Fakultät Elektrotechnik, Medien und Informatik

Medienproduktion und -technik

Bachelorarbeit

von

Anke B r e n n e r

Refactoring von Code mithilfe moderner Webtechnologien – Best Practices umgesetzt mit der JavaScript Bibliothek React

Refactoring Code with Modern Web Technologies – Best Practices Implemented with the JavaScript Library React

Ostbayerische Technische Hochschule Amberg-Weiden

Fakultät Elektrotechnik, Medien und Informatik

Medientechnik und Medienproduktion

Bachelorarbeit

von

Anke B r e n n e r

Refactoring von Code mithilfe moderner Webtechnologien – Best Practices umgesetzt mit der JavaScript Bibliothek React

Refactoring Code with Modern Web Technologies – Best Practices Implemented with the JavaScript Library React

Erstbetreuer:	Prof. Dr. Dieter Meiller
Zweitbetreuer:	M. Eng. Veit Stephan
Bearbeitungszeitraum:	11.12.2020 bis 07.06.2021

Ostbayerische Technische Hochschule Amberg-Weiden

Fakultät Elektrotechnik, Medien und Informatik

Bestätigung gemäß § 12 APO

Name und Vorname der Studentin/des Studenten: **Brenner, Anke**

Studiengang: **Medienproduktion und Medientechnik**

Ich bestätige, dass ich die Bachelorarbeit mit dem Titel:

Refactoring von Code mithilfe moderner Webtechnologien – Best Practices umgesetzt mit der JavaScript Bibliothek React

selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Datum:

Unterschrift:

Ostbayerische Technische Hochschule Amberg-Weiden

Fakultät Elektrotechnik, Medien und Informatik

Bachelorarbeit Zusammenfassung

Studentin/Student (Name, Vorname): **Brenner, Anke**

Studiengang: **Medienproduktion und Medientechnik**

Aufgabensteller, Professor: **Prof. Dr. Dieter Meiller**

Durchgeführt in (Firma/Behörde/Hochschule): **Evidentmedia**

Betreuer in Firma/Behörde: **Bernd Gerlang-Gradl**

Ausgabedatum: 11.12.20 Abgabedatum: 07.06.21

Titel:

Refactoring von Code mithilfe moderner Webtechnologien – Best Practices umgesetzt mit der JavaScript Bibliothek React

Zusammenfassung:

Softwareentwicklung ist ein Prozess, bei dem das Design stetig an neu entstehende Anforderungen angepasst werden sollte. Zu diesem Zweck wird die weitverbreitete Technik des Refactorings eingesetzt, die das Ziel verfolgt, die Struktur der Software zu verbessern. Moderne Webtechnologien, wie beispielsweise Web-Frameworks und Web-Bibliotheken bieten zusätzlich neue Möglichkeiten für die Organisation des Quellcodes und werden immer beliebter. Die Firma Evidentmedia hat das Monte-Tool, eine Webanwendung entwickelt. Beim Veranstaltungseditor dieser Anwendung, wird im Zuge des Praxisteils ein Refactoring in Kombination mit React, einer JavaScript Bibliothek durchgeführt. Das Ziel dieser Arbeit ist es, herauszustellen, ob ein Refactoring mithilfe moderner Webtechnologien geeignet ist, um die Wartbarkeit einer Webanwendung zu verbessern.

Schlüsselwörter:

- Refactoring
- Webtechnologie
- Best Practices
- JavaScript
- React

Inhaltsverzeichnis

Abbildungsverzeichnis vii
Abkürzungsverzeichnis viii
Glossar ix
1 Problemstellung: Verbesserung der Wartbarkeit von Code 1
2 Wartbarkeit 3
3 Wartungstechniken 4
3.1 Refactoring 5
3.1.1 Gründe für ein Refactoring 6
3.1.2 Vorgehensweise bei einem Refactoring 7
3.1.3 Geeignete Zeitpunkte für Refactoring 7
3.1.4 Geeigneter Zeitpunkt für einen Rewrite 8
3.2 Integration moderner Webtechnologien 8
3.2.1 Vorteile 9
3.2.2 Nachteile 9
3.3 Passende Wartungstechnik für das Praxisprojekt 10
4 React: eine moderne Webtechnologie auf dem Vormarsch 10
4.1 Leistungsspektrum von React 11
4.1.1 Components 11
4.1.2 Daten der Components 12
4.1.2.1 Props 12
4.1.2.2 State 13
4.1.3 Virtueller DOM 13
4.1.4 Style 14
4.1.5 JSX 14
4.2 Redux: eine Bibliothek für externes State-Management 15
4.3 Steigende Beliebtheit gegenüber anderen Frameworks 17
4.3.1 Angular 18
4.3.2 Vue 19
5 Unit Tests 19
5.1 Eigenschaften guter Tests 20
5.2 Tests in React 21
5.2.1 Jest 21
5.2.2 React Testing Library 23
6 Praktische Anwendung: Veranstaltungseditor des Monte-Tools 23

