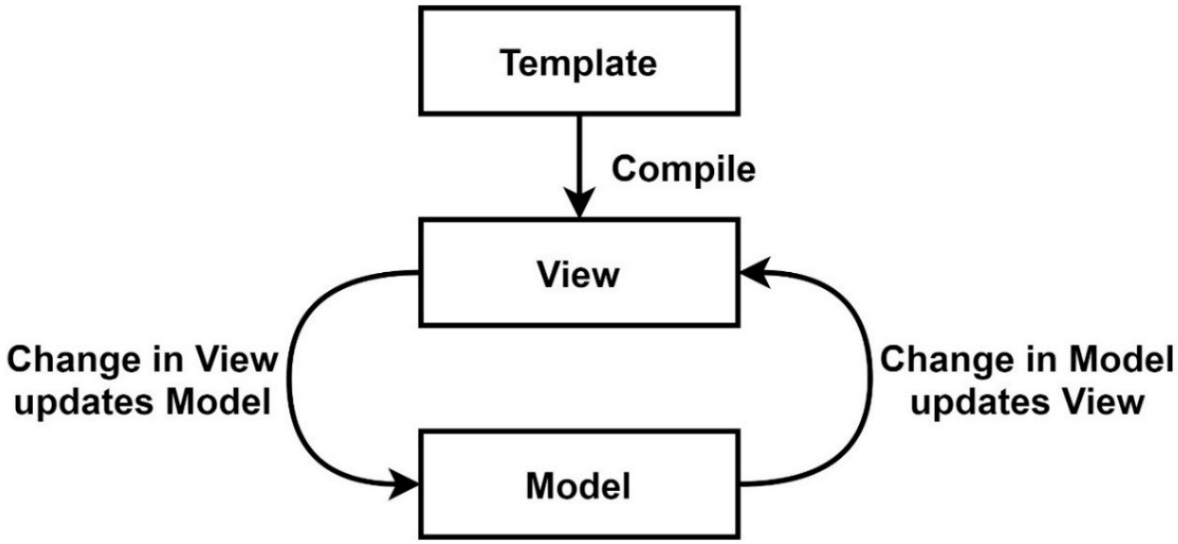

Abbildung 8: Two-Way Data Binding in Angular.js in Anlehnung an Angular.js, o. D.

4.2.12 Routing

Wie in Abschnitt 4.2.1 Modules beschrieben, sind Angular.js Webanwendungen modular aufgebaut. Diese Vorgehensweise sorgt dafür, dass die Views, Controller und Models entsprechend aufgeteilt sind. Die Zuweisung, wann welches Template und welcher Controller verwendet werden soll, erfolgt über sogenanntes Routing. Das Routing ermöglicht es einen Pfad anzugeben und diesen mit den entsprechenden Template und Controller zu verbinden (Williamson, 2015).

Listing 8 zeigt das Routing in Angular.js in Form einer beispielhaften Anwendung, die eine Liste von Smartphones darstellt. Im ersten Schritt wird das Module mit ngRoute definiert, damit das Routing verwendet werden kann. Danach wird der `$routeProvider` konfiguriert. In diesem Fall wird bei dem Pfad `/phones` das Template `<phone-list>` gerendert. Wird zusätzlich eine ID im Pfad angegeben wird das `<phone-detail/>` Template verwendet. Falls ein anderer Pfad aufgerufen wird, der nicht definiert wurde, wird mittels `otherwise()` Methode wieder auf den `/phone` Pfad weitergeleitet (Angular.js, o. D.).

```
angular.module('phoneApp', [
  'ngRoute'
]);

angular.
  module('phoneApp').
  config(['$routeProvider',
    function config($routeProvider) {
      $routeProvider.
        when('/phones', {
          template: '<phone-list /> '
        }).
        when('/phones/:phoneId', {
          template: '<phone-detail />'
        }).
        otherwise('/phones');
    }
  ]);
```

Listing 8: Beispiel von Routing in Angular.js in Anlehnung an Angular.js, o. D.

4.2.13 Style

Angular.js bietet die Möglichkeit, dass der Style eines HTML-Elements über ein Directive definiert wird. Hierfür wird mittels ngStyle die entsprechende Eigenschaft und der Wert angegeben, wie im nachfolgenden Listing dargestellt. In dem Beispiel wird zuvor ein Button erstellt, welcher bei Klick auf diesen die Variable `myStyle` mit der Eigenschaft `color` und dem Wert `blue` setzt. Dem nachfolgenden Text werden mittels Directive diese Variable als Style übergeben (Angular.js, o. D.).

```
<input type="button"
       value="Set Color"
       ng-click="myStyle={color:'blue'}">
<span ng-style="myStyle">Text</span>
```

Listing 9: Definition von Style in Angular.js in Anlehnung an Angular.js, o. D.

Zusätzlich ist es in Angular.js möglich eine allgemeine CSS-Datei zu importieren, die den Style einer gesamten Anwendung definiert. Dafür kann entweder in der index.html mittels `<link>` das Stylesheet importiert werden oder die Import-Funktion von JavaScript verwendet werden (Williamson, 2015).

4.3 React.js

Im Gegensatz zu Angular.js ist React.js eine Library und kein Framework. Im MVC Pattern betrachtet, stellt React.js grundsätzlich nur die View dar, jedoch ermöglicht das Ökosystem rund um React.js die Entwicklung einer vollständigen Frontend-Anwendung. Im Detail bedeutet das, dass React.js grundsätzlich nur für das Rendering im DOM verantwortlich ist und weitere Libraries benötigt werden, um ein vollständiges Frontend zu entwickeln (Wieruch, 2018).

4.3.1 JSX

React.js verwendet eine Syntax-Erweiterung für JavaScript, das sogenannte JSX. Die Entwicklung in React.js ist auch grundsätzlich ohne Verwendung von JSX möglich, jedoch wird JSX von den Entwicklern empfohlen (React, o. D.).

JSX ermöglicht es Elemente in React.js zu erstellen, wobei die Syntax an HTML angelehnt ist. Der Typ des Elements wird mit den HTML-Tag angegeben und kann frei definiert werden. Die Attribute des Tags stellen die Eigenschaften des Elements dar. JSX erlaubt es die HTML-Syntax in JavaScript Code zu verwenden. Somit kann in einer Datei die Logik so wie auch das Template definiert werden (Fedosejev, 2015).

Listing 10 zeigt die Verwendung von JSX. Dabei wird die Variable namens `element` definiert. Diese besteht aus einem `<div>` sowie einer `<h1>` und einer `<h2>` Überschrift.

```
const element = (
  <div>
    <h1>Hello</h1>
    <h2>and good morning</h2>
  </div>
);
```

Listing 10: Definition eines Elements in React.js mittels JSX in Anlehnung an React, o. D.

Die mit JSX erstellen Elemente selbst werden entsprechend im DOM gerendert, wobei ein Element in React.js kein DOM Element ist, sondern ein reines JavaScript Objekt. React.js verwendet React DOM, um das DOM zu aktualisieren und zu synchronisieren, eine Implementierung des in Abschnitt 4.1.2 beschriebenen Konzeptes. Dabei aktualisiert React.js stets immer nur die notwendigen DOM Elemente, was zu einer besseren Performance führt (React, o. D.).

4.3.2 Components

Die sogenannten Components in React.js erlauben das Unterteilen der UI in unabhängige und wiederverwendbare Teile. Jede Component kann dabei wieder eine oder mehrere Components beinhalten und somit verschachtelt werden wie HTML-Elemente. Es wird zwischen sogenannten Functional Components und Class-based Components unterschieden. Der Unterschied dabei ist

die jeweilige Implementierung in JavaScript. Bei Functional Components wird eine JavaScript Funktion verwendet und Class-based Components werden als JavaScript Klasse implementiert. Weitere Unterschiede sind, dass Functional Components grundsätzlich keine Logik beinhalten und über die `return()` Methode die HTML-Elemente übergeben, die zu rendern sind. Class-based Components sind stateful, was bedeutet, dass die Component eigene Daten beinhalten kann. Zusätzlich haben diese eine eigene `render()` Methode, der die HTML-Elemente übergeben werden, die zu rendern sind (React, o. D.).

Listing 11 zeigt in einem Beispiel die zwei Components `Welcome` und `App`. Beide sind als Class-based Components implementiert. Bei der Component `Welcome` wird eine Überschrift mit dem Text `Hallo!` definiert. Die `App` Component ruft diesen Component auf, verschachtelt in einem `<div>` als überstehendes Element (React, o. D.).

```
class Welcome extends React.Component {
  render() {
    return <h1>Hallo!</h1>;
  }
}

class App extends React.Component {
  render() {
    return (
          <div>
                <Welcome />
          </div>
    );
  }
}
```

Listing 11: Verschachtelung von React.js Components in Anlehnung an React, o. D.

4.3.3 Data Binding

Um den Components Daten übergeben zu können, verwendet React.js sogenannte Props. Diese können Variablen oder Objekte sein und haben die Eigenschaft, dass sie Read-Only sind. Somit kann eine Component nie die eigenen Props verändern. Die Props werden immer von der oberen Komponente an die untere übergeben (React, o. D.).

Jede Class-based Component hat zusätzlich einen eigenen State, der die internen Variablen und Objekte beinhaltet. Der State kann im Gegensatz zu den Props nur von der Component selbst verändert werden und nicht von anderen Elementen (A M & Sonpatki, 2016).

React.js verwendet dabei den Ansatz von One-Way Data Binding. Bei diesem Ansatz wird das Model mit dem Template verbunden und danach in der View dargestellt. Das Konzept wird in React.js verwendet, um die Modularität zu gewährleisten. Die nachfolgende Abbildung stellt das One-Way Data Binding grafisch dar (React, o. D.).

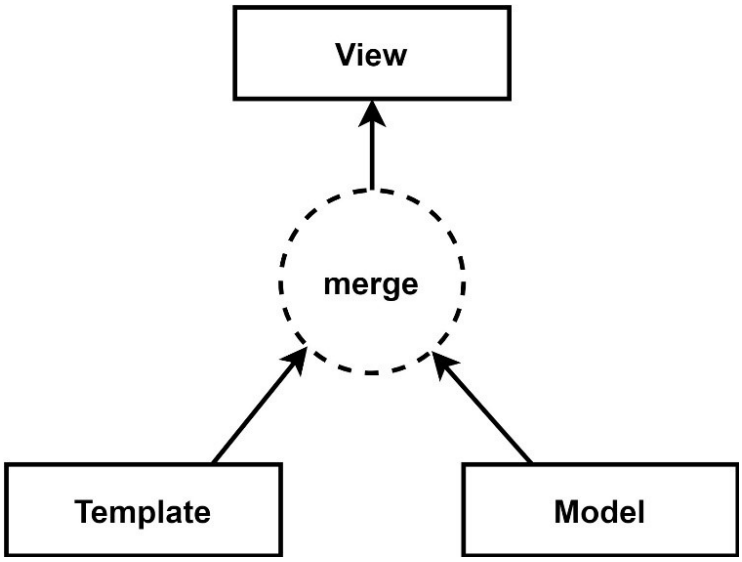

Abbildung 9: One-Way Data Binding in React.js in Anlehnung an Angular.js, o. D.

Damit jedoch der Datenfluss eines Elements auch zu überstehenden Elementen möglich ist, kann ein sogenannter Inverse Data Flow verwendet werden. Bei dieser Lösung wird eine Callback Funktion als Prop übergeben, so dass die Component diese verwenden kann, um mit dem überstehenden Element zu kommunizieren (Banks & Porcello, 2017). Ein Code-Beispiel für einen Inverse Date Flow wird an dieser Stelle nicht gegeben, da dies den Rahmen der vorliegenden Arbeit überschreiten würde.

4.3.4 Component Lifecycle

Der Component Lifecycle besteht aus Methoden, die aufgerufen werden, wenn eine Component bereitgestellt oder aktualisiert wird. Die Methoden selbst werden entweder vor oder nach dem Rendern ausgeführt. React.js unterscheidet grundsätzlich zwischen dem Mounting Lifecycle und dem Updating Lifecycle (Sidelnikov, 2017).

Der Mounting Lifecycle stellt Methoden zur Verfügung, die bei der Bereitstellung oder des Entfernens im DOM aufgerufen werden. Diese können beispielsweise dafür verwendet werden, um Daten von einem API eines Servers zu laden, die für das Darstellen der Components notwendig sind (Robbestad, 2016).

Die Methoden des Updating Lifecycle werden ausgeführt, sobald sich der State einer Component ändert. Dies kann beispielsweise dafür verwendet werden, um mit der Component nach der Aktualisierung zu interagieren damit die Darstellung geändert wird (Banks & Porcello, 2017).

4.3.5 Lifting State Up

Da wie zuvor beschrieben React.js ein One-Way Data Binding verwendet, wird empfohlen den State der Components und die Logik zum Verwalten deren in einer übergeordneten Component zu implementieren. Zur Erklärung wird an dieser Stelle als Beispiel eine Anwendung beschrieben, die Temperaturwerte in Grad Celsius und Fahrenheit darstellen kann. Zu Beginn kann eine

mögliche Implementierung so aussehen, dass eine `temperatureComponent` geschrieben wird, die eine eigene Logik zum Umschalten der Temperaturskala implementiert hat. Die Temperaturwerte selbst werden in der `temperatureComponent` jeweils in einem eigenen State in einer Variable geladen. Bei Änderung der Skala wird die Temperatur in die entsprechende andere Einheit umgerechnet. Sollte jedoch nun gewünscht sein, dass alle `temperatureComponents` synchron die Temperaturskala ändern, ist dies mit dieser Component nicht möglich, da jede für sich selbst die Skala ändert. Für solche Fälle empfiehlt React.js den State auf eine übergeordnete Component hochzuheben. In diesem Anwendungsfall könnte dafür eine `calculatorComponent` erstellt werden. In dieser wird im State definiert, dass Celsisus die Standard-Skala ist. Die `temperatureComponent` wird erweitert, dass die Skala per Prop übergeben werden kann. Die Logik zum Umschalten wird nun in die `calculatorComponent` gehoben. In dieser können entsprechend viele `temperatureComponents` erstellt werden, denen die Skala per Prop übergeben wird. Wird nun die Skala in der `calculatorComponent` geändert, ändert sich diese bei allen `temperatureComponents` synchron (React, o. D.).

4.3.6 Conditional Rendering

React.js bietet die Möglichkeit das Verhalten der Components durch sogenanntes Conditional Rendering anzupassen. Diese Vorgehensweise ermöglicht es, dass nur die relevanten Teile der Anwendung tatsächlich im DOM gerendert werden (React, o. D.).

Grundsätzlich kann dieses Verhalten auch durch If-Else Statements in JavaScript abgebildet werden. React.js ermöglicht es jedoch mit JSX im Code mittels Operatoren inline Entscheidungen zu treffen, ob ein Element für die Benutzerinnen und den Benutzer angezeigt wird oder nicht (Wieruch, 2018).

Listing 12 zeigt anhand eines Beispiels das Conditional Rendering. Dabei wird eine Functional Component als JavaScript Funktion definiert. Diese Component bekommt über Props ungelesene Nachrichten als Array übergeben, die vorübergehend in der Variable `unreadMessages` gespeichert werden. In der `return()` Methode werden die HTML-Elemente der Component definiert. Dabei wird mittels dem `&&` Operator eine Entscheidung getroffen. Falls die Länge des Arrays der `unreadMessages` Variable größer null ist, so sind Nachrichten vorhanden und das definierte `<h2>` Element wird mit der entsprechenden Anzahl an ungelesenen Nachrichten dargestellt (React, o. D.).

```
function Mailbox(props) {
  const unreadMessages = props.unreadMessages;
  return (
    <div>
      <h1>Hello!</h1>
      {unreadMessages.length > 0 &&
        <h2>
          You have {unreadMessages.length} unread messages.
        </h2>
      }
    </div>
  );
}
```

Listing 12: Beispiel von Conditional Rendering in React.js in Anlehnung an React, o. D.

4.3.7 Context

Wie in Abschnitt 4.3.3 Data Binding beschrieben wird, ist React.js grundsätzlich so aufgebaut, dass Daten von oberen Components an Untere weitergegeben werden. Für manche Fälle ist dies jedoch nicht von Vorteil. Beispielsweise bei einem Objekt, dass Informationen über die angemeldete Benutzerin oder den angemeldeten Benutzer beinhaltet. Dieses müsste stets von den oberen Components an die Unteren übergeben werden damit diese darauf zugreifen können. React.js bietet hierfür eine Lösung, den sogenannten Context. Dieser ermöglicht es Daten global in der Anwendung auszutauschen, anstelle dass diese über mehrere Ebenen hinweg übergeben werden müssen (React, o. D.).

4.3.8 Style

React.js bietet mehrere Möglichkeiten zur Definition des Styles. Eine Vorgehensweise ist die Definition des Styles inline über das `style` Attribut. Hierfür kann eine Variable definiert werden, die dem Element danach übergeben wird. Eine weitere Möglichkeit ist die Vergabe von CSS-Klassen mittels dem `className` Prop. Zusätzlich ist es möglich eine CSS-Datei zu importieren, um den Style dort zu definieren (React, o. D.).

Listing 13 zeigt das dynamische Setzen von CSS-Klassen in React.js. Zuvor wird die Klasse `nav` in der Variable `className` definiert und dem `<div>` Element per `className` übergeben. Zusätzlich wird überprüft, ob das Prop `isActive` der Component gesetzt ist. Ist dies der Fall, so wird die Klasse mit `nav-active` erweitert und somit dynamisch der Style angepasst (React, o. D.).

```
render() {
  let className = 'nav';
  if (this.props.isActive) {
    className += ' nav-active';
  }
  return <div className={className}>Link</div>
}
```

Listing 13: Erstellen von dynamischen CSS-Klassen in React.js in Anlehnung an React, o. D.

4.4 Vue.js

Vue.js ist ein JavaScript Framework für die Entwicklung von User Interfaces. Vue.js hat eine Core Library, die auf das ViewModel im MVVM Pattern fokussiert ist. Das Framework ist so konzipiert, dass je nach Bedarf weitere Libraries integrierbar sind. Vue.js verwendet wie React.js ein Virtual DOM, um die Performance zu erhöhen (Vue.js, o. D.).

4.4.1 Components

Wie die anderen in dieser Arbeit verwendeten Webtechnologien bietet Vue.js die Möglichkeit isolierte Components zu erstellen. Die Vorteile von Components sind, wie bei den anderen beiden Webtechnologien einerseits die Wiederverwendbarkeit, sowie andererseits die Erhöhung der Wartbarkeit (Djirdeh et al., 2018).

Listing 14 zeigt die Erstellung einer Component in Vue.js. Dabei wird im ersten Schritt der Name der Component definiert, in diesem Fall `myComponent`. Dieser können Optionen übergeben werden. Hier wird ein Template übergeben, dass den Text `Hello!` in einem `<div>` darstellt (Chau, 2017).

```
Vue.component('myComponent', {
  template:
    ` <div>
        Hello!
      </div>`
})
```

Listing 14: Component in Vue.js in Anlehnung an Djirdeh et al., 2018

4.4.2 Templates

Ähnlich wie Angular.js verwendet auch Vue.js Templates, die auf HTML basieren. Dabei bietet das Framework die optionale Möglichkeit die Syntax-Erweiterung JSX wie bei React.js zu verwenden. Vue.js kompiliert die Templates in das Virtual DOM und synchronisiert diese mit dem DOM. Das Framework rendert bei Änderungen nur so wenig wie notwendig neu, um die Anzahl

der Manipulationen des DOM zu minimieren und somit die Performance zu erhöhen (Vue.js, o. D.).

Das Template ist wie bei den zwei anderen Webtechnologien dafür da, um eine Vorlage zu erstellen, die danach gerendert wird. Das folgende Listing zeigt ein Template, das eine Nachricht mittels Data Binding darstellt. Die Variable `message` wird dem Template zuvor übergeben. Weiters wird hier dem `<div>` Element mittels Directive `v-bind` dynamisch das HTML-Attribut `id` übergeben. Auf die Funktionsweise von Directives wird im nächsten Abschnitt eingegangen. Vue.js erlaubt zudem die Verwendung von JavaScript Expressions im Data Binding, beispielsweise eine inkrementelle Erhöhung eines Wertes oder das Formatieren von Text (Street et al., 2018).

```
<div v-bind:id="id">
    <span>Nachricht: {{ message }}</span>
</div>
```

Listing 15: Template in Vue.js in Anlehnung an Vue.js, o. D.

4.4.3 Directives

Directives in Vue.js sind spezielle Attribute im HTML Template, die mit dem Präfix `v-` angegeben werden, wie das zuvor verwendete `v-bind`. Directives sind dafür da, um auf Änderungen im DOM zu reagieren oder um Änderungen durchzuführen. So ist es möglich mittels `v-if` If-Else Statements im Template zu erstellen (Kyrakidis & Maniatis, 2016).

Ein weiteres Beispiel ist die Definition einer Funktion bei Klick auf ein Element mittels `v-on:click`. Vue.js ermöglicht auch die Verwendung dynamischer Argumente wie beispielsweise `v-on:[eventName]="method"`. Hier wird das Event, auf das reagiert werden soll, dynamisch über die Variable `eventName` übergeben (Vue.js, o. D.).

Vue.js bietet zudem eine verkürzte Schreibweise der Directives, um die Templates übersichtlicher zu halten. Listing 16 zeigt die Syntax für `v-bind`, dass mit einem Doppeltpunkt abgekürzt werden kann. `V-on` wird mit einem @ abgekürzt. In dem Listing sind jeweils ein Hyperlink für die komplette sowie die verkürzte Syntax von `v-bind` und `v-on` definiert (Djirdeh et al., 2018).

```
<!-- v-bind -->
<!-- Komplette Syntax -->
<a v-bind:href="url"> ... </a>

<!-- Verkürzte Syntax -->
<a :href="url"> ... </a>

<!-- v-on -->
<!-- Komplette Syntax -->
<a v-on:click="method"> ... </a>

<!-- Verkürzte Syntax -->
<a @click="method"> ... </a>
```

Listing 16: Verkürzte Syntax von v-bind und v-on in Vue.js in Anlehung an Vue.js, o. D.

4.4.4 Props

Jeder Vue.js Component können Props übergeben werden, die danach im Template verwendet werden können, beispielsweise für die Darstellung von Variablen im Text. Dabei bietet Vue.js die Möglichkeit die Props in der Component ohne oder mit Datentyp zu definieren (Passaglia, 2017).

Das folgende Listing zeigt dabei den Unterschied. Im ersten Absatz wird die Component `name-without-type` definiert. Die Props dieses Components werden als Array mit den einzelnen Namen definiert. Im Template werden danach die beiden Variablen `firstName` und `lastName` dargestellt. Die zweite Component `name-with-type` unterscheidet sich nicht von der Darstellung im Template. Der Unterschied liegt bei der Definition der Props. Diese werden in einem Objekt dargestellt, bei denen der Typ definiert wird. In diesem Listing sind beide Werte vom Typ String. Bei falsch übergebenen Typen loggt Vue.js automatisch in der Konsole des Browsers mit (Vue.js, o. D.).

```
<!-- Props ohne Typ -->
Vue.component('name-without-type', {
  props: ['firstName', 'lastName'],
  template: '<p>Hallo {{ firstName }} {{ lastName }}</p>'
})
<!-- Props mit Typ -->
Vue.component('name-with-type', {
  props: {
    firstName: String,
    lastName: String
  },
  template: '<p>Hallo {{ firstName }} {{ lastName }}</p>'
})
```

Listing 17: Definition von Props eines Components in Vue.js in Anlehnung an Vue.js, o. D.

4.4.5 Computed Properties

Wie in Abschnitt 4.4.2 beschrieben bietet Vue.js die Möglichkeit Logik in Templates einzubauen, wie beispielsweise das Wechseln der Reihenfolge von Werten in einem Array mittels JavaScript Expressions. Vue.js rät jedoch davon ab, zu viel Logik im Template einzubauen. Stattdessen können sogenannte Computed Properties verwendet werden, um komplexe Logiken zu implementieren (Macrae, 2018).

Die Computed Properties sind gecached, anders als normale Methoden. Das bedeutet, dass diese nicht bei jedem Aufruf neu ausgeführt werden, sondern nur wenn sich das darunterliegende Datenmodell ändert. Zusätzlich können Computed Properties keine Parameter verwenden (Imsirovic, 2018).

In Listing 18 werden die Computed Properties anhand eines Beispiels erklärt. Das Listing zeigt, wie im ersten Schritt ein ViewModel mit dem Namen `vm` definiert wird. Dies geschieht über `new Vue()`, dass eine neue Vue.js Instanz erstellt. Im nächsten Schritt werden statische Daten angelegt, in diesem Beispiel die Variable `message` mit dem Inhalt `Hallo`. Zusätzlich wird eine Computed Property angelegt, welche die Reihenfolge der Buchstaben der Variable `message` mittels JavaScript Expressions umkehrt. Die Computed Property ermöglicht es somit die Logik an einer zentralen Stelle zu definieren, anstelle der Verwendung von Logik in den Templates (Vue.js, o. D.).

```
var vm = new Vue({
  data: {
    message: 'Hallo'
  },
  computed: {
    reversedMessage: function () {
      return this.message.split('').reverse().join('')
    }
  }
})
```

Listing 18: Definition von Computed Properties in Vue.js in Anlehung an Vue.js, o. D.

4.4.6 Data Binding

Vue.js bietet zwei Möglichkeiten des Data Bindings. Mittels dem Directive `v-bind` ist ein One-Way Data Binding möglich. Es wird jedoch empfohlen ein Two-Way Data Binding mittels `v-model` zu verwenden. Das bietet den Vorteil, dass bei Änderungen, beispielsweise eines Inputs, die UI neu gerendert wird (Vue.js, o. D.).

Im folgenden Listing wird das Data Binding anhand von zwei beispielhaften Templates dargestellt. Im ersten Input wird mittels `v-bind` die Variable zugewiesen. Das führt dazu, dass sich die

Variable `input` nicht dynamisch ändert. Wird jedoch, wie im zweiten Template, das Directive `v-model` verwendet, updaten sich entsprechend die UI und die Variable (Macrae, 2018).

```
<!-- Verwendung von v-bind -->
<div id="app">
    <input type="text" v-bind:value="input">
    <p>Text: {{ input }}</p>
</div>

<!-- Verwendung von v-model -->
<div id="app">
    <input type="text" v-model="input">
    <p>Text: {{ input }}</p>
</div>
```

Listing 19: Data Binding mittels v-bind und v-model in Vue.js in Anlehnung an Macrae, 2018

4.4.7 Event Handling

Wie zuvor erwähnt kann in Vue.js mittels `v-on` auf Events reagiert werden. Die Methoden, die ausgeführt werden sobald Events auftreten, können entweder inline im Template definiert werden oder in ein `Methods` Objekt ausgelagert werden, wobei letzteres von Vue.js empfohlen wird (Macrae, 2018).

Nach Vue.js (o. D.) gibt es zudem noch die folgenden Modifier um mittels Events Methoden aufzurufen:

- `.stop`
- `.prevent`
- `.capture`
- `.self`
- `.once`
- `.passive`

Ein Beispiel dafür ist das Unterbinden, dass beim Absenden eines Formulars die Seite neugeladen wird und stattdessen eine eigens definierte Methode aufgerufen wird. In Vue.js könnte eine mögliche Implementierung wie im folgenden Listing aussehen. Hier wird mittels `v-on:submit` auf das Submit Event gehorcht. Mittels `.prevent` Modifier wird das Standardverhalten unterbunden und somit die Seite nicht neu geladen, stattdessen wird die `onSubmitForm()` Methode aufgerufen (Macrae, 2018).

```
<form v-on:submit.prevent="onSubmitForm"></form>
```

Listing 20: Event Handling mit Event Modifier in Vue.js in Anlehnung an Vue.js, o. D.

4.4.8 Routing

Für das Routing in Vue.js gibt es zwei Möglichkeiten. Für einfaches Routing, dass nur zwischen Seiten weiterleitet, ist eine Implementierung mittels Computed Properties möglich. Die zweite Möglichkeit ist die Verwendung von Libraries. Vue.js empfiehlt für Anwendungen die eigene vue-router Library zu verwenden (Vue.js, o. D.).

Listing 21 zeigt eine Implementierung mittels den zuvor angeführten Computed Properties. Zu Beginn werden drei beispielhafte Objekte angelegt die nur Templates enthalten. Zusätzlich dazu wird ein Objekt mit den Routen angelegt. Ohne weiteren Pfad in dem Uniform Resource Locator (URL) wird auf Home geroutet, bei `/help` wird entsprechend auf Help geroutet. Die Weiterleitung selbst geschieht über die Variable `window.location.pathname`. Ändert sich diese, wird mittels Computed Property auf das entsprechende Template weitergeleitet oder, sollte keine gefunden werden, auf das `NotFound` Template (Vue.js, o. D.).

```
const NotFound = { template: '<p>Seite nicht gefunden</p>' }
const Home = { template: '<p>Home</p>' }
const Help = { template: '<p>Help</p>' }

const routes = {
  '/': Home,
  '/help': Help
}

new Vue({
  el: '#app',
  data: {
    currentRoute: window.location.pathname
  },
  computed: {
    ViewComponent () {
      return routes[this.currentRoute] || NotFound
    }
  },
  render (h) { return h(this.ViewComponent) }
})
```

Listing 21: Implementierung von Routing in Vue.js in Anlehnung an Vue.js, o. D.

4.4.9 Style

Wie die beiden anderen Webtechnologien bietet auch Vue.js mehrere Möglichkeiten, um den Style zu definieren. So kann eine CSS-Datei eingebunden werden in der das Aussehen der Anwendung definiert wird. Zusätzlich können mittels `v-bind` CSS-Klassen vergeben werden. Vue.js ermöglicht hierbei die dynamische Vergabe von Klassen, wie im nachfolgenden Listing dargestellt (Djirdeh et al., 2018).

Listing 22 zeigt die Vergabe von dynamischen Klassen mithilfe einer Computed Property. In der ersten Zeile ist ein Auszug eines Templates zu sehen, bei dem mittels `v-bind` das Objekt `classObject` übergeben wird. In den darunter folgenden Zeilen ist eine Component definiert, in deren Daten die Variable `isActive` definiert ist. Zusätzlich ist das Objekt `classObject` als Computed Property angelegt. Sollte sich nun die Variable `isActive` verändern, wird auch in der Computed Property die `active` Variable geändert und somit auch die Klasse des `<div>` Elements (Vue.js, o. D.).

```
<div v-bind:class="classObject"></div>

export default {
 data: {
   isActive: true
 },
 computed: {
   classObject: function () {
     return {
       active: this.isActive
     }
   }
}
```

Listing 22: Definition von Style in Vue.js in Anlehnung an Vue.js, o. D.

4.4.10 Single File Components

Vue.js bietet die Möglichkeit die Definition des Template, des Components und des Styles in einer einzelnen Datei zusammenzufassen. Dafür wird eine Datei mit der Endung `.vue` verwendet. Die Vorteile dabei sind, dass in einer Datei alle relevanten Inhalte der Component zu finden sind und dies übersichtlicher ist. Grundsätzlich entstehen auch keine Nachteile, da immer die Möglichkeit besteht die Inhalte trotzdem auf mehrere Dateien aufzuteilen. Dies bietet sich beispielsweise bei größeren und umfangreicheren Components an (Djirdeh et al., 2018).

An dieser Stelle wird angemerkt, dass nach Vue.js (o. D.) für die Verwendung der Single File Components sogenannte Build Tools wie zum Beispiel Webpack benötigt werden. Diese sind jedoch nicht Teil dieser Arbeit, weshalb an dieser Stelle keine Erklärung von Build Tools angeführt wird.

Listing 23 zeigt den Aufbau einer Single File Component. Zu Beginn wird das Template innerhalb der `<template>` Tags definiert. In diesem Beispiel ein `<div>` Element mit zwei Paragraf-Elementen darin. Im ersten `<p>` wird der Text `hallo` und der per Props übergebene Name dargestellt, sowie die CSS-Klasse `hello` gesetzt. Im zweiten `<p>` wird die Variable `greeting` aus den Daten der Component dargestellt. Im `<script>` wird die Component selbst definiert. In diesem Fall werden die Props als Objekt definiert. Hier wird der Name mit den Typ String und einem leeren Standardwert definiert. Zusätzlich wird in den Daten die Variable `greeting` definiert,

die über eine in den `methods()` definierte Funktion geladen wird. Am Ende befindet sich im `<style>` die entsprechende CSS-Formatierung für die zuvor verwendete Klasse. In diesem Beispiel wird eine rote Schriftfarbe gesetzt (Vue.js, o. D.).

```
<template>
 <div>
   <p v-bind:class="hello"> Hallo {{ givenName }}</p>
   <p> {{ this.greeting }} </p>
 </div>
</template>

<script>
export default {
  props: {
    givenName: {
      type: String,
      default: '',
    }
  },
  data() {
    return {
      greeting: null
    }
  },
  methods: {
    fetch() {
     this.$http.get(API.GREETING).then((response) => {
        this.greeting = response.data.greeting;
      });
    }
  }
}
</script>

<style>
.hello {
  color: red;
}
</style>
```

Listing 23: Single File Component in Vue.js in Anlehnung an Vue.js, o. D.

5 METHODEN UND STRATEGIEN FÜR PORTIERUNGEN

In diesem Kapitel werden Methoden und Strategien angeführt, die Einsatz bei Portierungen im Software Engineering finden. An dieser Stelle wird angemerkt, dass in diesem Kapitel nur Methoden und Strategien aufgelistet werden, die für die vorliegende Arbeit relevant sind, eine Vollständigkeit somit wird ausgeschlossen.

5.1 Methoden

Die folgenden Methoden dienen zur Einordnung der Portierungen im praktischen Teil, sowie zur Abgrenzung der Ergebnisse. An dieser Stelle wird zusätzlich angemerkt, dass nach Wagner (2014) keine strikte Trennung der Methoden in der Realität möglich ist. Aufgrund der unterschiedlichen Möglichkeiten können stets Mischformen der Methoden in Projekten auftreten und verwendet werden.

5.1.1 Neuentwicklung

Bei einer Neuentwicklung wird ein neues System parallel zu dem bestehenden aufgebaut. Diese Methode ist mit großen Kosten verbunden da keine Wiederverwendbarkeit gegeben ist. Zusätzlich tritt bei Neuentwicklungen stets eine Übergangsphase auf, bei der vom alten auf das neue System gewechselt wird. In dieser Phase besteht hohes Risiko, dass Teile nicht richtig funktionieren und somit das System nicht im vollem Umfang genutzt werden kann (Wagner, 2014).

In dieser Arbeit wird eine Neuentwicklung als das schlechteste Ergebnis gesehen da keine Programmteile oder Code wiederverwendet werden können.

5.1.2 Wrapping

Beim Wrapping werden Teile des originalen Systems transformiert. Das bedeutet, dass Software-Komponenten wiederverwendet werden, die bereits implementiert und gut getestet sind. Bei dieser Methode werden Programmteile als Black-Box betrachtet, die mittels eines Interfaces mit dem neuen System kommunizieren. Meistens wird Wrapping jedoch nur als kurzfristige Lösung betrachtet, da auch die Teile des originalen Systems gewartet werden müssen. Somit entsteht zusätzlicher Aufwand durch das Wrapping (Masak, 2006).

An dieser Stelle wird angemerkt, dass in der vorliegenden Arbeit auch Expressions und Logiken als Software-Komponente betrachtet werden. Grund dafür ist, dass alle der verwendeten Webtechnologien die Implementierung von Expressions und Logiken in reinem JavaScript ermöglichen und dieser Code bestenfalls, als Black-Box betrachtet, übernommen werden kann.

5.1.3 Migration

Bei Migrationen liegt der Erhalt der Funktionalität im Vordergrund, während das System überarbeitet und modernisiert wird. Hier werden schrittweise Teile im neuen System entwickelt und danach wird vom alten System auf das neue gewechselt. Diese Methode ist komplexer als Wrapping und hat ein geringeres Risiko als eine Neuentwicklung (Demeyer et al., 2009).

5.2 Strategien

Die folgenden Strategien sind Möglichkeiten, wie der Wechsel von Systemen bei Portierungen durchgeführt werden kann.

5.2.1 Cut-and-Run

Die Cut-and-Run Strategie ist eine Übergangsstrategie, bei der das alte System zu einem bestimmten Zeitpunkt deaktiviert wird und zeitgleich das neue System im selben Schritt aktiviert wird. Diese Strategie birgt jedoch hohes Risiko, da keine Erfahrungen mit dem neuen System vorliegen und es somit zu vielen Fehlern kommen kann (Tripathy & Naik, 2015).

5.2.2 Phased Interoperability

Bei Phased Interoperability werden inkrementell einzelne Teile des alten Systems zu dem neuen System getauscht. Dabei sind beide Systeme als getrennte Einheiten zu sehen. In jedem Schritt wird dabei ein definierter Teil vom alten System durch die neue Implementierung im neuen ausgetauscht. Die Reihenfolge, in welcher die Teile ausgetauscht werden, kann von unterschiedlichen Faktoren abhängen und ist frei definierbar. Vergleicht man das Risiko der Phased Interoperability mit der Cut-and-Run Strategie, so hat Phased Interoperability ein niedrigeres Risiko. Der Grund dafür liegt in der Vorgehensweise – bei einem Fehler während des Systemwechsels ist immer nur ein Teil und nicht das gesamte System betroffen (Wagner, 2014).

Für diese Arbeit wird soweit möglich, eine Phased Interoperability Strategie angestrebt. Das genaue Vorgehen dieser Arbeit wird im nächsten Kapitel angeführt.

6 PORTIERUNG DER WEBTECHNOLOGIEN

Dieses Kapitel behandelt den praktischen Teil der vorliegenden Arbeit. Es wird zuvor auf die Aufgabenstellung, die Anforderungen, sowie auf die verwendeten Methoden eingegangen. Abschließend werden die Analysen und Bewertungen der Portierungen angeführt.