6.1 Funktionalität des Veranstaltungseditors ... 23
6.1.1 Anlegen einer Veranstaltung ... 25
6.1.2 Editieren einer Veranstaltung ... 26
6.1.3 Buchungsprozess ... 26
6.1.4 Änderungsanfragen ... 27
6.2 Verwendete Tools ... 28
6.2.1 CakePHP ... 28
6.2.2 MySQL, JQuery und Bootstrap ... 29
6.2.3 Integration von React ... 30
6.3 Probleme der bisherigen Softwarestruktur ... 31
6.4 Erarbeitungsprozess der Lösung ... 33
6.4.1 Grafische Visualisierung des Codes ... 33
6.4.2 Übersichtliche Gliederung durch Components ... 35
6.4.3 Grundlegende Entscheidungen ... 36
6.4.3.1 Ausschließlich „Functional Components“ oder mitunter „Class Components“ ... 36
6.4.3.2 Verwendung von Redux ... 38
6.4.4 Code-Implementierung ... 39
6.4.4.1 State-Organisation mit Redux ... 39
6.4.4.2 Handhabung der Benutzereingaben ... 41
6.4.4.3 Buchungsprozess ... 44
6.4.4.3.1 Dialogfenster („Modal“) ... 44
6.4.4.3.2 `Booking`-Component ... 45
6.4.4.4 Anzeige der Änderungsanfragen ... 46
6.4.4.5 Einbinden weiterer Modals ... 47
6.4.4.6 Verbesserung der User Experience ... 48
6.4.4.7 Clientseitige Formular Validierung ... 48
6.4.4.8 Abspeichern der Formulardaten ... 50
6.4.4.9 Abfangen von Edge Cases ... 52
6.4.4.10 Anpassungen für Veranstaltung-Erstellen-Dialog ... 53
6.4.5 Spezifikation von Unit Tests ... 53
6.5 Ergebnis ... 57
7 Fazit ... 58
8 Literaturverzeichnis ... 61
Anhang ... 64

Abbildungsverzeichnis

Abbildung 1: React-Components 12
Abbildung 2: Virtueller DOM 14
Abbildung 3: Redux 16
Abbildung 4: Regelmäßige Nutzer der Web-Technologien React, AngularJS, Angular und Vue in Prozent 18
Abbildung 5: Triple-A-Schema 22
Abbildung 6: Benutzeroberfläche Veranstaltung erstellen 25
Abbildung 7: Hinweis zu Änderungsanfrage bei übergeordneter Aufgabe 27
Abbildung 8: Hinweis zu Änderungsanfrage direkt unter betroffener Buchung 28
Abbildung 9: Integration von React in das Monte-Tool 30
Abbildung 10: Use-Case-Diagramm 34
Abbildung 11: Component-Tree 35
Abbildung 12: Gliederungspunkte im Store, zugeklappt 40
Abbildung 13: Two way Binding 42
Abbildung 14: Benutzerauswahl Modal 44
Abbildung 15: Dienst mit Buchung 46
Abbildung 16: Dienste ohne Buchung 46
Abbildung 17: Validierung auf Input-Ebene 49
Abbildung 18: Rückmeldung der Validierung auf Formular-Ebene mittels Modal 50
Abbildung 19: Edge-Case-Modal 53
Abbildung 20: aktualisierter Component-Tree 57

Abkürzungsverzeichnis

API	Application Programming Interface, Programmierschnittstelle
BDUF	Big Design Upfront
CSS	Cascading Style Sheets
DOM	Document Object Model
DRY	Don't repeat yourself
HTML	Hypertext Markup Language
IDE	Integrated Development Environment, integrierte Entwicklungsumgebung
Props	Properties
RTL	React Testing Library
SPAs	Single Page Applications
SQL	Structured Query Language
UI	User Interface
UML	Unified Modeling Language
UUID	Universally Unique Identifier