6.1 Aufgabenstellung

Die Aufgabenstellung besteht aus den folgenden Teilen:

- Die Entwicklung von Basisanwendungen in Angular.js, React.js sowie Vue.js, die als Grundlage für die Portierungen dienen. Der genaue Aufbau und die Bestandteile dieser werden in den nächsten Abschnitten angeführt.
- Der zweite Teil ist die Portierung der Basisanwendungen in jeweils die anderen Webtechnologien und die Dokumentation der Erkenntnisse.
- Der darauffolgende Teil besteht aus der Erarbeitung von Richtlinien und Hilfestellungen, die für ähnliche Portierungen als Input dienen können.
- Abschließend werden die ausgewählten Technologien bewertet und die Erkenntnisse zusammengefasst, sowie tiefergehende Forschungsansätze angeführt.

6.2 Anforderungen

In diesem Abschnitt werden die genauen Anforderungen für die Aufgabenstellung definiert. Alle drei Basisanwendungen haben identische Funktionen und dieselbe Optik. Für die Endbenutzerinnen und Endbenutzer soll nicht erkenntlich sein, dass im Hintergrund eine andere Technologie verwendet wird. Um den Rahmen dieser Arbeit einzugrenzen, wird die Funktionalität der Basisanwendungen auf Formularinhalte beschränkt, die in den folgenden Abschnitten beschrieben sind.

An dieser Stelle wird zusätzlich noch eine Auflistung mit den Teilen angeführt, die nicht in dieser Arbeit behandelt werden.

- TypeScript

  Diese Arbeit beschränkt sich auf JavaScript, die Verwendung und Portierung von TypeScript ist nicht Teil dieser Arbeit.

- Validierungen

  Es wird keine Validierung der Benutzereingaben in den Formularinhalten implementiert. Auch für die Kommunikation mit dem Backend wird auf diese verzichtet und somit auch nicht portiert. Diese Arbeit beschränkt sich rein auf die Portierung der Formularinhalte.

- Build Skripte

  Für die Implementierung der Basisanwendungen werden zwar Build Skripte verwendet, jedoch behandelt diese Arbeit nicht die Portierung dieser.

- Automatisierte Tests

  Im Rahmen dieser Arbeit werden keine automatisierten Tests, wie beispielsweise Unit-Tests, implementiert.

- Security

  Diese Arbeit beschäftigt sich nicht mit der Implementierung und Portierung von Security-Maßnahmen. Auch wenn die verwendeten Webtechnologien bereits bestimmte Maßnahmen als Standard implementiert haben, wird dies nicht explizit angeführt und ist nicht Teil dieser Arbeit.

- Performance

  Im Rahmen dieser Arbeit werden keine Performance-Tests durchgeführt und verglichen.

6.2.1 Routing

Eine Navigation in Form eines Menüs gehört zu den Grundelementen einer Anwendung und ist somit relevant für die Portierungen. In jeder Basisanwendung wird ein Menü implementiert, das stets am oberen Rand des Bildschirms sichtbar ist. In der nachfolgenden Auflistung sind alle Hyperlinks und Seiten der Navigation definiert. Die Inhalte und Funktionalität dieser Seiten sind in den nachfolgenden Abschnitten angeführt.

- Home (Startseite, URL: `/`)
- Input (URL: `/input`)
- Textarea (URL: `/textarea`)
- Checkbox (URL: `/checkbox`)
- Select (URL: `/select`)
- Radio (URL: `/radio`)

6.2.2 Startseite

Die Startseite dient als reine Informationsseite. Es wird angezeigt mit welcher Webtechnologie die Basisanwendung implementiert ist. Zusätzlich wird ein Überblick über die Components der Anwendung aufgelistet. Alle Inhalte der Startseite sind statisch implementiert, die Inhalte der anderen Seiten werden jedoch dynamisch vom Backend geladen. Das Design der Startseite ist in ANHANG A - Layout der Basisanwendung in Form von Bildern festgehalten.

6.2.3 Text Input

Das Darstellen und Bearbeiten von Text wird als Grundanforderung an die Basisanwendungen gestellt. Für diese Arbeit sind die folgenden zwei Fälle implementiert.

Erstens wird auf der Seite `/input` ein einzeiliges `<input>` Element dargestellt. Dieses Element lädt einen String vom Backend und stellt diesen in dem HTML-Element dar, in dem dieser bearbeitet werden kann. Die zweite gewählte Textdarstellungsform ist ein mehrzeiliges `<textarea>` Element auf der Seite `/textarea`. Auch hier wird Text vom Backend geladen und im Element dargestellt. Beide Seiten besitzen zusätzlich einen Button zum Speichern der Daten, mehr dazu im nachfolgenden Abschnitt 6.2.7 Datenfluss.

6.2.4 Checkbox

Ein weiterer Formularinhalt, der für die Basisanwendungen gewählt wurde, ist eine Checkbox. Auf der Seite `/checkbox` wird ein einzelnes `<input type="checkbox">` Element dargestellt. Diese Checkbox wird dynamisch generiert. Das bedeutet, dass das Frontend keine statischen Informationen über den Namen und die ID hat, sowie auch keine Informationen über den `checked` Status, der angibt, ob die Checkbox ausgewählt ist oder nicht. All diese Informationen werden vorerst vom Backend geladen und danach dynamisch generiert, sobald das Frontend die Daten erhält. Zusätzlich ist auch hier ein Button zum Speichern der Daten. Es wird jedoch nur die Information über den `checked` Status an das Backend übergeben.

6.2.5 Dropdown

Auf der Seite `/select` wird ein Dropdown Menü dargestellt. Dies wird mittels `<select>` und den entsprechenden `<option>` Elementen implementiert. Wie auch bei den Checkboxen werden hier die Inhalte dynamisch geladen. Das Backend gibt dabei mehrere Auswahlmöglichkeiten vor. Das Dropdown wurde gewählt, um die unterschiedlichen For-Schleifen der Webtechnologien zu ermitteln. Sowie bei den anderen Seiten, befindet sich auch auf dieser Seite ein Button, der die aktuelle Auswahl speichert.

6.2.6 Radio Button

Das letzte Element, dass für die Basisanwendungen ausgewählt wurde sind Radio Buttons. Analog zu den Auswahlmöglichkeiten des Dropdowns werden die Radio Buttons auch dynamisch generiert und mittels For-Schleifen dargestellt. Auch diese Seite hat einen Button zum Speichern der aktuellen Auswahl.

6.2.7 Datenfluss

Um die grundlegenden Funktionen eines Webservices von realen Anwendungen nachzubilden wird der Datenfluss implementiert, der in der nachfolgenden Abbildung 10 dargestellt ist. Jedes

der Formularinhalte lädt zu Beginn die entsprechenden Daten mittels HTTP GET vom Backend. Zum Speichern werden die Daten mittels HTTP POST an das Backend gesendet. Das Backend besteht für die vorliegende Arbeit aus einem JSON Webservice, das eine lokale Datei als persistenten Speicher verwendet. Es handelt sich dabei um dieselbe Kommunikation wie in Abschnitt 3.1.4 beschrieben wird. Der einzige Unterschied ist, dass die HTTP POST Methode verwendet wird, anstelle der HTTP PUT Methode.

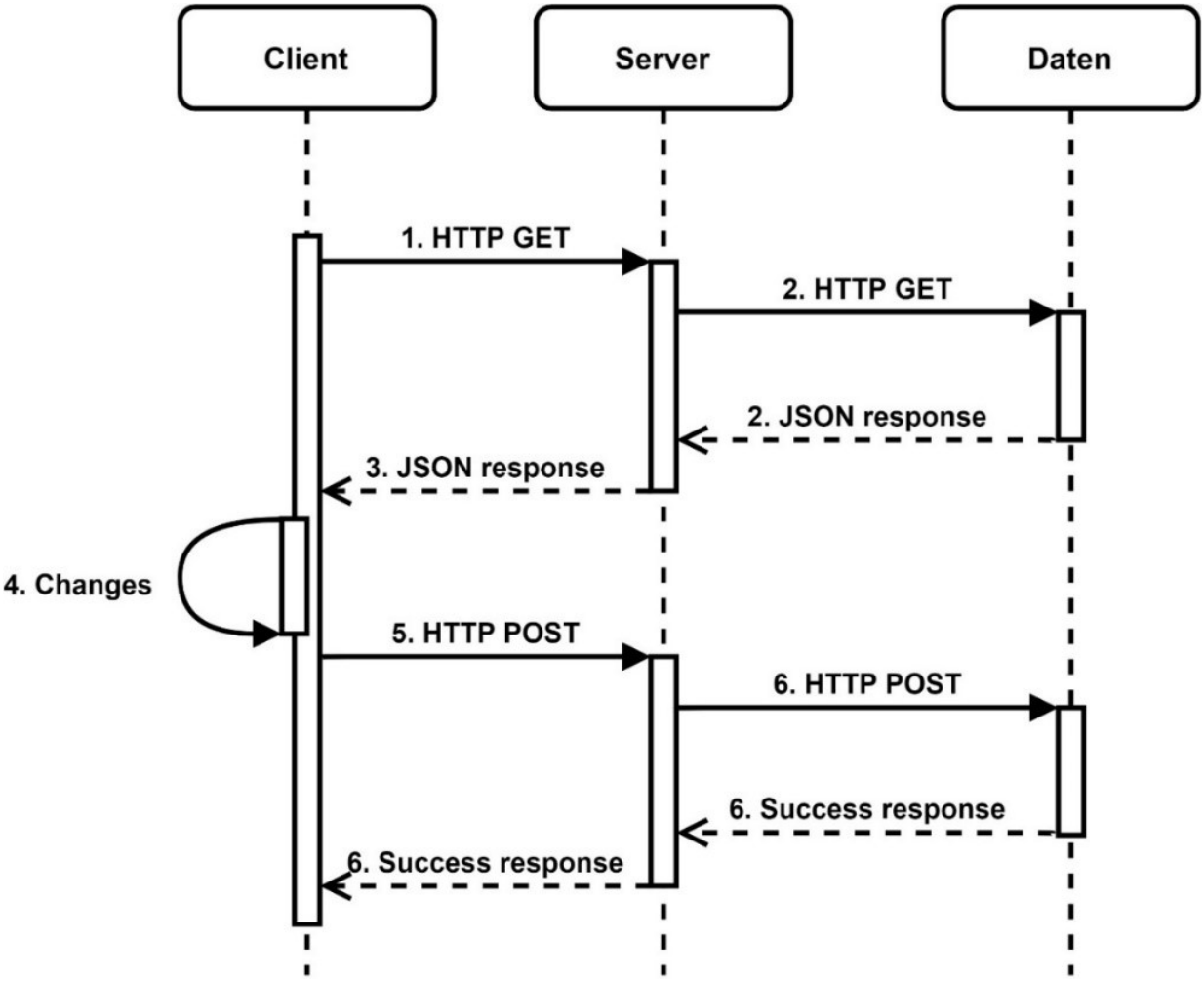

Abbildung 10: Datenfluss der Anwendungen in Anlehnung an Elliot, 2014

In den folgenden Absätzen werden die Ressourcen, sowie die entsprechenden Daten definiert, die bei den Aufrufen zurückgegeben werden. Für jede Ressource sind dabei die HTTP-Methoden GET, POST, PUT, PATCH, DELETE und OPTIONS möglich, um auf die Daten zuzugreifen und diese zu Bearbeiten. Alle Ressourcen geben die Daten im JSON Format zurück, das in Abschnitt 4.1.1 beschrieben ist.

- `/radioSelection`

  Diese Ressource gibt die Auswahlmöglichkeiten der Radio Buttons zurück. Jeder Button hat dabei eine ID sowie einen Wert für das HTML-Attribut `value`. Das folgende Listing stellt den Inhalt einer Antwort eines GET Aufrufs dar.

```
{
  "options": [
    {
      "id": 1,
      "value": "Button1"
    },
    {
      "id": 2,
      "value": "Button2"
    }
  ]
}
```

Listing 24: Antwort des Backend auf die Ressource /radioSelection

- `/radioSelected`

  Die `/radioSelected` Ressource gibt den ausgewählten `value` zurück, der entsprechend danach bearbeitet werden kann. Da sich diese Arbeit nicht auf das Backend fokussiert, wird auf das Speichern der ID verzichtet. Im folgenden Listing wird die Antwort des GET Aufrufs dargestellt.

```
{
  "value": "Button2"
}
```

Listing 25: Antwort des Backend auf die Ressource /radioSelected

- `/dropSelection`

  Analog zur `/radioSelection` Ressource gibt die `/dropSelection` die Auswahl für das Dropdown zurück. Hier werden in einem Objekt die Options mit jeweils ID und Name zurückgegeben. Das nachstehende Listing zeigt die Antwort des Backend.

```
{
  "options": [
    {
      "id": 1,
      "name": "Selection1"
    },
    {
      "id": 2,
      "name": "Selection2"
    }
  ]
}
```

Listing 26: Antwort des Backend auf die Ressource /dropSelection

- `/dropSelected`

  Diese Ressource gibt den Namen der aktuellen Auswahl im Dropdown zurück. Auch hier wird keine ID gespeichert. Das folgende Listing zeigt die Antwort eines Aufrufs mittels HTTP GET.

```
{
  "name": "Selection1"
}
```

Listing 27: Antwort des Backend auf die Ressource /dropSelected

- `/checkbox`

  Bei der Ressource `/checkbox` wird ein Objekt zurückgegeben, dass die Daten für eine einzelne Checkbox beinhaltet. Es wird eine ID, der Name und das `checked` Attribut zurückgegeben. Beim Speichern wird stets nur das `checked` Attribut aktualisiert. Im nachstehenden Listing wird das Objekt dargestellt.

```
{
  "id": 1,
  "name": "Checkbox",
  "checked": true
}
```

Listing 28: Antwort des Backend auf die Ressource /checkbox

- `/input`

  Die Ressource Input gibt die Daten für das HTML-Element `<input>` zurück. Beim Speichern wird im Objekt der entsprechende Text aktualisiert. Das folgende Listing zeigt die Antwort des Backend bei einem HTTP GET Aufruf.

```
{
  "string": "inputString"
}
```

Listing 29: Antwort des Backend auf die Ressource /input

- `/text`

  Wie zuvor bei `/input` beschrieben wird auch der Text für das mehrzeilige Element gespeichert. Der einzige Unterschied ist, dass im Objekt der Variablenname `text` verwendet wird, wie im folgenden Listing dargestellt.

```
{
  "text": "Lorem ipsum dolor sit amet, consetetur ..."
}
```

Listing 30: Antwort des Backend auf die Ressource /text

6.2.8 Style

Jede der Basisanwendungen hat eine CSS-Datei, in welcher der Style für die gesamte Anwendung definiert ist. Das Ziel der Portierungen ist den Style ohne Änderungen zu übernehmen. Entsprechende Anpassungen werden in den Abschnitten der Portierungen angeführt. Die genaue Darstellung der Anwendung ist in ANHANG A - Layout der Basisanwendung in Form von Bildern festgehalten.

6.3 Methoden

In diesem Abschnitt wird einerseits das Konzept angeführt, welches für die Portierungen verwendet wird, sowie andererseits das schrittweise Vorgehen im Code beschrieben.

6.3.1 Konzept der Portierungen

Für alle Portierungen wird eine Migration angestrebt, siehe Abschnitt 5.1.3. Der Fokus dabei liegt darauf, so viel Code wie möglich ohne Änderungen zu übernehmen. Die Portierungen selbst werden Top-Down durchgeführt. In diesem Fall bedeutet das, dass mit der index.html Datei und allgemeinen Teilen begonnen wird. Danach werden schrittweise die einzelnen Components mit den Formularinhalten portiert.

6.3.2 Bewertungsschema

Um die Portierungen objektiv und einheitlich zu bewerten, wird das nachfolgende Bewertungsschema definiert, das in Tabelle 1 dargestellt ist. In der Tabelle ist die Portierbarkeit mit jeweils einem Bewertungsfaktor festgehalten. Je einfacher die Portierung ist, desto höher ist der Bewertungsfaktor. Somit bedeutet eine hohe Bewertung eine bessere Portierbarkeit. Niedrige Bewertungen zeigen, dass die Portierung entweder nicht möglich oder nur mit entsprechendem Aufwand durchführbar ist.

Portierbarkeit	Bewertungsfaktor
Nicht möglich	0
Hoher Aufwand	1
Mittlerer Aufwand	2
Geringer Aufwand	3
Minimaler Aufwand	4
Ohne Änderungen möglich	5

Tabelle 1: Bewertungsschema zur Bewertung der Portierungen

Nach Maniraho (2018) wird empfohlen bei Portierungen eine Bottom-Up Strategie zu verwenden, bei der zuvor die unterstehenden Components portiert werden, danach erst die Übergeordneten. Wie beschrieben wird für die vorliegende Arbeit jedoch eine Top-Down Strategie verwendet. Dies wird dadurch begründet, dass bei dieser Vorgehensweise bereits zu Beginn eine lauffähige Anwendung entwickelt wird, die danach entsprechend weiterentwickelt werden kann (Maniraho, 2018).

Zusätzlich empfiehlt Maniraho (2018) die Aufteilung der Inhalte in Components, Events, State und Forms. Aufgrund der unterschiedlichen Aufgabenstellung wird für diese Arbeit jedoch eine andere Aufteilung gewählt. Der Grund dafür ist, dass die Portierungen mit der folgenden Aufteilung objektiver verglichen werden können. Zusätzlich werden daraus allgemeine Informationen über alle Portierungen abgeleitet, beispielsweise ob bestimmte Teile aller Basisanwendungen nicht portierbar sind.

Die Bewertung der Portierungen wird wie folgt aufgeteilt:

- Allgemeine Inhalte

  In diesem Teil wird die Portierbarkeit von Allgemeinen Inhalten bewertet, die spezifisch für die Webtechnologie sind. Hierzu zählt der Style, der Aufbau der index.html Datei und der Aufbau der Anwendung.

- Routing

  Bei allen Basisanwendungen ist das Routing zwischen den Seiten mit externen Libraries implementiert. In diesem Teil wird überprüft ob Inhalte bei den Portierungen übernommen werden können.

- Templates

  Da die Webtechnologien stets eine unterschiedliche Syntax zur Beschreibung von Templates verwenden, wird dieser Teil getrennt bewertet.

- Struktur der Components

  Die Webtechnologien haben einen unterschiedlichen Datenfluss in den Components, siehe 4.2.11, 4.3.3 und 4.4.6. Aus diesem Grund wird der Aufbau der Struktur der Components getrennt bewertet.

- Logik der Components

  Getrennt von dem Aufbau wird die Logik bewertet. Der Hintergrund dafür ist, dass die Logik in den Components in reinem JavaScript geschrieben ist und somit als eigener Teil bewertet wird.

- Kommunikation mit dem Backend

  Der letzte Teil bewertet die Portierbarkeit der Kommunikation mit dem Backend. Dies ist notwendig da Angular.js, wie in Abschnitt 4.2.4 beschrieben, ein eigenes Service bietet, um die Kommunikation abzuwickeln. React.js und Vue.js sind jedoch auf keine Library beschränkt, siehe Abschnitt 4.3 sowie 4.4. Aus diesem Grund wird dieser Teil getrennt bewertet.