Glossar

Action	Eine Action ist ein Konzept von Redux. Actions sind JavaScript Objekte, die Informationen beinhalten, um den Store zu aktualisieren. Bei diesen Informationen handelt es sich um einen `Type` und einen `Payload`.
Action Creator	Ein Action Creator ist eine Funktion, die eine Action erzeugt, also ein JavaScript Objekt zurückgibt.
Ajax	Abkürzung für „Asynchronous JavaScript and XML“. Konzept, nach dem asynchrone HTTP-Anfragen durchgeführt werden können, ohne dass die Webseite neu geladen werden muss.
Angular	Von Google entwickeltes JavaScript Framework.
AngularJS	Google entwickelte eine nicht abwärtskompatible Nachfolgeversion von Angular. Um beide Versionen unterscheiden zu können wird Version 1 meistens als AngularJS bezeichnet.
Application Programming Interface	Eine Programmierschnittstelle ermöglicht die Interaktion zwischen zwei Softwareprogrammen.
A TRIP	Akronym, welches das Merken der fünf Eigenschaften guter Tests vereinfachen soll: Automatic, Thorough, Repeatable, Independet und Professional.
Bad Smells	Schwachstellen im Software-Design
Big Design Upfront	Veraltetes Prinzip, Software zu entwickeln. Das endgültige Softwaredesign wird zu Beginn des Projektes festgelegt, noch bevor der Quellcode geschrieben wird.
Bootstrap	Frontend CSS-Framework
CakePHP	Framework für das Entwickeln von Anwendungen in PHP
Chrome Developer Tools	Im Google Chrome Browser integrierte Anwendung für Software-Entwickler, die das Analysieren und Debuggen von Anwendungen erleichtert.
Component	In sich abgeschlossener, unabhängiger Baustein der React-Anwendung, der innerhalb der Anwendung beliebig oft wiederverwendet werden kann.
Component-Tree	Baumdiagramm, das die Verschachtelung der React-Components und damit auch der Parent-Child-Beziehungen veranschaulicht.

Denial of Service	Bei einem Denial of Service wird der Zugriff auf einen Internetdienst verweigert. Dies kann durch eine Überlastung des Datennetzes oder durch einen mutwillig verursachten Server-Angriff bezweckt werden.
Document Object Model	Baumstruktur, die vom Browser erstellt wird. Jedes HTML-Tag bildet einen Knotenpunkt.
Edge Case	Grenzfall, der nur in Ausnahmesituationen eintritt.
Emergent Design	Aktuell beliebtes Prinzip, Software zu entwickeln. Während dem gesamten Entwicklungsprozess wird das Software Design fortlaufend an neu entstehende Anforderungen angepasst.
FaxJS	Von Jordan Walke entwickelte Software. Vorgänger von React.
Flux	Von Facebook entwickelte Technologie, welche die Idee eines globalen State-Managements hervorbrachte und einigen neueren Technologien wie beispielsweise Redux als Vorlage dienten.
Framework	Ein Grundgerüst, das die Entwicklung neuer Anwendungen erleichtern soll. Auf der einen Seite stellt ein Framework Funktionen bereit, die genutzt werden können. Auf der anderen Seite stellt ein Framework aber auch Ansprüche und Anforderungen an den zu entwickelnden Code.
Hypertext Markup Language	HTML ist eine textbasierte Auszeichnungssprache, die für das Erstellen von Webseiten eingesetzt wird. HTML-Dokumente können vom Browser angezeigt werden.
Integrierte Entwicklungsumgebung	Eine integrierte Entwicklungsumgebung stellt Entwicklern nützliche Werkzeuge beim Erstellen von Software zur Verfügung. Hierzu zählt zum Beispiel ein Editor, der den Code farbig hervorheben und formatieren kann, ein Debugger oder ein Interpreter.
Jasmine	Testframework für JavaScript Anwendungen. Alternative zu Jest.
JavaScript	1995 von Netscape entwickelte Programmiersprache, die bei der Programmierung von dynamischen Webseiten benötigt wird. Neben HTML und CSS ist JavaScript eine wichtige Technologie im Bereich Webentwicklung.
Jest	Von Facebook entwickeltes Testframework für JavaScript Anwendungen. In Kombination mit React Anwendungen ist es aktuell das Beliebteste.

JQuery	JavaScript Bibliothek, die darauf abzielt, DOM-Manipulationen, Event-Handling, Animationen und Ajax-Aufrufe zu vereinfachen.
jsdom	Testumgebung, die ähnlich einem Browser ist, jedoch auf den Head eines richtigen Browsers verzichtet. Es ist die Standardtestumgebung für das Testen mit Jest.
JSX	Syntaxerweiterung, die HTML sehr ähnlichsieht und im Regelfall in React-Components eingesetzt wird.
Matcher	In Jest werden Matcher eingesetzt, um Erwartungen an den Code aufstellen und testen zu können. Beliebte Matcher sind zum Beispiel `toBeNull()` oder `toBeDefined()`.
Mechanics	Die Mechanics von Martin Fowler sind Anleitungen, wie Refactorings in Teilschritte aufgeteilt und sicher durchgeführt werden können.
Mocha	Testframework für JavaScript Anwendungen. Alternative zu Jest.
Model-View-Controller	Beliebtes Design-Pattern für das Erstellen von Software, bei der die Bereiche Daten, Ansicht und Steuerung voneinander getrennt werden.
MySQL	Open Source Software für relationale Datenbankverwaltungssysteme
Netscape Communications	Ehemaliges Softwareunternehmen aus den USA, das 1994 gegründet und 2003 aufgelöst wurde.
Open Source	Open Source bedeutet, dass eine Software der Öffentlichkeit kostenlos zur Verfügung steht.
Props	In React Anwendungen werden Props dafür eingesetzt, um Daten von einem Parent- an einen Child-Component weiterzugeben.
Pyramid	Projekt von Microsoft, bei dem für das Textverarbeitungsprogramm Word ein Rewrite durchgeführt werden sollte. Das Projekt wurde vorzeitig abgebrochen.
Quelltextabdeckung	Die Quelltextabdeckung beschreibt, welcher Anteil des Quellcodes von Tests abgesichert wird.
React	Von Facebook entwickelte JavaScript Bibliothek für das Schreiben von Single-Page-Applications.
React Developer Tools	Erweiterung der Chrome Developer Tools. Werkzeug, mit dem das Debuggen von React Anwendungen vereinfacht wird.