Für jede Portierung werden somit sechs Teile mit jeweils einem Bewertungsfaktor von null bis fünf bewertet. Die Bewertung der einzelnen Teile wird anschließend aufsummiert. Für jede Portierung sind maximal 30 Punkte erreichbar.

Da jede Webtechnologie in die zwei Anderen portiert wird, wird zusätzlich noch eine zusammenfassende Bewertung pro Webtechnologie angeführt. Hierfür werden die Ergebnisse der Portierungen von der Webtechnologie zu den anderen addiert und der Mittelwert gebildet. Der gleiche Wert wird für die Portierungen zu der Webtechnologie hin berechnet. Durch diese zwei Werte soll die allgemeine Portierbarkeit beurteilt werden.

6.3.3 Praktischen Vorgehen

In diesem Abschnitt wird ein einheitliches Schema definiert in welcher Reihenfolge bei den Portierungen vorgegangen wird. Dieses Schema ist von den zuvor definierten Teilen der Bewertung abgeleitet. Jedoch wird an dieser Stelle angemerkt, dass Portierungen von diesem Schema abweichen können, falls dies als effizienter empfunden wird oder andere Gründe dafürsprechen. Diese Fälle werden in den entsprechenden Abschnitten der Analysen angeführt.

1. Abschätzung des Aufwands und der Ähnlichkeit

   Bevor mit der Portierung begonnen wird, wird eine Abschätzung durchgeführt, ob eine Portierung zu aufwändig ist. Zusätzlich wird die Ähnlichkeit des Codes mittels der vorhandenen Basisanwendungen verglichen. In Kombination dienen diese zwei Erkenntnisse als erster Richtwert, ob eine Portierung möglich oder eine Neuentwicklung notwendig ist.

2. Portierung der allgemeinen Inhalte

   Im nächsten Schritt werden allgemeine Inhalte portiert. Begonnen mit der index.html und der Ordnerstruktur. Danach werden allgemeine Inhalte wie Konfigurationen oder Definitionen von API-Routen portiert. Das Ziel bei diesem Schritt ist eine lauffähige Anwendung durch die Portierung zu erstellen, ohne tiefergehende Inhalte wie ausprogrammierte Components.

3. Routing

   Nachdem eine lauffähige Anwendung ohne Inhalt erstellt wurde, wird das Routing portiert. In diesem Schritt wird mit der Definition des HTML Templates begonnen, danach die entsprechende Logik für das Routing portiert, stets mit dem Ziel so viel Sourcecode wie möglich unverändert wiederzuverwenden.

4. Components mit Templates und Controller

   Im nächsten Schritt werden die Components mit den Formularinhalten ohne Kommunikation mit dem Backend portiert. Die Logiken in den Templates und der Datenaustausch mit dem Controller werden vorerst über statische Variablen implementiert. Hier steht die Portierung der Logiken im Vordergrund mit den verschiedenen Directives und Expressions der Webtechnologien, siehe Abschnitt 4.2.6, 4.2.7, 4.3.1 sowie 4.4.3.

5. Controller und Kommunikation mit Backend

   Im letzten Schritt werden die statischen Variablen durch dynamische Aufrufe mit dem Backend ersetzt. Falls möglich werden entsprechende Libraries wiederverwendet.

In den entsprechenden Abschnitten der Portierungen wird stets der genaue Ablauf und das Vorgehen dokumentiert.

6.4 Aufbau der Basisanwendungen

In diesem Abschnitt wird auf den Aufbau der Basisanwendungen eingegangen. Bei jeder Anwendung handelt es sich um eine SPA, siehe Abschnitt 3.2. Jede Basisanwendung hat die zuvor definierten Anforderungen implementiert. An dieser Stelle wird angemerkt, dass sich die vorliegende Arbeit ausschließlich mit den JavaScript Code sowie den HTML- und CSS-Dateien befasst. Auf jegliche Konfiguration von Webserver, des Backend oder sonstigen Tools wird nicht eingegangen.

6.4.1 Angular.js

Die Angular.js Basisanwendung weist die folgende Ordnerstruktur auf:

- `public`

  Hier befinden sich die index.html Datei sowie das Favicon.

- `src`
  - `common`

    In diesem Ordner sind die Konfiguration von API-Routen sowie die index.css Datei abgelegt.

- `components`

  Hier befinden sich für alle Components der Anwendung Unterordner, in denen jeweils das Template in einer HTML-Datei und die Component in einer JavaScript-Datei definiert sind.

- `services`

  In diesem Ordner befinden sich für jede Component ein eigener Service, der für die Kommunikation mit dem Backend genutzt wird.

- `views`

  Wie für die Components befindet sich in diesem Ordner jeweils ein Unterordner für die Seiten der Anwendung. Hier sind die entsprechenden Views und die Templates definiert.

In der index.html ist zuvor im `<head>` der Titel der Seite, der Link zum Favicon, das Charset und die Verlinkung auf JavaScript gesetzt. Der `<body>` beschränkt sich auf ein Element für die Navigation, sowie eines für die UI, in der die Components gerendert werden.

Zu Beginn wird Angular.js konfiguriert. Hierfür werden die Module der Views registriert und die Konfiguration für die Navigation gesetzt. In den Views werden die definierten Seiten der Anwendung erstellt und dort die entsprechende Component mit den Formularinhalt geladen und aufgerufen. Beispielsweise wird die `CheckboxView` als grundlegendes Template für die Seite `/checkbox` verwendet. In dieser View wird die Component `MyCheckbox` aufgerufen, in der die Logik und das Template der Checkbox-Component implementiert sind.

Eine Besonderheit bei der Implementierung der Basisanwendung in Angular.js ist die Verwendung von einer persistenten Component für das Menü. Grundsätzlich unterscheidet diese sich nur dadurch von anderen Components, dass diese nicht neugeladen wird und dauerhaft als Navigation angezeigt wird, sowie direkt in der index.html angegeben wird. Zusätzlich wird die Angular.js Library uirouter verwendet, welche für das Routing zwischen den Seiten der Anwendung verwendet wird.

6.4.2 React.js

Die folgende Auflistung zeigt die Ordnerstruktur der React.js Basisanwendung:

- `public`

  Wie bei Angular.js befinden sich hier die index.html sowie das Favicon.

- `src`

  - `common`

    In diesem Ordner sind die API-Routen für die Kommunikation zum Backend definiert, sowie die allgemeine index.css abgelegt.

- `components`

  Anders als bei Angular.js wird bei der React.js Basisanwendung kein Ordner für jede Component angelegt. Stattdessen wird nur eine JavaScript-Datei für jede Component angelegt.

Zu Beginn wird in der index.html das Charset, der Titel sowie die Metadaten definiert. Anders als bei Angular.js wird im `<body>` Tag nur ein `<div>` Element mit der id `root` angegeben. Dieses wird in der index.js referenziert, um dort die gesamte Anwendung zu rendern.

Ebenso anders als bei Angular.js, ist bei React.js keine Konfiguration notwendig. In der app.js werden die Routen für die Navigation zwischen den Seiten definiert und die entsprechende Component gesetzt, die dabei aufgerufen wird.

Es wird das in Abschnitt 4.3.1 beschriebene JSX verwendet. Somit ist für die Definition einer Component eine einzelne JavaScript-Datei ausreichend. In dieser wird die Logik implementiert, in der die Daten geladen und gespeichert werden als auch das Template definiert.

Die Besonderheiten der React.js Basisanwendung sind, dass einerseits die Library react-router-dom verwendet wird und andererseits die Verwendung der Library axios. Bei react-router-dom handelt es sich um eine Library welche die Navigation der Seiten in einer SPA übernimmt. Bei axios handelt es sich um einen JavaScript HTTP-Client, der für die Kommunikation mit dem Backend verwendet wird. Der Grund für die Verwendung einer Library ist, dass React.js im Vergleich zu Angular.js keinen HTTP-Client zur Verfügung stellt, siehe Abschnitt 4.2.4 sowie 4.3.

6.4.3 Vue.js

Die Vue.js Basisanwendung hat die folgende Ordnerstruktur:

- `public`

  Wie bei den beide vorherigen Basisanwendungen sind in diesem Ordner die index.html sowie das Favicon abgelegt.

- `src`
  - `common`

    Analog zur React.js Basisanwendung sind hier die API-Routen zur Kommunikation zum Backend definiert, sowie die allgemeine index.css.

  - `components`

    In diesem Ordner ist für jede Component eine eigene Datei angelegt, in der Template sowie die Component selbst definiert sind.

  - `router`

    In diesem Ordner befindet sich eine JavaScript-Datei, in der die Routen für die Seiten Navigation der Anwendung definiert sind.

- `views`

  Analog zu Angular.js wird für jede Component eine darüberstehende View angelegt.

Wie bei den anderen beiden Basisanwendungen sind in der index.html allgemeine Metadaten definiert. Der `<body>` ist vergleichbar mit dem Aufbau von React.js. Hier wird nur ein `<div>` Element mit der id `app` angelegt, welches in der main.js referenziert wird, um dort die Anwendung zu laden.

Wie bei React.js ist bei Vue.js ist keine Konfiguration notwendig. Zu Beginn wird die Navigation definiert, wobei in der index.js die Routen der entsprechenden Views verlinkt werden. In der View wird danach die entsprechende Component geladen und im Template dargestellt. Die Component selbst sind als Single File Component in einer Datei zusammengefasst, in der das Template und die Logiken definiert sind, siehe Abschnitt 4.4.10.

Die Basisanwendung in Vue.js verwendet die Library vue-router sowie axios. Letztere wird wie bei React.js für die Kommunikation mit dem Backend verwendet. Die Library vue-router wird für die Navigation innerhalb der Anwendung verwendet.

6.5 Analyse von Angular.js

In diesem Abschnitt sind die Analysen der Portierungen von Angular.js zu React.js, sowie zu Vue.js angeführt. Bei jeder Portierung werden jeweils nur die Unterschiede festgehalten, keine Schritt-für-Schritt Anleitung. Die erarbeiteten Hilfestellungen für Portierungen werden im nachfolgenden Kapitel 7 angeführt.

6.5.1 Portierung zu React.js

Im Vergleich zu Angular.js benötigt React.js weniger Ordner, um die Anwendung übersichtlich und strukturiert zu halten. Aus diesem Grund kann die Ordnerstruktur auf die Struktur der React.js Basisanwendung reduziert werden. Allgemein gibt es in React.js keine Modules und somit auch keine Notwendigkeit diese zu registrieren und zu konfigurieren, was zu weniger Angular.js Code führt.

In der index.html können alle Metadaten im `<head>` erhalten bleiben. Jedoch werden die HTML-Elemente der Anuglar.js Basisanwendung `<ui-view>` und `<navigation>` zu einem `<div>` vereinfacht. Die allgemeine CSS-Datei kann ohne Änderungen in den `common` Ordner übernommen werden. Auch die Konfigurationsdatei der API-Routen kann übernommen werden. Die einzige Änderung dabei ist, dass die URLs der Anwendungs-Routen entfernt werden, da diese in React.js keine Anwendung finden. Da in React.js auch keine Views benötigt werden, können die Views der Angular.js Basisanwendung verworfen werden.

In den Components selbst werden jedoch Änderungen vorgenommen, im Folgenden aufgelistet:

- Die Templates werden mit dem Controller in einer Datei zusammengefasst.
- Die Expressions im Template werden mit `{ variable }` anstelle von `{{ variable }}` aufgerufen.
- Die Methoden im Template werden wie Variablen aufgerufen, ohne Klammern am Ende und ohne Anführungszeichen. Beispielsweise wird die `submit()` Methode in Angular.js mit `ng-submit="$ctrl.submit()"` aufgerufen, in React.js jedoch mit `onSubmit={this.submit}`.
- Alle Directives werden angepasst. So verwendet React.js beispielsweise `onSubmit` anstelle des Angular.js Directive `ng-submit`.
- Die Verwendung von `ng-attr` ist nicht notwendig, Attribute können in React.js direkt gesetzt werden.
- Das Directive `ng-model` muss manuell über den State implementiert werden. Es wird eine Methode implementiert, die das Verhalten bei Änderungen von den Formularinhalten definiert.
- Es ist nicht möglich mehrere Elemente mittels dem Directive `ng-repeat` zu erstellen. React.js bietet keine eigene Möglichkeit, um mehrere Elemente mittels For-Schleife zu erstellen. Als Alternative wird die JavaScript-Methode `array.map()` verwendet.

Zusätzlich wird die Logik des Controllers angepasst. Das Laden der Daten wird in die Lifecycle Methoden verschoben. Der State von React.js wird von Grund auf neu implementiert, siehe Abschnitt 4.3.4 Component Lifecycle.

Die Kommunikation mit dem Backend, die in der Angular.js Basisanwendung über Services implementiert wurde, kann in die Component verschoben werden. Grundsätzlich ist es auch in React.js möglich eigene Services zu implementieren, jedoch wird dadurch kein Vorteil gegenüber der Implementierung mittels einer Library gewonnen. Der HTTP-Client von Angular.js hat eine ähnliche Syntax wie axios, weshalb die Methoden selbst mit geringem Aufwand portierbar sind.

Das Routing zwischen den Seiten wird neuentwickelt. Die Unterschiede zwischen den Implementierungen der Webtechnologien sind zu groß, um Teile davon portieren zu können. Das Routing wird mit der zuvor verwendeten Library react-router-dom neu implementiert.

6.5.2 Portierung zu Vue.js

Bei der Portierung von Angular.js zu Vue.js wird ein Großteil der Ordnerstruktur übernommen. Einzig der Ordner der Services kann je nach Implementierung entfernt werden. Auf die Implementierung der Services wird im Verlauf dieses Abschnitts genauer eingegangen.

Alle Metadaten im `<head>` der index.html werden übernommen. Im `<body>` kann die `<navigation>` und `<ui-view>` zu einem `<div>` vereinfacht werden. Die allgemeinen Inhalte, die CSS-Datei sowie die Definition der API-Routen werden ohne Änderungen übernommen. Einzig

die Anwendungs-URLs werden entfernt. Grund dafür ist, dass das Routing zwischen den Seiten nicht übernommen werden kann.

Vue.js verwendet keine Modules, somit ist die Konfiguration dieser nicht notwendig. Es ist nur die Erstellung der Navigation und danach die Implementierung der Views mit den Components notwendig. Das Routing selbst wird mittels vue-router neu implementiert. Bei der Erstellung der Views und Components werden die HTML- und JavaScript-Dateien von Angular.js in jeweils eine Datei in einer Single File Component zusammengefasst, siehe Abschnitt 4.4.10. Die Templates der Components sind mit geringem Aufwand portierbar. Der Grund dafür ist, dass Vue.js und Angular.js eine ähnliche Syntax verwenden, siehe 4.2.5 sowie 4.4.2.

In Listing 31 sind die Templates der Checkbox-Component der Basisanwendungen in Angular.js und Vue.js festgehalten, um die Unterschiede darzustellen. Die Zuweisung der Attribute mit dem Directive `ng-attr` kann durch `v-bind` ersetzt werden. Die Schreibweise der Variablen ändert sich nur bei den Attributen, diese können ohne geschwungen Klammern angegeben werden. Die Variable, in welcher der Zustand des Elements gespeichert ist, kann mit geringem Aufwand von `ng-model` auf `v-model` ausgetauscht werden, es müssen keine zusätzlichen Methoden dafür implementiert werden. Nicht im Listing dargestellt sind For-Schleifen. Hier kann das `ng-repeat` durch ein `v-for` ersetzt werden, um die Funktionalität zu portieren.

```
<!-- Angular.js -->
<form ng-submit="$ctrl.submit()">
  <input
    type="checkbox"
    ng-attr-id="{{$ctrl.id}}"
    ng-attr-name="{{$ctrl.name}}"
    ng-model="$ctrl.checked"
  />
  <label ng-attr-for="{{$ctrl.id}}">{{ $ctrl.name }}</label>
  <button type="submit">Save</button>
</form>

<!-- Vue.js -->
<form v-on:submit.prevent="onSubmit">
  <input
    type="checkbox"
    v-bind:id="id"
    v-bind:name="name"
    v-model="checked"
  />
  <label v-bind:for="id">{{ this.name }}</label>
  <button type="submit">Save</button>
</form>
```

Listing 31: Gegenüberstellung von Templates in Angular.js und Vue.js

Die Kommunikation zum Backend kann entweder wie in Angular.js auf eigene Services ausgelagert werden oder in die Component selbst implementiert werden. Da die Auslagerung auf Services mehr Aufwand ist, wird die Kommunikation mit dem Backend in den Components implementiert. Die Syntax des Angular.js HTTP-Clients ähnelt der Syntax von axios, weshalb die Methoden mit geringem Aufwand portiert werden können.

6.6 Analyse von React.js

Dieser Abschnitt behandelt die Analyse der Portierungen von React.js zu Angular.js und Vue.js.

6.6.1 Portierung zu Angular.js

Die Ordnerstruktur von Angular.js ist im Vergleich zu der von React.js umfangreicher. Dies liegt daran, dass Angular.js für einen übersichtlichen Aufbau mehr Struktur und Code benötigt, sowie Modules verwendet, siehe Abschnitt 4.2.1.

In der index.html werden alle Metadaten im `<head>` Element übernommen. Das `<div>` Element, in dem React.js die Anwendung rendert, wird in eine `<ui-view>` geändert. Allgemeine Informationen wie die API-Routen zum Backend werden übernommen. Die Anwendungs-URLs für das Routing in Angular.js müssen ergänzt werden. Die allgemeine CSS-Datei wird ohne Änderungen portiert.

Da React.js JSX verwendet sind Template und Logik in einer Datei zusammengefasst. Deshalb muss der Code in die zwei Teile aufgeteilt werden, wodurch die Ordnerstruktur dieser Portierung umfangreicher wird.

Grundsätzlich ist es möglich auf Views zu verzichten, jedoch wirkt sich dies negativ auf die Übersicht des Codes in Angular.js aus, weshalb zusätzliche Views bei dieser Portierung erstellt werden. Das Routing zwischen den Seiten kann aufgrund der unterschiedlichen Libraries nicht übernommen werden und muss neuentwickelt werden.

Im Detail kann kaum Code ohne Veränderung einer React.js Component übernommen werden, wie im nachfolgenden Listing dargestellt. In Listing 32 wird das Template der Checkbox Component mit dem Template in React.js gegenübergestellt. Die grundlegenden HTML-Elemente werden übernommen. Es werden bei den Attributen und Methoden die folgenden Anpassungen durchgeführt:

- Die Variablen im Template werden mit `{{ variable }}` anstelle von `{ variable }` geschrieben.
- Der React.js State wird zu einer eigenen Variable im Controller, im Listing die Variable `ctrl`.
- Attribute wie beispielsweise `id` müssen über `ng-attr` gesetzt werden.
- React.js verwendet eine Methode, um die Information über die Variablen des Status zu ändern. Im Listing verwendet React.js die Methode `this.handleChange()`, um die

Variable `checked` im State zu ändern. Bei Angular.js wird hierfür die Variable über `ng-model` übergeben.