Reducer	Konzept von Redux. Reducer aktualisieren den Store mit den Informationen, die sie von den Actions erhalten.
Redux	Technologie für globales State-Management. Beliebt in Kombination mit React.
Refactoring	Beim Refactoring wird der Quellcode einer Anwendung überarbeitet, mit dem Ziel, die Struktur und Wartbarkeit von diesem zu verbessern. Neue Funktionalitäten sollen beim Refactoring nicht implementiert werden.
Rewrite	Bei einem Rewrite wird der Code einer Anwendung verworfen und von Grund auf neu geschrieben.
Routing	Beim Routing wird festgelegt, welche Components bei einer bestimmten URL sichtbar sein sollen. Durch das Routing entsteht selbst in Single-Page-Applications das Gefühl, man navigiere durch mehrere Seiten.
Ruby	Objektorientierte Programmiersprache
Ruby on Rails	Framework für das Entwickeln von Webanwendungen in Ruby
Single-Page-Application	Webanwendung, die nur aus einem HTML-Dokument besteht.
Structured Query Language	Abfragesprache, mit der auf Datensätze einer relationalen Datenbank zugegriffen werden kann. Datensätze können mittels SQL-Befehlen erstellt, bearbeitet oder gelöscht werden.
State	Als State wird der Zustand einer Anwendung bezeichnet. Dieser ist darauf ausgerichtet, verändert zu werden. Abhängig vom State werden Änderungen am UI vorgenommen.
Store	Konzept von Redux. Ort, an dem der State der Anwendung abgespeichert wird. Alle Components können auf den Store zugreifen.
Style	Unter Style versteht man das Aussehen einer Anwendung, welches durch den Einsatz von CSS erreicht wird.
Strukturverlust	Strukturverlust bedeutet, dass das zugrunde liegende Software-Design zunehmend verschwindet, beispielsweise aufgrund von vermehrt eingesetzten Notlösungen.
Template	In CakePHP sind Templates Dokumente, die innerhalb des MVC-Patterns das „V“ verkörpern, also Views darstellen.
Testsuite	Mehrere Tests können eine Testsuite bilden. Diese können als Gruppe ausgeführt oder ausgelassen werden.

Triple-A-Schema	Schema für das Aufstellen von Tests. Die As stehen für Arrange, Act und Assert. Ein Test muss zunächst Vorbereitungen treffen, danach Aktionen ausführen und zum Schluss Erwartung mit der Realität abgleichen.
Two way binding	Prinzip, nach dem eine Änderung am Model, die Änderung der View zur Folge hat und umgekehrt.
TypeScript	TypeScript ist eine Programmiersprache, die auf JavaScript basiert und zusätzliche Features bereitstellt, durch die der JavaScript Code noch besser dokumentiert und validiert werden kann.
Unified Modeling Language	Vereinheitlichte Modellierungssprache, die dafür eingesetzt wird Softwareprozesse zu veranschaulichen.
Unit Test	Ein Unit Test überprüft einen kleinen, klar umrissenen Teil der Funktionalität. Er soll eine Garantie dafür liefern, dass der Code genau das bewirkt, was der Entwickler im Sinn hatte.
Universally Unique Identifier	Zufällig generierter String, der Einzigartigkeitscharakter aufweist und dafür eingesetzt wird, um Informationen voneinander zu unterscheiden.
Use-Case-Diagramm	Auch als Anwendungsfalldiagramm bekannt. Ein Use-Case-Diagramm stellt aus Anwendersicht dar, welche Funktionen ein System beinhaltet.
User Experience	Die User Experience beschreibt das Erlebnis eines Nutzers bei der Interaktion mit einem System. Ein benutzerfreundliches System, bietet eine gute User Experience.
User Interface	Benutzerschnittstelle. Stelle an der ein Benutzer mit einer Maschine in Interaktion treten kann. Bei Software ist mit der Benutzerschnittstelle meist die grafische Benutzeroberfläche gemeint.
Vanilla JavaScript	JavaScript in seiner ursprünglichen Form, ohne das Verwenden von Frameworks oder Bibliotheken.
Virtueller DOM	Vereinfachte Kopie des reellen DOM. Da Änderungen am richtigen DOM sehr aufwendig und zeitintensiv sind, gibt es in React den virtuellen DOM. Änderungen werden zunächst an diesem durchgeführt. Dann werden Abweichungen zum richtigen DOM berechnet und Anpassungen werden möglichst ressourcenschonend ausgeführt.
Vue	2014 erschienenes JavaScript Framework.