- In React.js wird die `array.map()` Methode verwendet, um dynamisch mehrere Elemente zu erstellen. In Angular.js wird dies mittels `ng-repeat` kompakter dargestellt.

```
<!-- Angular.js -->
<form ng-submit="$ctrl.submit()">
  <input
    type="checkbox"
    ng-attr-id="{{$ctrl.id}}"
    ng-attr-name="{{$ctrl.name}}"
    ng-model="$ctrl.checked"
  />
  <label ng-attr-for="{{$ctrl.id}}">{{ $ctrl.name }}</label>
  <button type="submit">Save</button>
</form>

<!-- React.js -->
<div id="Checkbox">
 <form onSubmit={this.handleSubmit}>
  <input
    type="checkbox"
    id={this.state.id}
    name={this.state.name}
    onChange={this.handleChange}
    checked={this.state.checked}
  />
  <label htmlFor={this.state.id}>{this.state.name}</label>
  <button type="submit">Save</button>
 </form>
</div>
```

Listing 32: Gegenüberstellung von Templates in Angular.js und React.js

Die Logik des Controllers wird angepasst. Grundsätzlich kann der State von React.js in Angular.js durch eine Variable im Controller angenähert werden, die zugrunde liegenden Konzepte sind jedoch unterschiedlich, siehe Abschnitt 4.2.2 sowie 4.3.3. Dies kann je nach Anforderung, zu ungewolltem Verhalten führen. Auch die Lifecycle-Methoden können nicht übernommen werden.

Die Methoden für die Kommunikation mit dem Backend werden mit geringem Aufwand portiert. In React.js wird die Library axios verwendet, deren Syntax des HTTP-Client in Angular.js ähnelt.

6.6.2 Portierung zu Vue.js

Bei der Portierung von React.js zu Vue.js wird zu Beginn die Ordnerstruktur um einen Ordner ergänzt in dem die Views definiert werden. Die index.html kann ohne Änderungen übernommen

werden. Einzig beim Initialisieren von Vue.js wird auf das `<div>` Element mit der id `root` gemounted, anstelle von der id `app`. Auch die API-Routen und die CSS-Datei können ohne Änderung übernommen werden.

Da die React.js react-router-dom Library nicht zu Vue.js portierbar ist, muss das Routing neu implementiert werden. Es wird die Library vue-router verwendet. Hierfür wird zusätzlich eine Datei mit den Routen angelegt. Da die Basisanwendung in React.js keine Views implementiert hat, werden diese neu entwickelt.

Das Template von React.js, das in der `render()` Methode definiert wird, kann in ein Vue.js Template übernommen werden, jedoch sind hier noch Anpassungen notwendig, wie im späteren Verlauf dieses Abschnitts angeführt.

Die Logik und Daten können grundsätzlich von React.js übernommen werden. Hier entspricht die Definition des State im Konstruktor in React.js der Definition der Daten in Vue.js. Die Lifecycle-Methode `ComponentDidMount()` entspricht der `mounted()` Methode in Vue.js.

Aufgrund der Verfügbarkeit der Library axios in Vue.js, werden die Methoden zur Kommunikation mit dem Backend mit minimalen Änderungen übernommen. Der Unterschied ist hierbei, dass anstelle, dass der State gesetzt wird, in Vue.js mittels `this` auf die Daten der Component zugegriffen wird.

Der meiste Aufwand fällt beim Umwandeln des Templates an. Die Attribute der HTML-Elemente werden mittels `v-bind` oder dessen Kurzform gesetzt. Die Variablen werden mit der Schreibweise `{{ variable }}` aufgerufen. Das `onSubmit` muss durch ein `v-on:submit` ersetzt werden. For-Schleifen sind durch die Portierung zu Vue.js mittels `v-for` kürzer darstellbar als in React.js.

6.7 Analyse von Vue.js

In den folgenden zwei Abschnitten werden die Portierungen von Vue.js zu Angular.js und React.js analysiert.

6.7.1 Portierung zu Angular.js

Die Ordnerstruktur der Basisanwendung von Vue.js lässt sich gut in den Aufbau von Angular.js übernehmen. Beide besitzen die Ordner `components` und `views`. Da das Routing zwischen den Seiten aufgrund der unterschiedlichen Libraries nicht portierbar ist, wird der Ordner `router` der Vue.js Basisanwendung verworfen.

In der index.html können alle Metadaten übernommen werden. Im HTML-Element `<body>` muss jedoch bei Verwendung der Angular.js Library uirouter das `<div>` Element auf `<ui-view>` und `<navigation>` erweitert werden. Allgemeine CSS-Inhalte werden übernommen, sowie die definierten API-Routen zur Kommunikation mit dem Backend. Da sich bei Vue.js Components das Template sowie Logik in einer Datei befinden, werden diese in jeweils in ein HTML-Template und eine JavaScript-Datei für den Angular.js Controller aufgeteilt.

Die Views werden für Angular.js umgeschrieben, jedoch müssen hierfür Modules erstellt werden, die entsprechend konfiguriert werden. Zusätzlich werden die Views auch in jeweils ein HTML-Template und eine JavaScript-Datei aufgeteilt. Die Kommunikation mit dem Backend kann entweder als eigener Service implementiert werden oder direkt in dem Controller in Angular.js definiert werden.

Die Portierung der Templates ist aufgrund der Ähnlichkeiten der Webtechnologien mit geringem Aufwand möglich. Die Zuordnung von Attributen mittels `v-bind` kann auf `ng-attr` geändert werden. Die Variablen selbst werden mit der Syntax `{{ variable }}` aufgerufen, hier müssen keine Änderungen durchgeführt werden. Die For-Schleife `v-for` wird durch `ng-repeat` ersetzt.

Bei der Erstellung des Controllers in Angular.js fällt auf, dass nur wenige Änderungen notwendig sind. Anstelle mittels `this` auf die Daten des Vue.js Components zu referenzieren, kann eine Variable im Controller in Angular.js definiert werden und diese dafür verwendet werden. Werden zusätzlich noch die API-Aufrufe in den Controller direkt implementiert, so können die API-Aufrufe mit minimalen Änderungen in den Angular.js Controller übernommen werden.

6.7.2 Portierung zu React.js

Bei der Portierung von Vue.js zu React.js kann die Komplexität der Ordnerstruktur reduziert werden, da nur noch die Ordner `common` und `components` notwendig sind. Die index.html ist ohne Anpassungen in React.js verwendbar. Einzig bei der Initialisierung von React.js muss das entsprechende Element mit der id `app` angegeben werden. Auch die CSS-Datei und die API-Routen werden ohne Anpassungen portiert. Da in React.js keine Views notwendig sind, werden diese entfernt. Analog zu allen anderen Portierungen ist das Routing nicht portierbar und muss neu implementiert werden, in diesem Fall mit der Library react-router-dom.

Bei den Components selbst kann das Vue.js Template in der `render()` Methode des React.js mit Anpassungen übernommen werden, die Details dazu sind im nachfolgendem Listing angeführt.

Die Definition der Daten einer Vue.js Component ähnelt dem State von React.js, der im Konstruktor erstellt wird, siehe Abschnitt 4.3.3 und 4.4.4. Alle definierten Methoden der Vue.js Component können in die React.js Klasse portiert werden. Auch der Lifecycle kann von Vue.js zu React.js übernommen werden. Die verwendete Methode in `mounted()` kann in React.js in der `ComponentDidMount()` Methode aufgerufen werden. Da auch bei dieser Portierung die Library axios verwendet wird, werden die Methoden zur Kommunikation mit dem Backend mit geringem Aufwand portiert.

Listing 33 stellt das Template der Checkbox-Component der Basisanwendung von React.js mit dem gleichen Template von Vue.js gegenüber. Die Attribute können nun direkt angegeben werden, ohne `v-bind`. Da React.js jedoch keine Möglichkeiten wie das `v-model` bietet, wird eine eigene Methode implementiert, in der das Verhalten bei Änderungen definiert wird, im Listing als `this.handleChange` angeführt. Die Expressions, mit denen die Variablen im Template aufgerufen werden, werden von zwei geschweiften Klammern auf eine reduziert.

```
<!-- React.js -->
<form onSubmit={this.handleSubmit}>
  <input type="checkbox"
    id={this.state.id}
    name={this.state.name}
    onChange={this.handleChange}
    checked={this.state.checked}
  />
  <label htmlFor={this.state.id}>{this.state.name}</label>
  <button type="submit">Save</button>
</form>

<!-- Vue.js -->
<form v-on:submit.prevent="onSubmit">
  <input type="checkbox"
    v-bind:id="id"
    v-bind:name="name"
    v-model="checked"
  />
  <label v-bind:for="id">{{ this.name }}</label>
  <button type="submit">Save</button>
</form>
```

Listing 33: Gegenüberstellung von Templates in React.js und Vue.js

6.8 Bewertung und Erkenntnisse der Portierungen

Dieser Abschnitt behandelt die Beurteilung der Portierungen nach dem in Abschnitt 6.3.2 definierten Schema. Im ersten Schritt wird für jede Webtechnologie die Bewertung formuliert und danach zusammenfassende Tabellen und Grafiken angeführt. Wie im Schema definiert, wird jede Webtechnologie in jeweils sechs Teilen von null bis fünf beurteilt. Die Bewertung der Teile sind pro Portierung in jeweils einer Grafik dargestellt. An dieser Stelle wird angemerkt, dass in allen Grafiken die Beurteilung des Routings nicht dargestellt wird, da alle Portierungen in diesem Bereich mit null Punkten bewertet wurden.

6.8.1 Angular.js zu React.js

Die Portierung von Angular.js zu React.js wird mit 9 von maximal 30 möglichen Punkten bewertet. Die Struktur der Angular.js Basisanwendung kann durch die Portierung vereinfacht werden, da die Modules und Views nicht benötigt und deshalb verworfen werden. Die Services werden mit hohem Aufwand in die entsprechende Component portiert. Auch die Definition von Modules wird in React.js nicht benötigt. Die Components selbst werden bei dieser Portierung in einer Datei zusammengefasst, wodurch mehr Übersichtlichkeit gewonnen wird.

Der Aufwand liegt bei dieser Portierung bei der Anpassung der Templates und Controller. Da React.js Components anders aufgebaut sind und auch Konzepte wie der State implementiert werden müssen, ist der Aufwand der Portierung bei den Components entsprechend hoch.

In Abbildung 11 sind die Ergebnisse der Teilbewertung der Portierung von Angular.js zu React.js grafisch dargestellt.

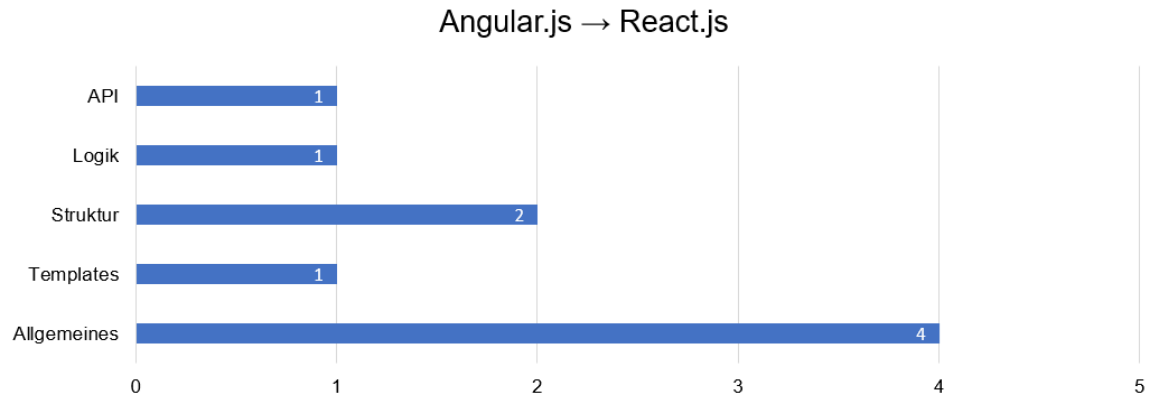

Abbildung 11: Teilbewertung der Portierung von Angular.js zu React.js

6.8.2 Angular.js zu Vue.js

Die Portierung von Angular.js zu Vue.js wird mit 13 von 30 Punkten bewertet. Der Aufbau von Vue.js ist weniger umfangreich, es werden keine Modules benötigt und somit auch weniger Code. Auch die ähnliche Syntax bei der Definition von Templates von Angular.js und Vue.js erleichtert diese Portierung und ermöglicht es Templates mit geringem Aufwand zu portieren.

Der größte Aufwand liegt bei der Portierung der Controller und der API. Es kann zwar Code des HTTP-Client zum Teil übernommen werden, jedoch muss die Logik und die API zum Teil neu implementiert werden.

Abbildung 12 zeigt die Ergebnisse der Teilbewertung der Portierung von Angular.js zu Vue.js.

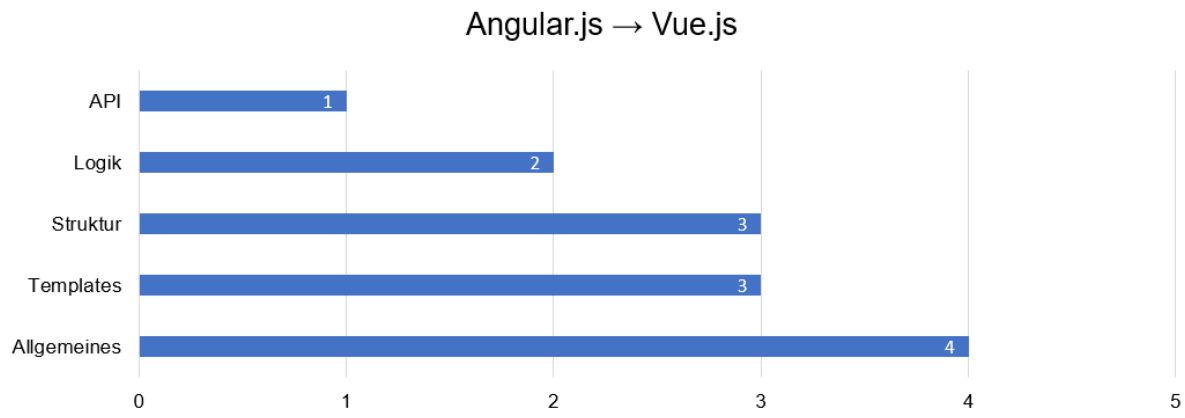

Abbildung 12: Teilbewertung der Portierung von Angular.js zu Vue.js

6.8.3 React.js zu Angular.js

Die Portierung von React.js zu Angular.js wird mit 8 von 30 Punkten bewertet. Allgemeine Inhalte wie die CSS-Datei und die API-Routen sind mit geringem Aufwand portierbar. Zudem können die Methoden für Kommunikation mit dem Backend mit geringem Aufwand portiert werden.

Da sich jedoch die Syntax bei der Erstellung von Templates in Angular.js von der von React.js unterscheidet, fällt hoher Aufwand bei der Portierung dieses Teils an. Auch die Logik im Controller kann nur durch Anpassungen übernommen werden. Zusätzlich können die Modules nicht portiert werden, weshalb die Struktur neu aufgebaut werden muss.

In Abbildung 13 sind die Ergebnisse der Bewertung der Teile der Portierung von React.js zu Angular.js dargestellt.

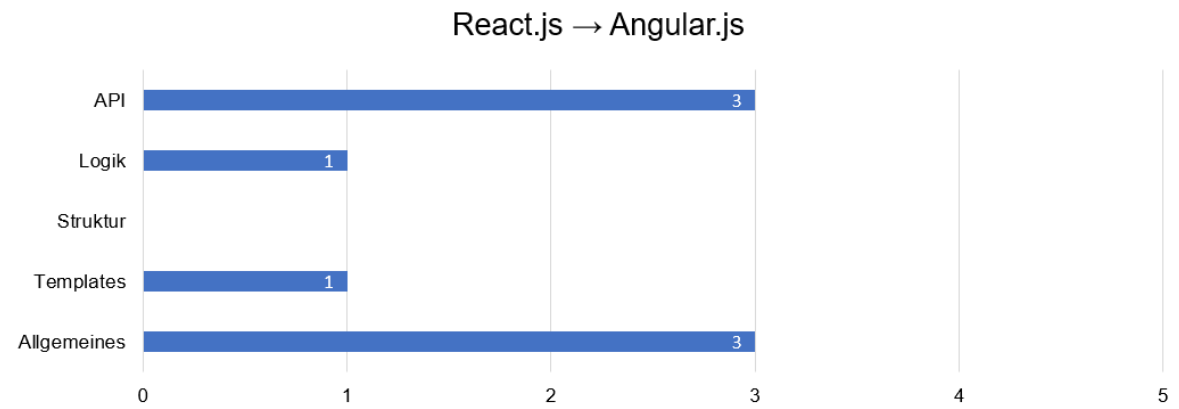

Abbildung 13: Teilbewertung der Portierung von React.js zu Angular.js

6.8.4 React.js zu Vue.js

Die Portierung von React.js zu Vue.js wird mit 15 von 30 Punkten bewertet. Im Gegensatz zur Portierung zu Angular.js fällt bei dieser Portierung Code für die Erstellung und die Konfiguration der Modules weg. Zudem sind Templates und Logik bereits in einer Datei zusammengefasst wodurch die Portierung erleichtert wird. Auch die verwendete Lifecycle-Methode kann übernommen werden. Die Controller und Methoden können mit mittlerem Aufwand portiert werden.

Die Views werden in React.js nicht benötigt, weshalb diese neuentwickelt werden müssen. Zudem fällt mittlerer Aufwand bei der Portierung der Templates an, da die Webtechnologien eine unterschiedliche Syntax verwenden.

Abbildung 14 zeigt die Ergebnisse der Teilbewertung der Portierung von React.js zu Vue.js.

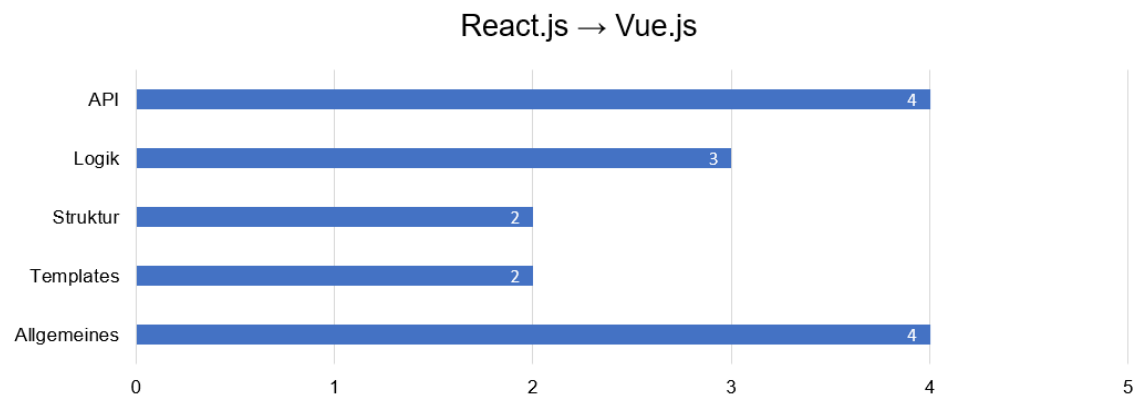

Abbildung 14: Teilbewertung der Portierung von React.js zu Vue.js

6.8.5 Vue.js zu Angular.js

Die Portierung von Vue.js zu Angular.js wird mit 15 von 30 Punkten bewertet. Die Struktur der Anwendung muss in geringem Ausmaß angepasst werden. Die Templates können aufgrund der ähnlichen Syntax ebenso mit mittlerem Aufwand portiert werden, müssen jedoch aus der Single File Component in einzelne Dateien aufgeteilt werden. Die Views können mit geringem Aufwand angepasst werden.

Der Aufwand dieser Portierung liegt bei der Erstellung der Modules in Angular.js. Zudem fällt mittlerer Aufwand für die Portierung der Controller aufgrund des anderen Aufbaus an.

Abbildung 15 stellt die Teilbewertung der Portierung von Vue.js zu Angular.js dar.

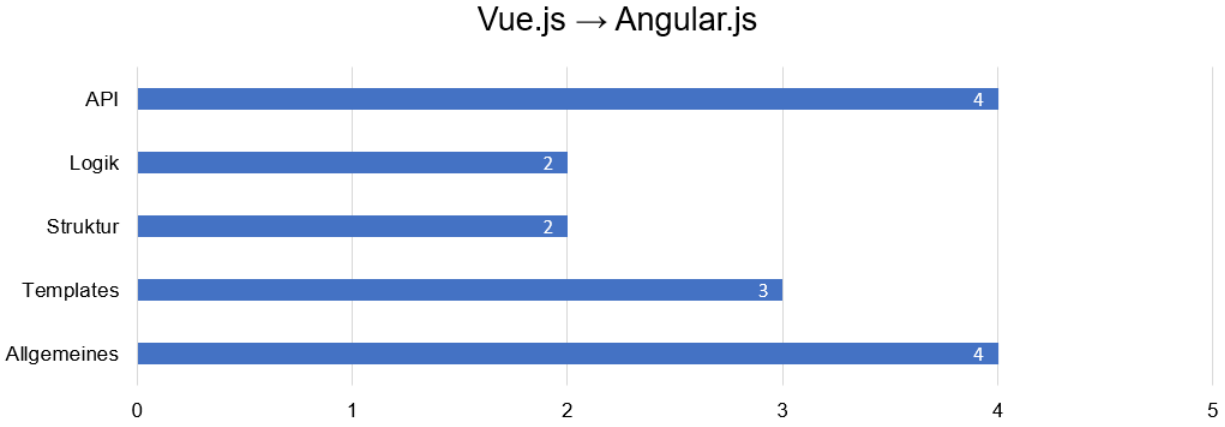

Abbildung 15: Teilbewertung der Portierung von Vue.js zu Angular.js

6.8.6 Vue.js zu React.js

Die Portierung von Vue.js zu React.js wird mit 16 von maximal 30 Punkten bewertet. Die Gründe dafür sind, dass bei dieser Portierung die Ordnerstruktur vereinfacht werden kann, sowie Views vernachlässigt werden können und somit der Aufwand geringer als bei den anderen Portierungen ist. Besonders im Controller sind große Ähnlichkeiten, so können Methoden von Vue.js in die Klasse der Component übernommen werden und die Daten der Component in einen React.js

State umgeschrieben werden. Auch die Kommunikation mit dem Backend kann mit minimalem Aufwand portiert werden.

Der größte Aufwand fällt bei der Portierung der Templates an da sich die Syntax von React.js unterscheidet führt dies zu längerem Code.

Abbildung 16 zeigt die Teilbewertung der Portierung von Vue.js zu React.js.

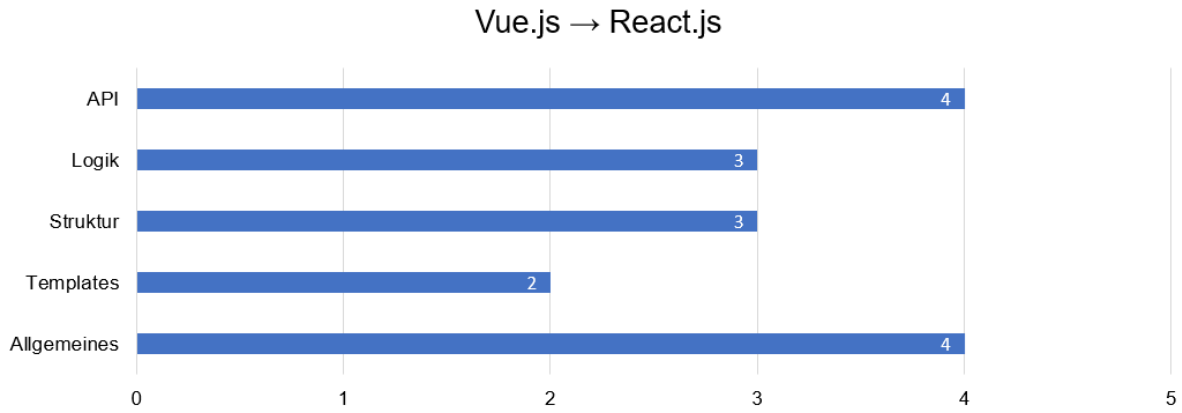

Abbildung 16: Teilbewertung der Portierung von Vue.js zu React.js

6.8.7 Übersicht der Bewertungen

In diesem Abschnitt sind die Bewertungen der einzelnen Teile und die gesamte Bewertung in Tabellen angeführt. Für eine klare Übersicht werden Teil- und Gesamtbewertungen in getrennten Tabellen dargestellt.

Tabelle 2 zeigt die Teilbewertungen aller Portierungen. Die Bewertung erfolgt nach dem in Abschnitt 6.3.2 beschriebenen Schema. Pro Teil wird von null bis fünf bewertet, wobei null für die schlechteste Bewertung und keine Portierbarkeit steht, fünf hingegen bedeutet eine Portierung ohne Aufwand und ist somit die bestmögliche Bewertung. Bei allen Portierungen war es nicht möglich das Routing zu übernehmen, weshalb hier keine Punkte vergeben wurden.

Portierung	Allgemeines	Routing	Templates	Struktur	Logik	API
Angular.js → React.js	4	0	1	2	1	1
Angular.js → Vue.js	4	0	3	3	2	1
React.js → Angular.js	3	0	1	0	1	3
React.js → Vue.js	4	0	2	2	3	4
Vue.js → Angular.js	4	0	3	2	2	4
Vue.js → React.js	4	0	2	3	3	4

Tabelle 2: Übersicht der Teilbewertungen aller Portierungen

In der nachfolgenden Tabelle 3 sind die Gesamtbewertungen pro Portierung angeführt. Pro Portierung ist maximal eine Punktezahl von 30 Punkten zu erreichen.

Portierung	Gesamtbewertung
Angular.js → React.js	9
Angular.js → Vue.js	13
React.js → Angular.js	8
React.js → Vue.js	15
Vue.js → Angular.js	15
Vue.js → React.js	16

Tabelle 3: Übersicht der Gesamtbewertungen aller Portierungen

Abschließend stellt Tabelle 4 die Mittelwerte der Bewertungen der Portierungen nach der Definition im Bewertungsschema dar. Dabei wird der Mittelwert für jede Portierung in diese Webtechnologie, sowie jener aus dieser angegeben. Ein höherer Wert steht für eine bessere Portierbarkeit, ein niedriger Wert für eine schlechtere Portierbarkeit.

Portierung	Mittelwert der Bewertungen
Angular.js → React.js und Vue.js	11
React.js und Vue.js → Angular.js	11,5
React.js → Angular.js und Vue.js	11,5
Angular.js und Vue.js → React.js	12,5
Vue.js → Angular.js und React.js	15,5
Angular.js und React.js → Vue.js	14

Tabelle 4: Mittelwerte der Bewertung der Portierungen pro Webtechnologie

6.8.8 Allgemeine Erkenntnisse

Bei den Portierungen war erkennbar, dass die nachfolgenden allgemeinen Inhalte immer portierbar sind:

- index.html

  Die Metadaten der index.html sind bei allen Technologien ohne Anpassungen portierbar. Einzig bei den Portierungen mit Angular.js werden wegen der Verwendung der Library uirouter zwei HTML-Elemente im `<body>` angepasst.

- Style mittels CSS-Datei

  Die CSS-Datei mit der Definition des Styles kann bei jeder Portierung ohne Änderungen übernommen werden. Bei jeder Portierung kann die Datei mittels Imports geladen und verwendet werden.

- API-Routen

  Da die API-Routen in einem JavaScript-Objekt definiert wurden, kann dieses stets wiederverwendet werden. Bei den Portierungen zu Angular.js werden zusätzlich noch die Anwendungs-Routen für die Library uirouter definiert.

Das Routing zwischen den einzelnen Seiten ist aufgrund der Verwendung unterschiedlicher Libraries nie portierbar und muss stets neu implementiert werden. Aufgrund einer Vielzahl an Libraries, die das Routing vereinfachen, ist der Aufwand für die Implementierung jedoch gering und überschaubar.

Eine Erkenntnis aus den Portierungen ist, dass Templates von Vue.js zu Angluar.js und umgekehrt gut portierbar sind. Die Ähnlichkeiten der Template-Syntax erleichtern die diese Portierungen. Die Portierung der Templates von Vue.js und Angular.js zu React.js war jedoch immer mit Aufwand verbunden. Bei den Components war erkennbar, dass Vue.js und React.js zwar einen unterschiedlichen Aufbau verwenden, der Code jedoch mit geringem Aufwand wiederverwendbar ist.

7 HILFESTELLUNGEN UND RICHTLINIEN

In diesem Kapitel werden Hilfestellungen und Richtlinien angeführt, die bei Portierungen oder bei einem Technologiewechsel in eine der ausgewählten Webtechnologien unterstützen können. An dieser Stelle wird angemerkt, dass in den Abschnitten React.js und Vue.js keine Hilfestellungen und Richtlinien zum Wechsel zu Angular.js angegeben werden. Die Erklärung dafür erfolgt im nächsten Kapitel, der Technologie-Empfehlung.

7.1 Allgemeines

In diesem Abschnitt werden allgemeine Hilfestellungen und Richtlinien thematisiert, die für alle der Portierungen relevant sind.

7.1.1 Aufbau der Anwendung

Da für die vorliegende Arbeit die Basisanwendungen von Grund auf neu implementiert werden und der Umfang stark begrenzt ist, ist der Aufbau der Anwendung überschaubar. Da in der Realität Anwendung deutlich komplexer sein können, wird an dieser Stelle empfohlen, sich vor der Portierung mit dem Aufbau der Anwendung auseinander zu setzten. Dabei soll jedes Feature der zu portierenden Anwendung analysiert und verstanden werden. Zudem kann dieser Zeitpunkt genutzt werden, um entsprechende Teile der Anwendung zu überdenken.

7.1.2 Technologiewechsel

Nachdem die zu portierende Anwendung analysiert wurde, wird empfohlen die Vorteile der Technologie zu ermitteln in die portiert wird. Beim Aufbau der Basisanwendungen wurde sichtbar, dass jede der ausgewählten Technologien gewisse Vorteile bietet aber teilweise Nachteile hat. Der Technologiewechsel ist ein empfehlenswerter Zeitpunkt, um unsaubere Implementierungen aus dem Code zu überarbeiten und Features der Webtechnologien auszunutzen. Wird beispielsweise eine Anwendung von Angular.js zu Vue.js rein nach Vorlage der bestehenden Software portiert, kommen womöglich Features wie Computed Properties nicht zum Einsatz. Auch der Component Lifecycle sollte bei jeder Portierung in der entsprechenden Webtechnologie betrachtet und geplant werden. Je nach Anwendungsfall ist es möglich durch einen Technologiewechsel die Anwendung zu vereinfachen.

7.1.3 Planung der Portierung

Bevor die Portierung durchgeführt wird, sollte vorab der Ablauf geplant werden. Dabei können die folgenden Punkte als Orientierungshilfe dienen:

- Wahl der Portierungsmethode

  Im ersten Schritt sollte eine Methode für die Portierung festgelegt werden. In Abschnitt 5.1 sind die drei Möglichkeiten Neuentwicklung, Wrapping und Migration angeführt. Bei einer Neuentwicklung wird das komplette System neuentwickelt, wodurch hohe Kosten entstehen können und somit tendenziell eher für kleine Projekte relevant ist. Beim Wrapping werden Teile der Anwendung isoliert und wiederverwendet, alle anderen Inhalte neuentwickelt. Jedoch sollte diese Methode nur als kurzfristige Lösung betrachtet werden, siehe Abschnitt 5.1.2. Die letzte der angeführten Methoden ist die Migration. Hier liegt die Funktionalität im Fokus, während im Hintergrund das System überarbeitet wird. Die Entscheidung welche Methode gewählt wird ist von den Anforderungen und der Größe der Anwendung abhängig, weshalb an dieser Stelle keine tiefergehende Hilfestellung angeführt werden kann.

- Strategie des Systemwechsels wählen

  In Abschnitt 5.2 sind zwei Strategien definiert, die für den Übergang der Systeme gewählt werden können. Eine Möglichkeit ist mittels Cut-and-Run Strategie zu einem definierten Zeitpunkt das alte System zu deaktivieren und zeitgleich das Neue zu aktivieren. Alternativ kann mittels Phased Interoperability Strategie ein inkrementeller Wechsel von alten auf das neue System durchgeführt werden.

- Priorisierung der Features

  Falls bei Portierungen die Phased Interoperability Strategie gewählt wird, sollte vorab eine Priorisierung der Features erstellt werden. Wie in Abschnitt 5.2.2 beschrieben, gibt Phased Interoperability keine Priorisierung der Reihenfolge vor, in welcher die Teile und Features der Anwendung portiert werden sollen. Die Priorisierung ist für jede Anwendung unterschiedlich, somit kann an dieser Stelle kein genaues Vorgehen angegeben werden.

7.1.4 Portierung von Index-Dateien

Alle der ausgewählten Webtechnologien bauen auf einer index.html Datei auf. Bei den Portierungen werden alle Metadaten aus dem `<head>` übernommen. Somit können beispielsweise verwendete externe JavaScript Libraries oder bestehende Stylesheets übernommen werden. Der Aufwand der Portierung der Index-Datei ist gering.

7.1.5 Verwendung von HTTP-Clients

Es wird empfohlen eine externe Library für die Kommunikation mit dem Backend zu verwenden. Dabei sollte überprüft werden, ob diese Library auch mit der zu portierenden Webtechnologie kompatibel ist. Beispielsweise bietet sich die Library axios an, die für diese Arbeit verwendet wurde. Sie kann in React.js und Vue.js verwendet werden. Das hat den Vorteil, dass der Code mit minimalem Aufwand übernommen werden kann, da die Syntax identisch ist und somit nur der Zugriff auf die Variablen geändert werden muss.

Falls die Möglichkeit besteht, ist es empfehlenswert vor der Portierung in dem bestehenden Code auf die neue Library zu wechseln. Der Grund dafür ist, dass sich die Portierung selbst als komplexe Aufgabe herausstellt – wird zudem noch eine neue und unbekannte Library verwendet, ist dies eine potenzielle Fehlerquelle und erhöht die Komplexität.

7.1.6 Portierung des Styles

In diesem Abschnitt sind Hilfestellungen für die Portierung des Styles angeführt. Wie in Abschnitt 4.1.4. angeführt beschäftigt sich die vorliegende Arbeit nicht mit tiefergehenden Stylesheet-Sprachen wie beispielsweise SCSS, SASS oder LESS. Aus diesem Grund gelten die Hilfestellungen in diesem Abschnitt nur für CSS.

Anhand der Erkenntnisse aus den Portierungen wurden zwei Möglichkeiten ermittelt, um den Style zu übernehmen:

1. Zusammenfassung des Style in einer Datei

   Bei diesem Vorgehen wird der gesamte Style der Anwendung in einer Datei zusammengefasst, welche in die neue Anwendung portiert wird. Der Vorteil dabei ist, dass bei dieser Methode kein Aufwand anfällt und bei jeder der ausgewählten Webtechnologien durchführbar ist. Der Nachteil hingegen ist, dass bei vielen Style-Definitionen die CSS-Datei unübersichtlich werden könnte.

2. Aufteilen des Style in die Components

   Eine weitere Möglichkeit ist das Aufteilen des Style in den entsprechenden Components. Jede Style-Definition ist bei diesem Ansatz nur für die Component gültig, in welcher der Style definiert wird. Dadurch entsteht ein modularer Aufbau mit dem Vorteil, dass für mehr Übersicht gesorgt wird und erkenntlich ist für welche Component die Style-Definition gilt, ohne dabei Andere zu beeinflussen. Zusätzlich ist für die Entwicklerin und den Entwickler schneller erkenntlich, wo der Style definiert ist und kann sich somit positiv auf den Entwicklungsaufwand auswirken. Der Nachteil dabei ist, dass im ersten Schritt eine Aufteilung und Zuordnung der Style-Definitionen durchgeführt werden muss. Zudem sollte zuvor überprüft werden, wie die entsprechende Webtechnologie den Style implementiert. Beispielsweise bietet Vue.js dafür mit den Single File Components eine Möglichkeit den Style in der gleichen Datei zu definieren.

Vorbereitend auf eine Portierung kann in der bestehenden Anwendung der Style auf eine oder mehrere CSS-Dateien ausgelagert werden. Zusätzlich soll so wenig wie notwendig inline in den HTML-Elementen definiert werden. Auch das dynamische Setzten von CSS-Klassen, wie in Abschnitt 4.3.8 beschrieben, sollte so wenig wie möglich verwendet werden. Dies ist dem geschuldet, dass dieses Verhalten bei jeder Portierung neu implementiert werden muss.

7.2 Angular.js

Dieser Abschnitt beschäftigt sich mit den Hilfestellungen und Richtlinien für einen Technologiewechsel von Angular.js auf React.js und Vue.js.

7.2.1 Wechsel zu React.js

Angular.js und React.js verwenden unterschiedliche Ansätze beim Data Binding, was bei Portierungen zu Aufwand führen kann, siehe Abschnitt 4.2.11 sowie 4.3.3. Im Rahmen der vorliegenden Arbeit waren die Aufwände jedoch aufgrund der Beschränkung auf Formularinhalte gering. Es wird an dieser Stelle angemerkt, dass wie zuvor beschrieben, Angular.js ein Two-Way Data Binding verwendet, React.js ein One-Way Data Binding. Dies führt dazu, dass untergeordnete Components in React.js grundsätzlich nicht die übergeordneten Components verändern können. Sollte bei dem Wechsel von Angular.js auf React.js solch eine Funktionalität erforderlich sein, wird empfohlen zu überprüfen, ob dieses Verhalten grundsätzlich notwendig ist. Um einen solchen Funktionsumfang zu implementieren, können in den React.js Components eigene Methoden über Props übergeben werden. Diese können dazu genutzt werden, um in der entsprechenden übergeordneten Component eine Änderung durchzuführen. Dabei handelt es sich um sogenannte Inverse Data Flows, die in Abschnitt 4.3.3 beschrieben sind.