Wartbarkeit	Die Wartbarkeit ist eine Kenngröße, die bestimmt mit welchem Aufwand Änderungen an einer Anwendung durchgeführt werden können. Sie ist ein wichtiges Bewertungskriterium für Software-Qualität.
Webtechnologie	Werkzeuge, mit denen Webentwickler arbeiten. Beliebte Webtechnologien sind unter anderem HTML, CSS, XML, PHP, Java oder SQL.
XHP	Von Facebook entwickelte Software. PHP-Bibliothek, die im Gegensatz zu seinem Nachfolger React eher schlecht für den Bau von dynamischen Webseiten geeignet war.

1 Problemstellung: Verbesserung der Wartbarkeit von Code

> „Du hast es nicht selbst geschrieben.
> Wer auch immer es geschrieben haben mag – er ist weg.
> Es gibt keine Dokumentation.
> Der Code ist ziemlich hässlich.
> Und 500.000 Zeilen lang.
> Niemand weiß, wie es funktioniert.
> Jeder weiß, wann es laufen muss.
> Und jetzt?“ (Bommer et al., 2008, S. 2)

Das ist wohl das Worst-Case-Szenario für viele Softwareentwickler. Was kann in so einem Fall getan werden? Beziehungsweise wie kann verhindert werden, dass es so weit kommt? Es gibt eine Handvoll Lösungsansätze, die sich mit der Problematik schwer erweiterbaren Codes beschäftigen. In diesem Zusammenhang ist das Refactoring anzuführen, sowie das Einbeziehen einer modernen Webtechnologie. Auch die Firma Evidentmedia wurde mit dem Problem konfrontiert, dass Software mit der Zeit wächst und dadurch Einbuße im Bereich der Wartbarkeit entstehen.

Ihre 2015 entwickelte Webanwendung Monte-Tool beinhaltet einen Veranstaltungseditor, bei dem es zunehmend schwerer fällt, Anpassungen vorzunehmen. Die Entwickler dieser Anwendung sind immer noch Teil des Teams, aber genau wie im Einstiegsbeispiel ist der Code mittlerweile sehr lang, an einigen Stellen hässlich und für neue Teammitglieder, die ihn nicht mitentwickelt haben, schwer zu verstehen. Um dieses Problem zu visualisieren, wird im Folgenden darauf eingegangen, wie das Einfügen eines neuen Dienstes bewerkstelligt wurde. Hierfür wurde mit JQuery[1] gearbeitet.

[1] https://jquery.com/

```
1. function admCreateService(element, copyService) {
2. copyService = (typeof copyService === 'undefined' ? false : copyService);
3. // save parent element
4. var listGroup = $($(element).parents('.list-group'));
5.
6. // last element should have highest service number
7. var elementsInListGroup = listGroup.children().length - 1;
8. var lastElem = $(listGroup.children()[elementsInListGroup]);
9. var nameInput = $(lastElem.find('input[name*="name"]')[0]);
10. // tasks[0][services][0][name] filter task and service number
11. var nrArr = nameInput.attr('name').match(/[0-9]+/g);
12. var taskNr = nrArr[0];
13. var serviceNr = parseInt(nrArr[1])+1;
14.
15. var newElem = lastElem.clone();
16.
17. // get value of service to copy
18. var elemValue = $($(element).parents('.list-group-item').find('in-
    put[name*="name"]')[0]).val();
19. if(copyService == false) elemValue = '';
20.
21. // update numbers in input fields
22. $(newElem.find("input")).each(function(){
23.       var elem = $(this);
24.       var textFragments = elem.attr('name').match(/[a-zA-Z-_]+/gi);
25.       [...]
26.       elem.attr('id', textFragments[0] + '-' + taskNr + '-' + textFrag-
    ments[1] + '-' + serviceNr + '-' +textFragments[2]);
27.       elem.attr('name', textFragments[0] + '[' + taskNr + ']' +  '[' +
    textFragments[1] + ']' + '[' + serviceNr + ']' + '[' + textFragments[2] +
    ']');
28.       if(textFragments[2] == 'name') {
29.             elem.val(elemValue);
30.       }
31. });
32. [...]
33. listGroup.append(newElem);
34. }
```

In diesem Abschnitt wird deutlich, wie kompliziert vorgegangen werden muss, um das Hinzufügen eines Dienstes zu implementieren. Verkürzt gesagt, wird in diesem Code-Fragment das letzte Dienst-Element identifiziert, in dessen Gruppe ein neues Dienst-Element eingefügt werden soll. Jeder Dienst benötigt einen Wert für das `name`-Attribut, in der gleichen Form wie das Beispiel aus Zeile zehn. Aus diesem Wert können schließlich die Nummer der Aufgabe und des Dienstes extrahiert werden. Für das Hinzufügen eines weiteren Dienstes, wird die Nummer der Aufgabe wiederverwendet, die Nummer des Dienstes wird um eins erhöht. Als nächstes muss

ein neues Dienst-Element zusammengebaut werden. In Zeile 15 werden die Elemente des alten Dienstes geklont und in die neue Variable `newElem` gespeichert. Für jedes Eingabefeld des neuen Dienstes müssen die Werte für die Attribute `id` und `name` passend zusammengebaut werden. Zum Schluss wird das neue Dienst-Element in das Document Object Model (DOM) eingefügt.

Diese Art und Weise neue Elemente in das Formular einzufügen ist äußerst aufwendig und schwer erweiterbar. Ziel der Arbeit ist es, dieses Problem zu lösen und die Wartbarkeit der Anwendung nachhaltig zu verbessern, sodass zukünftig einfacher Änderungen durchgeführt und Erweiterungen vorgenommen werden können.

Dafür soll in einem ersten Schritt betrachtet werden, was man unter Wartbarkeit versteht und welche Möglichkeiten es gibt, Software zu warten. Zudem soll herausgestellt werden, welche Wartungstechnik für die Überarbeitung des Monte-Tool-Veranstaltungseditors geeignet ist. In einem zweiten Schritt werden die grundlegenden Konzepte von React, einer modernen Webtechnologie, beleuchtet und deren Vorzüge in Bezug auf die Wartbarkeit einer Anwendung. Als nächstes wird das Testen einer Anwendung behandelt: eine Möglichkeit sicherzustellen, dass die Funktionalität der Anwendung mit den Erwartungen des Entwicklers im Einklang steht. Im Anschluss folgt die Beschreibung der praktischen Umsetzung und ihrer Ergebnisse. Diese besteht darin, den Code des Veranstaltungseditors des Monte-Tools mithilfe der modernen Webtechnologie React neu zu schreiben. Schlussendlich soll die Frage beantwortet werden, ob ein Refactoring mithilfe einer modernen Webtechnologie geeignet ist, um die Wartbarkeit einer Webanwendung zu verbessern.

Im Speziellen ist diese Arbeit für die Firma Evidentmedia von Interesse, die den Event-Editor nutzt und zukünftig auch weiterentwickeln möchte. Darüber hinaus für Firmen, die ebenfalls ein Refactoring ihrer Softwareanwendung oder die Einbeziehung einer neuen Technologie planen. Sie werden durch diese Arbeit über den aktuellen Forschungsstand zu den Themen Refactoring und Integration moderner Webtechnologien informiert und können sich durch den hohen Praxisbezug der Arbeit Best Practices vor Augen führen, angewendet an einem konkreten Beispiel.

2 Wartbarkeit

Die Qualität von Software ist abhängig von verschiedenen Einflussfaktoren. Einer davon ist die Wartbarkeit. Gemäß der weltweit anerkannten Norm ISO/IEC 25010:2011 (International Organization for Standardization, 2011) existieren neben der Wartbarkeit sieben weitere Teilbereiche, in denen sich die Qualität einer Software unter Beweis stellen muss. Zu diesen zählen

die funktionale Eignung, Performance Effizienz, Kompatibilität, Benutzerfreundlichkeit, Verlässlichkeit, Sicherheit und Portabilität. Im Folgenden wird lediglich die Wartbarkeit, ein ausgesprochen wichtiges Bewertungskriterium für Softwarequalität, näher betrachtet, denn diese gilt es beim Veranstaltungseditor des Monte-Tools zu verbessern. (Visser, 2016, S. 2)
Bommer, Spindler und Barr (2008) definieren den Begriff der Wartbarkeit folgendermaßen:

> „Die Wartbarkeit ist ein äußerst wichtiges Kriterium in der Softwarewartung und bestimmt, mit welchem Aufwand und welcher Qualität Änderungen in einem Softwaresystem durchgeführt werden können." (Bommer et al., 2008, S. 25)

Ein Grund für die große Relevanz der Wartbarkeit besteht im wirtschaftlichen Mehrwert, der sich aus ihr ergibt. Bei einer Anwendung mit guter Wartbarkeit, können Probleme leichter behoben und Anpassungen schneller durchgeführt werden. In Folge werden weniger Mitarbeiter benötigt, wodurch die Kosten für Wartungsarbeiten sinken. Durch rapide Entwicklungsfortschritte kann die Zeitspanne bis zur Vermarktung einer Software verkürzt und konkurrierende Firmen, die an einer ähnlichen Anwendung arbeiten, können ausgestochen werden. Für schlecht wartbare Anwendungen hingegen gibt es keine Zukunft. Wenn bei jeder Anpassung neue Fehler in den Code eingearbeitet werden, muss die Entwicklung in vielen Fällen eingestellt werden. (Visser, 2016, S. 3-4)
Neben den ökonomischen Vorzügen gibt es einen weiteren Grund, warum die Wartbarkeit von Software wichtig ist: Sie begünstigt alle weiteren Qualitätsaspekte, die in der oben genannten Norm ISO/IEC 25010:2011 (International Organization for Standardization, 2011) aufgeführt werden. In einem sehr gut wartbaren System wird beispielsweise das Schließen einer Sicherheitslücke leichter gelingen als in einem schlecht wartbaren System. Ganz gleich bei welchem der acht Themengebiete – es geht immer um das passende Modifizieren des Codes, welches umso einfacher abgewickelt werden kann, je besser die Wartbarkeit ausfällt. (Visser, 2016, S. 4)