Bei der Portierung selbst sollten vorbereitend die Templates aus den externen Dateien in die gleiche Datei wie der JavaScript Controller verschoben werden. Dadurch entsteht im ersten Schritt ein Aufbau gegen die empfohlene Struktur von Angular.js, welcher die Portierung jedoch in den weiteren Schritten erleichtert.

Beim Erstellen der Components in React.js sollte das in 4.3.5 Lifting State Up beschriebene Konzept zum Anheben des States der Components bedacht werden. Dieses Vorgehen ist auch für den Wechsel auf das One-Way Data Binding hilfreich.

Bei der Portierung wird es als vorteilhaft empfunden zuvor mit den Allgemeinen Inhalten und dem Routing auf die entsprechenden Seiten das Grundgerüst der Anwendung aufzubauen. Danach wird empfohlen mit Components zu beginnen, die keine Logik beinhalten und nur Daten darstellen, die über Props übergeben werden. Darauffolgend sollen übergeordneten Components mit statischen Variablen im State implementiert werden. Für den Fall, dass die Logik über mehrere Component-Ebenen übergeben wird, wird nahegelegt den State stets in der niedrigeren Component zu implementieren, danach auf die höhere zu heben und über Props zu übergeben, bis die Logik in der richtigen Component implementiert ist.

Bei den Templates wird empfohlen die Attribute, die in Angular.js mittel `ng-attr` gesetzt werden zu entfernen und die zuvor erwähnten Inhalte aus Abschnitt 6.5.1 Portierung zu React.js zu beachten.

Wie in Abschnitt 4.2.10 beschrieben verwendet Angular.js Dependency Injection um Services zu übergeben. Da dies in React.js nicht möglich ist, wurden zwei Varianten aus den Portierungen abgeleitet, um die Kommunikation mit dem Backend zu portieren:

- Implementierung von Dependency Injection in React.js

  Bei dieser Möglichkeit wird Dependency Injection für Services in React.js implementiert. Hierfür wird mittels der `require()` Methode von JavaScript die bestehende Angular.js Datei geladen, in welcher der Service definiert wird und diese in einer Variable festgehalten. Danach können die Dependencies im Konstruktor der React.js Component aufgerufen werden. Je nach Implementierung der Services ist es erforderlich, dass Angular.js in der Anwendung geladen wird, beispielsweise über externe Scripts im `<head>` der index.html. Diese Methode eignet sich für Wrapping, da die bestehenden Services unverändert bleiben und von React.js aufgerufen werden. Jedoch wird diese Methode nicht als langfristige Lösung empfohlen, da die Angular.js Dependencies übergeben und überprüft werden müssen.

- Portierung mittels Library

  Diese Variante wird für die Portierungen dieser Arbeit verwendet. Hier wird wie in Abschnitt 7.1.5 beschrieben, empfohlen bereits im Code von Angular.js auf eine externe Library zu wechseln damit die Komplexität bei den Portierungen selbst geringer wird.

7.2.2 Wechsel zu Vue.js

Bei einem Technologiewechsel von Angular.js auf Vue.js wird, wie im vorherigen Abschnitt nahegelegt, zuerst die allgemeine Struktur sowie die Views aufzubauen und das Routing zu implementieren. Weiters empfiehlt es sich die Templates von Angular.js vorab in einer Datei mit dem Controller zusammenzufassen.

Nachdem die allgemeine Struktur und die Views implementiert wurden, wird empfohlen die einzelnen Components von Angular.js in Vue.js zu portieren. Dafür wurden aus den Portierungen die folgenden Schritte als Richtlinien abgeleitet:

1. Portierung des Templates

   Der Aufwand für die Portierung des Templates ist gering im Vergleich zur Portierung zu React.js. Es kann das Template von Angular.js kopiert werden. Alle Angular.js Directives mit `ng-` können auf die entsprechenden von Vue.js mit `v-` geändert werden.

2. Portierung des Controllers

   Im zweiten Schritt wird die Logik des Controllers von Angular.js in die entsprechende Single File Component übernommen. Es wird empfohlen mit den Variablen und Daten des Angular.js Controllers zu beginnen und diese in den Daten der Vue.js Components zu definieren. Danach wird die entsprechende Logik aus dem Angular.js Controller in Methoden aufgeteilt und in den `methods` von Vue.js implementiert.

3. Portierung des Services

   Der letzte Schritt ist die Portierung des Services. Wie im vorherigen Abschnitt ist auch eine manuelle Implementierung von Dependency Injection in Vue.js möglich, da diese auf reinen JavaScript-Funktionen basiert. Es wird an dieser Stelle empfohlen vorab in der bestehenden Anwendung auf eine Library zu wechseln, die für Angular.js als auch für Vue.js verfügbar ist und erst danach den Code zu portieren. Bei den Services selbst ist es von Vorteil diese in den `methods` von Vue.js zu implementieren, sofern diese nur für die Component nutzbar sein soll.

7.3 React.js

Dieser Abschnitt behandelt den Technologiewechsel von React.js zu Angular.js sowie Vue.js und beinhaltet Hilfestellungen und Richtlinien für die Portierungen.

7.3.1 Wechsel zu Angular.js

Wie zuvor angemerkt werden keine Hilfestellungen und Richtlinien für die Portierung von React.js zu Angular.js erarbeitet. Die Erkenntnisse der durchgeführten Portierung von React.js zu Angular.js wurden in Abschnitt 6.6.1 definiert.

7.3.2 Wechsel zu Vue.js

Wie bei den bisherigen Hilfestellungen wird auch bei dieser Portierung empfohlen vorab die Grundstruktur der Anwendung und das Routing aufzubauen, bevor die einzelnen Components portiert werden.

Beim Wechsel von React.js zu Vue.js wird angemerkt, dass die `render()` Methode aus den React.js Components zu Vue.js portierbar ist, jedoch ist dies nicht empfehlenswert. Es fällt bei der Methode weniger Aufwand an, dabei können aber nicht die Vorteile von Templates in Vue.js genutzt werden. Allgemein wird die Verwendung von Single File Components empfohlen, da diese als übersichtlich empfunden werden und dem Aufbau von React.js ähnlich sind.

Bei dem Wechsel von React.js zu Vue.js wird folgendes Vorgehen empfohlen:

1. Portierung des Templates

   Je nach Umfang und Funktionalität des Templates fällt in diesem Schritt der größte Aufwand an. Dabei werden die in JSX geschriebenen Elemente in Vue.js Templates portiert und entsprechende Zuweisung der Attribute geändert.

2. Portierung des States in React.js mit statischen Daten

   Im nächsten Schritt wird der State von React.js in den Daten der Vue.js Component implementiert. Alle Methoden, die den State in React.js manipulieren, werden in den Methoden der Vue.js Component übernommen. Dabei ändert sich der Zugriff auf den State mittels `this.variable` anstelle von `this.state.variable`.

3. Portierung der API-Aufrufe in den Vue.js-Lifecycle

   Nachdem die statischen Daten implementiert sind, werden die Methoden zur Kommunikation mit dem Backend implementiert, um diese abschließend durch die Daten aus den Backend zu ersetzen. Aufgrund der Ähnlichkeit der Lifecycle von React.js und Vue.js können die Methoden mit geringem Aufwand im selben Lifecycle verwendet werden.

In Listing 34 wird die Definition der Daten, die Kommunikation mit dem Backend und der Lifecycle von React.js und Vue.js gegenübergestellt. Beide beinhalten die gleiche Funktionalität. Dabei wird angemerkt, dass die Components nicht vollständig dargestellt sind, um das Listing übersichtlich zu halten. Der Code ist den Basisanwendungen entnommen.

Ein Unterschied im Listing ist bereits bei der Namensgebung der Components. In React.js wird dafür die Klasse entsprechend benannt, in Vue.js wird der Name der Component über eine Eigenschaft gesetzt. Ein weiterer Unterschied ist die Definition der Daten. Hier ist im Listing zu sehen, dass in React.js im Konstruktor der State mittels `this.state` als Objekt gesetzt wird. In Vue.js hingegen wird die `data()` Methode definiert, die ein Objekt der Daten der Component zurückgibt. Beide Components haben eine `fetchData()` Methode, die Daten mittels der Library axios Daten aus dem Backend laden. Diese Methode unterscheidet sich nur darin, dass in React.js die Methode `this.setState()` mit den Daten aufgerufen wird, während in Vue.js direkt mittels `this` auf die Daten geschrieben wird. Die `fetchData()` Methoden werden bei beiden Components aufgerufen, sobald der Component gemountet wird. In React.js wird hierfür die Methode `componentDidMount()` verwendet, in Vue.js hingegen die `mounted()` Methode der Component.

```
<!-- React.js -->
class Checkbox extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: ""
    };
  }

  fetchData() {
    axios.get(API.URL_CHECKBOX).then((response) => {
      this.setState({
        name: response.data.name,
      });
    });
  }

  componentDidMount() {
    this.fetchData();
  }
}

<!-- Vue.js -->
export default {
  name: "Checkbox",
  data() {
    return {
      name: ""
    };
  },
  methods: {
    fetchData() {
      this.$http.get(API.URL_CHECKBOX).then((response) => {
        this.name = response.data.name;
      });
    }
  },
  mounted() {
    this.fetchData();
  }
};
```

Listing 34: Gegenüberstellung von Daten und Methoden von React.js und Vue.js

7.4 Vue.js

Dieser Abschnitt behandelt die Hilfestellungen für die Portierung von Vue.js.

7.4.1 Wechsel zu Angular.js

Wie bei der Portierung von React.js zu Angular.js werden an dieser Stelle keine Hilfestellungen und Richtlinien angeführt. Es wird auf die Inhalte der Portierung von Vue.js zu Angular.js in Abschnitt 6.7.1 verwiesen.

7.4.2 Wechsel zu React.js

Bei dem Wechsel von Vue.js auf React.js wird dieselbe Vorgehensweise empfohlen, wie bei den Hilfestellungen von React.js zu Vue.js, siehe Abschnitt 7.3.2. Beginnend mit dem Aufbau der Struktur und dem Routing zwischen den Seiten, wird empfohlen vorerst die Templates der Components zu portieren. Im zweiten Schritt werden die Daten von Vue.js in einen State von React.js umgewandelt. Abschließend wird empfohlen die Kommunikation mit dem Backend zu portieren.

Die in der Beschreibung von Listing 34 angeführten Hilfestellungen dienen ebenso für den Wechsel von Vue.js auf React.js. Alle Erkenntnisse der Gegenüberstellung sind für beide Portierungen gültig.

8 TECHNOLOGIE-EMPFEHLUNG

Dieses Kapitel setzt sich mit der Empfehlung der ausgewählten Webtechnologien auseinander. Dabei wird zuvor das Vorgehen der Bewertung festgehalten und danach die drei ausgewählten Webtechnologien bewertet.

8.1 Bewertungsmethode und Kriterien

Die Bewertung der Webtechnologien erfolgt mittels mehrerer Kriterien, im Folgenden aufgelistet. Dabei wird angemerkt, dass nur Kriterien beurteilt werden, die in Rahmen dieser Arbeit erarbeitet wurden.

- Popularität

  Es wird basierend auf den in Kapitel 2 angeführten Umfragen und Statistiken die Popularität von Angular.js, React.js und Vue.js bewertet.

- Entwicklungsaufwand

  Der Aufwand der Erstellung der Basisanwendungen wird dazu verwendet, um Erkenntnisse über die Unterschiede der Entwicklungsaufwände zu gewinnen. Zudem wird eingeschätzt, wie aufwendig es ist die Webtechnologie zu erlernen.

- Flexibilität

  Anhand der Erfahrungen während der Entwicklung der Basisanwendungen wird die Flexibilität der Webtechnologien bewertet. Unter Flexibilität wird der Freiheitsgrad der Entwicklung verstanden. Beispielsweise kann ein Framework durch Vorgaben einschränkend wirken und somit die Entwicklung aufwändiger gestalten.

- Portierbarkeit

  Zusätzlich werden aus den Daten der Portierungen Rückschlüsse gezogen, ob eine der ausgewählten Webtechnologien Vorteile gegenüber den anderen hat oder, ob eine bestimmte Technologie besonders geeignet für einen Technologiewechsel scheint. Dafür werden die Ergebnisse der Portierung aus der Webtechnologie zu den anderen beiden addiert und gemittelt, sowie der Mittelwert der Portierungen zur Webtechnologie hin als Richtwert genommen, wie in Abschnitt 6.3.2 beschrieben wird.

- Zukunftssicherheit

  Der letzte Punkt, der beurteilt wird, ist die Zukunftssicherheit der ausgewählten Technologien. Dafür werden primär die Daten und Erkenntnisse aus den Portierungen und Statistiken verwendet.

8.2 Bewertung

In diesem Abschnitt werden die drei ausgewählten Technologien nach den zuvor beschriebenen Kriterien bewertet.

8.2.1 Angular.js

Wie Statistiken und Umfragen zeigen, ist die Beliebtheit von Angular und Angular.js sinkend. In den Statistiken sind die beiden Technologien zusammengefasst, da Angular.js jedoch nach Darwin (2020) nur noch bis Ende 2021 Security Support erhält, könnte die Popularität weiter sinken. Zudem ist Angular.js durch das Supportende nicht mehr zukunftssicher.

Der Entwicklungsaufwand der Basisanwendung von Angular.js ist deutlich höher als bei den anderen beiden Webtechnologien. Dies ist auf den zusätzlichen Aufwand der Modules zurückzuführen, durch welche zusätzlicher Code notwendig ist. Zudem führt die umfangreiche Struktur zu Entwicklungsaufwand in Angular.js

Das Framework schränkt durch seine Vorgaben, wie beispielsweise den Aufbau von Modules, die Flexibilität ein. Dies führt auch zu einer steilen Lernkurve. Während der Entwicklung der Basisanwendungen traten bei Angular.js die meisten Fehler auf.

Die Mittelwerte der Portierungen sind 11 von 30 Punkten für die Portierungen von Angular.js weg und 11,5 für die Portierungen zu Angular.js. Die Portierung zu React.js hat 9 von 30 Punkten, die zu Vue.js 13 Punkte. Die Portierung von React.js zu Angular.js wurde mit 8 von 30 Punkten bewertet, der niedrigste Wert aller durchgeführten Portierungen. Der Wechsel von Vue.js zu Angular.js wurde hingegen mit 15 Punkten bewertet.

8.2.2 React.js

Sowohl die Stack Overflow Trends als auch die Google Trends zeigen, dass React.js derzeit die beliebteste Webtechnologie ist. Dies wird durch die Umfragen von Stack Overflow bestätigt, auch hier ist React.js die beliebteste Technologie im Webbereich.

Der Entwicklungsaufwand der Basisanwendung war deutlich geringer als bei Angular.js. Da React.js wenig Code und Dateien benötigt, ist die Library einsteigerfreundlich und schneller erlernbar als Angular.js.

Da React.js selbst nur eine Library ist und keine Vorgaben für beispielsweise einen HTTP-Client gibt, wirkt sich dies positiv auf die Flexibilität aus. Als nicht optimal erweist sich hier allerdings die Tatsache, dass unerfahrene Entwicklerinnen und Entwickler möglicherweise unsauberen Code schreiben könnten, da keine klaren Vorgaben gegeben sind wie eine Anwendung aufzubauen und zu strukturieren ist.

Die Bewertung der Portierungen von React.js weg zu den anderen Technologien wurden mit einem Mittelwert von 11,5 Punkten bewertet. Dies ist auf den umfangreicheren Aufbau von Angular.js und Vue.js zurückzuführen. Besser sind die Portierungen von Angular.js und Vue.js zu React.js bewertet, hier wurden 12,5 von 30 Punkten vergeben.

Betrachtet man die Statistiken, so scheint die Popularität stets zu steigen. So wird React.js auch in den nächsten Jahren weit verbreitet sein. Mit Facebook steht zudem noch ein großes Unternehmen hinter der Webtechnologie, weshalb React.js auch in den nächsten Jahren Support erhält und weiterentwickelt wird.

8.2.3 Vue.js

Vue.js hat laut den Stack Overflow Trends und den Google Trends seit 2016 bis 2019 kontinuierlich an Popularität gewonnen. Den Umfragen von Stack Overflow nach ist Vue.js die zweit beliebteste Webtechnologie.

Der Entwicklungsaufwand der Basisanwendung ist minimal höher als bei einer React.js Anwendung. Als Vorteil wird bei Vue.js die Syntax in den Templates gesehen, als auch der strukturierte Aufbau.

Wie React.js hat Vue.js keine Einschränkungen, was einen positiven Effekt auf die Flexibilität während der Entwicklung hat. Dadurch entsteht jedoch wie bei React.js der Nachteil, dass unerfahrene Entwicklerinnen und Entwickler zu unsauberem Code tendieren können.

Bei den Portierungen hat Vue.js am besten von allen Webtechnologien abgeschnitten. Der Mittelwert der Portierungen von Vue.js zu den anderen Webtechnologien liegt bei 15,5 von 30 Punkten. Aber auch die Portierungen von den anderen Technologien zu Vue.js sind mit 14 Punkten besser als alle anderen Portierungen bewertet. Grund dafür sind einerseits die starken Ähnlichkeiten der Template-Syntax zu Angular.js und andererseits, dass die Logik und Daten einer Component ohne viel Aufwand von Vue.js zu React.js portierbar sind.

In den angeführten Statistiken hat sich bisher im Jahr 2020 die Beliebtheit von Vue.js nicht mehr so stark wie die Vorjahre erhöht. Es wird aufgrund der hohen Beliebtheit unter Entwicklerinnen und Entwicklern dennoch davon ausgegangen, dass Vue.js in den nächsten Jahren kontinuierlich weiterentwickelt wird und die Webtechnologie vermehrt in der Praxis Anwendung findet.

8.3 Empfehlung

Bevor eine Empfehlung erläutert wird, soll angemerkt werden, dass die folgenden Erkenntnisse teilweise auf subjektiven Einschätzungen beruhen. Sofern dies der Fall ist, wird dies entsprechend angemerkt.

Angular.js wird von den drei ausgewählten Webtechnologien am wenigsten empfohlen, in diesem Fall sogar gar nicht. Dies ist dem Faktum geschuldet, dass wie zuvor angemerkt ab 31. Dezember 2021 der Long Term Support (LTS) endet und somit keine Security-Updates mehr entwickelt werden. Aus diesem Grund wurden auch keine Hilfestellungen für die Portierungen zu Angular.js erarbeitet. Der Entwicklungsaufwand von Angular.js ist deutlich höher als bei den anderen Webtechnologien. Betrachtet man die anderen zuvor ermittelten Kriterien, so schneidet Angular.js hinsichtlich Flexibilität und Portierbarkeit am schlechtesten ab.

An zweiter Stelle in dieser Bewertung liegt React.js, was jedoch sicher durch eine subjektive Wahrnehmung beeinflusst wird. So profitiert React.js zwar von höherer Popularität und einem minimal geringerem Entwicklungsaufwand, jedoch wird der Aufbau und die Struktur, sowie die Template-Syntax von Vue.js bevorzugt. Beide Webtechnologien haben hinsichtlich Flexibilität keine Einschränkungen.