3 Wartungstechniken

Mit Wartungstechniken lässt sich Einfluss auf die Wartbarkeit nehmen. Hierzu zählen das Refactoring, sowie die Integration moderner Webtechnologien, die Gegenstand des folgenden Kapitels sind.

3.1 Refactoring

Die Art und Weise, wie man das Design einer Software entwickelt, hat sich im Laufe der Zeit verändert. Früher wurde das Prinzip des „Big Design Upfront“ (BDUF) eingesetzt, das mittlerweile als veraltet gilt. Diesem liegt der Gedanke zu Grunde, dass man bereits in der Anfangsphase eines Projektes, noch vor dem Schreiben des Quellcodes, das endgültige Design der Software entwickelt, das später nicht mehr verändert oder angepasst werden muss. (Roock & Lippert, 2004, S. 1, 11)

Diese Strategie führt oftmals zu Schwierigkeiten. Für gewöhnlich entstehen während dem Entwicklungsprozess weitere Ansprüche an die Software, denen das zuvor festgelegte Design nicht ohne Weiteres gerecht werden kann. Man behilft sich in diesen Fällen mit Notlösungen. Häufen sich diese, verschwindet das zugrunde liegende Design zunehmend. Diesen Vorgang bezeichnet man auch als Strukturverlust. In Entwicklerkreisen wird hierbei von einem historisch gewachsenen System gesprochen. (Roock & Lippert, 2004, S. 1, 11-12)

Mit voranschreitendem Strukturverlust fällt es immer schwerer, mit der Software zu arbeiten und diese weiterzuentwickeln. Jegliche Anpassungen sind mit sehr hohem Aufwand verbunden. Aus diesem Grund wird heutzutage anders vorgegangen. Vor allem agile Entwicklungsprozesse werden favorisiert. In agilen Entwicklungsprozessen rechnet man damit, dass sich die Anforderungen der Software verändern können und reagiert darauf mit Anpassungen des Software-Entwurfs. (Roock & Lippert, 2004, S.1, 11-12)

Aktuell ist das „Emergent Design“ eine beliebte Technik. Bei dieser entsteht das Design der Software, Stück für Stück. Während dem gesamten Entwicklungsprozess wird durchgehend daran gearbeitet, sodass es auch neu entstandenen Bedürfnissen entspricht. Entgegen der traditionellen Haltung, dass Änderungen des Softwaredesigns aufgrund vorhergehender Fehler nötig wären, wird ein neuer Blickwinkel eingenommen: eine Modifikation des Entwurfs stellt die Weiterbildung und Weiterentwicklung der Ingenieure unter Beweis. Die kontinuierliche Verbesserung des Software-Designs hat sich als ein wesentlicher Arbeitsschritt beim Entwickeln von Software herauskristallisiert, dem Refactoring. (Roock & Lippert, 2004, S. 1, 11-12)

> „Refactoring bedeutet zunächst, die interne Struktur der Software zu verändern, um die Software besser verständlich und leichter änderbar zu gestalten, ohne dabei das beobachtbare Verhalten der Software zu modifizieren“ (Roock & Lippert, 2004, S. 2) .

Beim Refactoring geht es also nicht darum, neue Funktionalität zu implementieren. Lediglich die Softwarestruktur wird verändert. Der Nutzer der Software wird nach einem Refactoring in

der Regel keinen Unterschied gegenüber der vorhergehenden Version bemerken. Mitunter kann es zu Abweichungen des Laufzeitverhaltens kommen, die allerdings überwiegend so gering ausfallen, dass sie ausschließlich messbar sind und dem Nutzer nicht weiter auffallen. (Roock & Lippert, S. 13-19)

3.1.1 Gründe für ein Refactoring

Wenn ein Programm funktioniert, wieso sollte man es dann ändern? In der Tat hat der Compiler kein Problem mit unaufgeräumtem Code. Problematisch wird es erst, wenn Menschen in den Prozess involviert werden. Das ist zwangsläufig der Fall, wenn Änderungen vorgenommen werden sollen. Unsauberer Code erschwert es dem Menschen die richtige Stelle für Ausbesserungen zu identifizieren und begünstigt den unbeabsichtigten Einbau von Fehlern. (Fowler, 2004, S. 5) Dieser Umstand relativiert das Sprichwort „Never change a running system“ und zeigt, dass es sinnvoll ist, Refactoring einzusetzen.