Abschließend zeigen die Ergebnisse der Portierungen, dass einerseits die Portierungen zu Vue.js, sowie andererseits die Portierungen von Vue.js weg die besten Ergebnisse erzielt haben. Somit ist hinsichtlich Zukunftssicherheit Vue.js im Vorteil, da es leichter ist zu dieser Webtechnologie von anderen zu wechseln.

9 ZUSAMMENFASSUNG UND AUSBLICK

In der vorliegenden Arbeit wurde analysiert, welche der Webtechnologien Angular.js, React.js und Vue.js Zukunftssicherheit hinsichtlich Portierbarkeit und Wiederverwendbarkeit bieten. Die Portierbarkeit wurde in dieser Arbeit in einem begrenzten Rahmen überprüft. Es wurden einheitliche Anwendungen mit den ausgewählten Webtechnologien entwickelt, die sich auf Formularinhalte fokussieren. Jede dieser Anwendungen wurde in die anderen Webtechnologien portiert. Um eine objektive Bewertung zu gewährleisten, wurde ein einheitliches Vorgehen bei den Portierungen, sowie ein Bewertungsschema definiert. Nachdem Hilfestellungen und Richtlinien erarbeitet wurden, konnte abschließend eine Bewertung der Webtechnologien durchgeführt werden.

Dabei hat sich herausgestellt, dass Angular.js aufgrund der Einstellung des Supports nicht zu empfehlen ist. React.js und Vue.js hingegen bieten aktuell Zukunftssicherheit, sind weit verbreitet und profitieren von einer hohen Beliebtheit. Hinsichtlich der Portierbarkeit wird in dieser Arbeit Vue.js ein Vorteil gegenüber React.js zugeschrieben.

Aufgrund der Begrenzung der Portierungen auf Formularinhalte sind im Folgenden noch weitere Aspekte aufgelistet, die bei einer weiterführenden Forschung betrachtet werden könnten.

- Ein Vergleich mit Angular

  Für diese Arbeit wurde bewusst Angular.js gewählt, da somit allen Webtechnologien JavaScript als Basistechnologie zugrunde liegt. Angular verwendet jedoch TypeScript und hat einen anderen Aufbau. Somit bietet sich Angular für einen weiteren Vergleich mit React.js und Vue.js an.

- Die Implementierung und der Vergleich von Security-Maßnahmen

  In dieser Arbeit wurden keine Security-Maßnahmen analysiert und bewertet. Weiterführende Forschung könnte die verschiedenen Implementierungen der Webtechnologien vergleichen.

- Der Vergleich von Performance

  Die Evaluierung von Geschwindigkeiten wurde in dieser Arbeit nicht betrachtet. Hier könnte eine Gegenüberstellung von verschiedenen Ladegeschwindigkeiten den Vergleich der Frameworks aus einer anderen Perspektive betrachten.

ANHANG A - Layout der Basisanwendung

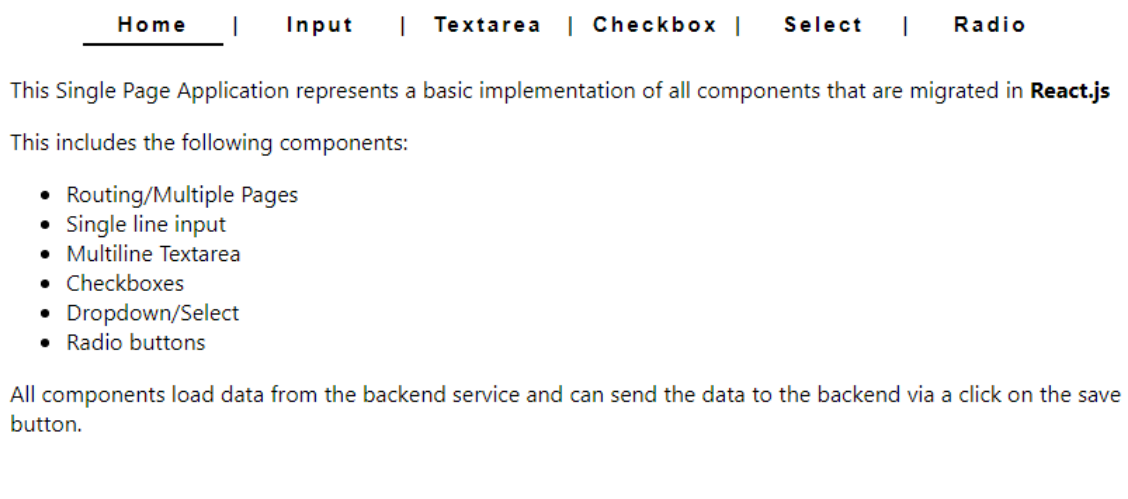

Abbildung 17: Startseite der Basisanwendung in React.js

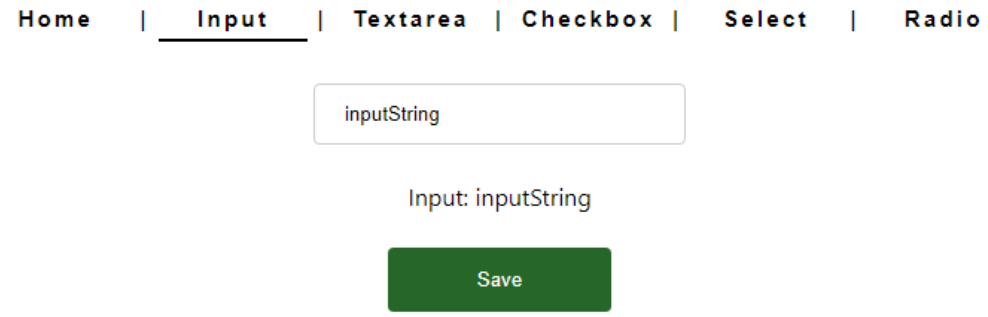

Abbildung 18: Seite der Basisanwendung mit Input-Component

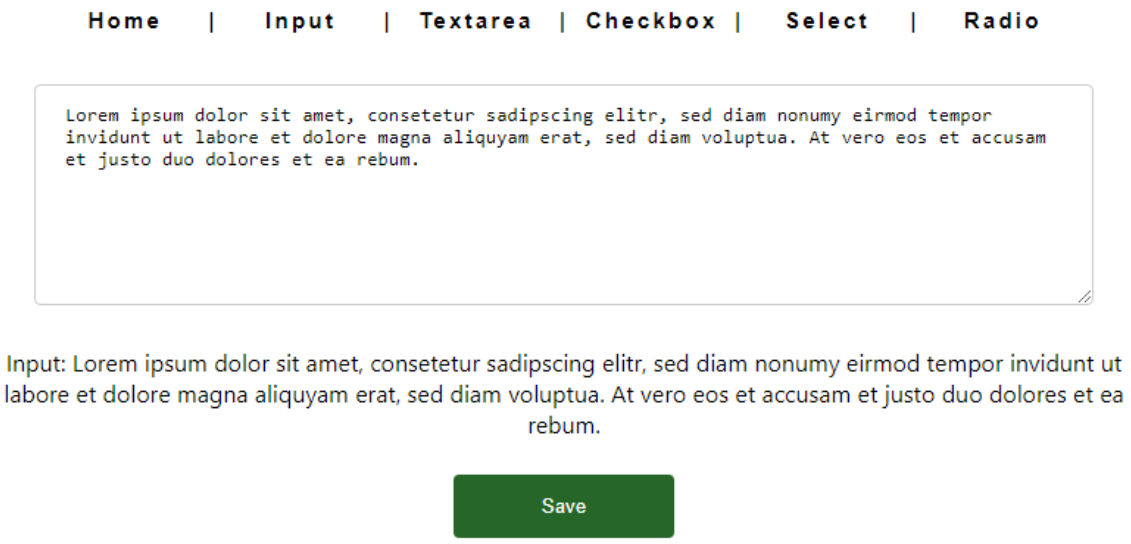

Abbildung 19: Seite der Basisanwendung mit Textarea-Component

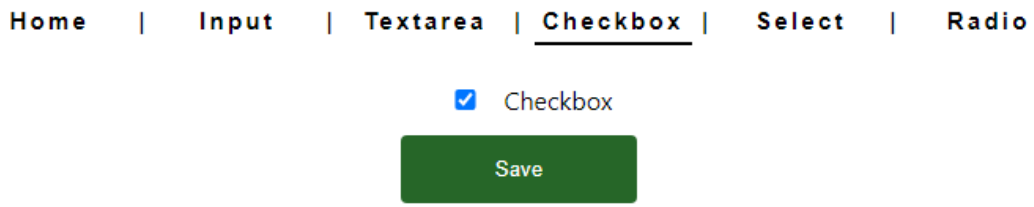

Abbildung 20: Seite der Basisanwendung mit Checkbox-Component

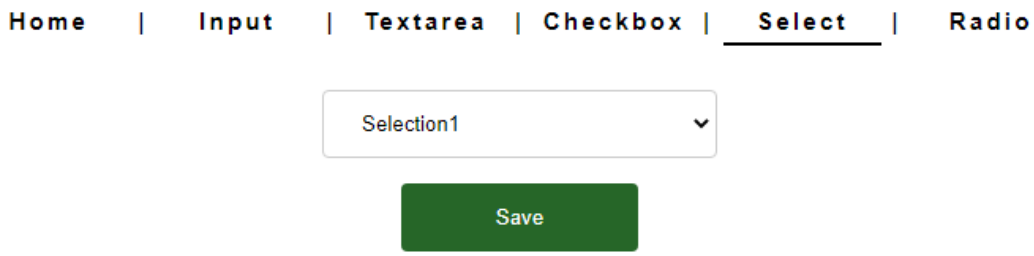

Abbildung 21: Seite der Basisanwendung mit Select-Component

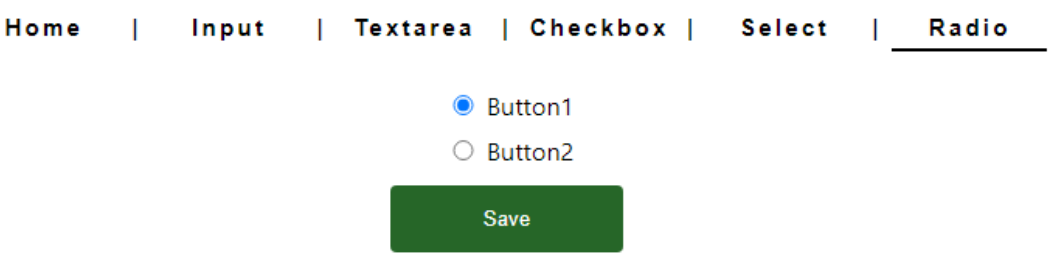

Abbildung 22: Seite der Basisanwendung mit Radio-Component

ABKÜRZUNGSVERZEICHNIS

API – Application Programmer Interface
CSS – Cascading Style Sheet
DI – Dependency Injection
DOM – Document Object Model
ES – ECMAScript
HTML – Hypertext Markup Language
JSON – JavaScript Object Notation
LTS – Long Term Support
MVC – Model View Controller
MVVM – Model View ViewModel
SPA – Singe Page Application
UI – User Interface
URL – Uniform Resource Locator

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LISTINGS

LITERATURVERZEICHNIS

A M, V. & Sonpatki, P. (2016). *ReactJS by Example - Building Modern Web Applications: Get up and running with ReactJS by developing five cutting-edge and responsive projects*. Packt Publishing.

Angular. (o. D.). *Angular.io. Abgerufen am 17.06.2020*. https://angular.io

Angular.js. (o. D.). *docs.angularjs.org. Abgerufen am 17.06.2020*. https://docs.angularjs.org

Banks, A. & Porcello, E. (2017). *Learning React: Functional Web Development With React And Redux* (First Edition). O'Reilly Media, Inc.

Brown, E. (2016). *Learning JavaScript: JavaScript Essentials For Modern Application Development* (3rd Edition). O'Reilly Media, Inc.

Chandermani. (2015). *AngularJS by Example: Learn AngularJS by creating your own apps, using practical examples which you can use and adapt*. Packt Publishing.

Chau, G. (2017). *Vue.js 2 Web Development Projects: Learn Vue.js by building 6 web apps*. Packt Publishing.

Darwin, P. B. (2020). *Stable AngularJS and Long Term Support. Abgerufen am 25.08.2020*. https://blog.angular.io/stable-angularjs-and-long-term-support-7e077635ee9c

Dayley, B. (2014). *Learning AngularJS*. Addison-Wesley Professional.

Demeyer, S., Ducasse, S. & Nierstrasz, O. (2009). *Object-Oriented Reengineering Patterns* (9/28/09 edition). Square Bracket Associates.

Djirdeh, H., Murray, N. & Lerner, A. (2018). *Fullstack Vue: The Complete Guide to Vue.js and Friends* (1. Aufl.). CreateSpace Independent Publishing Platform.

Duckett, J. (2010). *Beginning HTML, XHTML, CSS, and JavaScript* (1. Aufl.). Wrox.

Elliot, E. (2014). *Programming JavaScript Applications: Robust Web Architecture With Node, HTML5, And Modern JS Libraries* (First Edition). O'Reilly Media, Inc.

Ember. (o. D.). *Emberjs.com. Abgerufen am 17.06.2020*. https://emberjs.com

Fedosejev, A. (2015). *React.js Essentials: A fast-paced guide to designing and building scalable and maintainable web apps with React.js*. Packt Publishing.

Fink, G. & Flatow, I. (2014). *Pro single page application development: Using Backbone.js and ASP.NET*. Apress/Springer.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. M. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software* (1st ed.). Prentice Hall.

Gharat, A. & Nehlsen Matthias. (2014). *AngularJS UI Development: Design, build, and test production-ready applications in AngularJS*. Packt Publishing.

Google. (o. D.). *trends.google.com. Abgerufen am 29.06.2020*. https://trends.google.com

Green, B. & Seshadri, S. (2013). *AngularJS: Less Code, More Fun, and Enhanced Productivity with Structured Web Apps* (Third release). O'Reilly Media, Inc.

Imsirovic, A. (2018). *Vue.js Quick Start Guide: Learn how to build amazing and complex reactive web applications easily using Vue.js*. Packt Publishing.

Jackson, W. (2016). *JSON quick syntax reference*. Apress.

Jha, M. & Maheshwari, P. (2005). Reusing Code for Modernization of Legacy Systems. *Proceedings - 13th IEEE International Workshop on Software Technology and Engineering Practice, STEP 2005*, *2005.* https://doi.org/10.1109/STEP.2005.21

jQuery. (o. D.). *jQuery.com. Abgerufen am 17.06.2020*. https://jquery.com

Keith, J. & Sambells, J. (2011). *DOM Scripting*. Springer Fachmedien.

Kouraklis, J. (2016). MVVM as Design Pattern. In https://doi.org/10.1007/978-1-4842-2214-0_1

Kyrakidis, A. & Maniatis, K. (2016). *The Majesty of Vue.js*. Packt Publishing.

Macrae, C. (2018). *Vue.js: Up and Running: Building Accessible and Performant Web Apps* (First Edition). O'Reilly Media, Inc.

Maniraho, P. (2018). *The Deep Dive - Migration From Angular to React: A primer on converting AngularJS applications to ReactJS. Abgerufen am 22.08.2020*. https://medium.com/hoopeez/the-deep-dive-migration-from-angular-to-react-ea5a807e95eb

Marini, J. (2002). *Document Object Model*. McGraw-Hill Education.

Marrs, T. (2017). *JSON at work: Practical data integration for the web*. O'Reilly Media.

Masak, D. (2006). *Legacysoftware: Das lange Leben der Altsysteme*. Springer-Verlag Berlin Heidelberg.

Mazinanian, D. & Tsantalis, N. (2016, März). An Empirical Study on the Use of CSS Preprocessors. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* (S. 168–178). IEEE. https://doi.org/10.1109/SANER.2016.18

Osmani, A. (2012). *Learning JavaScript Design Patterns* (First Edition). O'Reilly Media, Inc.

Panda, S. (2014). *AngularJS: Novice To Ninja*. SitePoint.

Passaglia, A. (2017). *Vue.js 2 Cookbook: Build modern, interactive web applications with Vue.js*. Packt Publishing.

React. (o. D.). *Reactjs.org. Abgerufen am 17.06.2020*. https://reactjs.org

Richardson, L. (2007). *Web-Services mit REST* (1. Aufl.). *Safari Books Online*. O'Reilly Verlag.

Robbestad, S. A. (2016). *ReactJS Blueprints: Create powerful applications with ReactJS, the most popular platform for web developers today*. Packt Publishing.

Sidelnikov, G. (2017). *React.js Book: Learning React JavaScript Library From Scratch*. Independently Published.

Stack Overflow. (o. D.). *stackoverflow.com. Abgerufen am 01.06.2020*. https://stackoverflow.com

Stack Overflow. (2019). *Developer Survey Results 2019. Abgerufen am 09.04.2020*. https://insights.stackoverflow.com/survey/2019#most-loved-dreaded-and-wanted

Stack Overflow. (2020). *Stack Overflow Trends. Abgerufen am 17.06.2020*. https://insights.stackoverflow.com/trends?tags=angular%2Creactjs%2Cjquery%2Cvue.js%2Cember.js

Street, M., Passaglia, A. & Halliday, P. (2018). *Complete Vue.js 2 Web Development: Practical guide to building end-to-end web development solutions with Vue.js 2* (1. Aufl.). Packt Publishing.

Syromiatnikov, A. & Weyns, D. (2014). A Journey through the Land of Model-View-Design Patterns. In *2014 IEEE/IFIP Conference on Software Architecture*.

Tarasiewicz, P. & Böhm, R. (2014). *AngularJS: Eine praktische Einführung in das JavaScript-Framework*. dpunkt.verlag GmbH.

Tripathy, P. & Naik, K. (2015). *Software evolution and maintenance: A practitioner's approach*. John Wiley & Sons.

Uzayr, S. b., Cloud, N. & Ambler, T. (2019). *JavaScript Frameworks for Modern Web Development: The Essential Frameworks, Libraries, and Tools to Learn Right Now* (Second Edition). Apress.

van Deursen, S. & Seemann, M. (2019). *Dependency Injection Principles, Practices, and Patterns* (1st edition). Manning Publications.

Vue.js. (o. D.). *Vuejs.org. Abgerufen am 17.06.2020*. https://vuejs.org

Wagner, C. (2014). *Model-Driven Software Migration: A Methodology: Reengineering, Recovery and Modernization of Legacy Systems*. Springer Vieweg.

Wagner, G. & Diaconescu, M. (2017). Web Applications with Javascript or Java: Volume 1: Constraint Validation, Enumerations, Special Datatypes. In *De Gruyter Textbook. EBOOK PACKAGE Engineering, Computer Sciences 2017.* De Gruyter Oldenbourg. https://doi.org/10.1515/9783110499957

Wieruch, R. (2018). *The Road To Learn React: Your journey to master plain yet pragmatic React.js*. CreateSpace Independent Publishing Platform.

Wilken, J. (2018). *Angular In Action* (1st). Manning Publications.

Williamson, K. (2015). *Learning AngularJS: A Guide To AngularJS Development* (First Edition). O'Reilly Media, Inc.