In erster Linie wird Refactoring durchgeführt, um Schwachstellen des Software-Designs auszuräumen. Diese Schwachstellen werden auch als „Bad Smells“ oder übelriechender Code bezeichnet. Laut Fowler (2004) geht diese Bezeichnung auf seinen Kollegen und Co-Autor Kent Beck zurück. Eine mögliche Ursache für „Bad Smells“ ist die Missachtung der „Don’t- repeat-yourself“ (DRY) - Maxime. Gemäß dieser sollten Code-Abschnitte, wenn möglich, nur einmal im Quellcode vorkommen. (Roock & Lippert, 2004, S. 18)

Durch das Beseitigen von Redundanzen kann die Anzahl der Code-Zeilen reduziert werden. Im Normalfall hat das keine Auswirkung auf die Performanz des Systems, aber auf zukünftige Korrekturarbeiten. Die Gefahr entfällt, gedoppelte Code-Stellen zu übersehen, die ebenfalls einer Anpassung bedürfen. (Fowler, 2004, S. 43)

Neben den Bad Smells gibt es weitere Argumente, die dafürsprechen, Refactoring einzusetzen. Refactoring feilt am Design der Software und arbeitet Strukturverlust entgegen. Außerdem fördert Refactoring die Verständlichkeit des Codes. Dadurch können auch andere Entwickler die Arbeit an dem Code ohne Probleme aufnehmen. Aber auch der Verfasser muss nicht alles im Gedächtnis behalten, sondern kann sich jederzeit neu mit der Logik vertraut machen. Zudem erleichtert verständlicher Code das Aufdecken von Fehlern. (Fowler, 2004, S. 43-45)

Alle bisher genannten Motive ergeben ein weiteres: Refactoring spart Zeit. Zunächst könnte man meinen, genau das Gegenteil sei der Fall. Refactoring an sich kostet Zeit. Gleichzeitig reduziert sich der zeitliche Aufwand beträchtlich, wenn es leichtfällt, Änderungen durchzuführen, ein adäquates Design zugrunde liegt, der Code verständlich ist und schnell Fehler gefunden werden. (Fowler, 2004, S. 45-46)

3.1.2 Vorgehensweise bei einem Refactoring

Beim Refactoring wird an einem funktionstüchtigen System gearbeitet. Deshalb besteht die Gefahr, dass sich während des Refactoring-Prozesses neue Fehlerquellen in die Software einschleichen. Um dies zu vermeiden, sollten zwei Leitlinien berücksichtigt werden. Zum einen wird empfohlen, das Refactoring in möglichst kleine, in sich geschlossene Schritte aufzusplitten. Zum anderen wird geraten, nicht eher mit dem Refactoring zu beginnen, bis automatisierte Tests zur Verfügung stehen, mit denen die Funktionalität der Anwendung laufend überprüft werden kann. (Roock & Lippert, 2004, S. 19)

Anfänger im Bereich Refactoring machen tendenziell den Fehler, kleinere Anpassungen zu kombinieren und auf einmal zu realisieren. Bei vielen, zeitgleich ausgeführten Änderungen besteht die Gefahr, dass Defekte unbemerkt eingebaut werden. Dann kann meist nicht mehr unmittelbar festgestellt werden, welche Adaption zum Verlust der Funktionsfähigkeit des Systems geführt hat. Daher macht es Sinn, geplante Umstrukturierungen in möglichst kleine Portionen aufzugliedern. Dies bietet den Vorteil, zu jedem Zeitpunkt genau zu wissen, welcher Arbeitsschritt für die Einschränkung der Lauffähigkeit des Systems verantwortlich ist. (Roock & Lippert, 2004, S. 19-20)

Ein Refactoring in kleine Einheiten zu zerlegen kann mitunter kompliziert sein. Martin Fowler hat sich dieser Schwierigkeit angenommen. Er hat Leitlinien verfasst, sogenannte „Mechanics“. Diese bieten jedem Software-Entwickler eine Anleitung, wie verschiedenste Arten von Refactoring am günstigsten in Teilschritte zerlegt werden können, sodass das System nach jedem einzelnen Schritt ausführbar bleibt. (Fowler, 2004)

„Mechanics“ arbeiten bevorzugterweise mit Umleitungen. Roock und Lippert (2004) ziehen hier eine Parallele zum Straßenbau. Ähnlich wie bei Infrastruktur-Bauarbeiten dienen Umleitungen dazu, dass das System beziehungsweise die Straße trotz Umstrukturierungen weiterhin genutzt werden kann. Kurzfristig wird die Struktur des Systems noch komplizierter als bei der Ausgangssituation. Erst nachdem die neue Infrastruktur vollständig errichtet wurde, kann die Umleitung entfernt werden. In diesem Kontext ist es ratsam, das Refactoring zügig zu vollenden und eine möglichst geringe Anzahl an Refactoring-Prozessen gleichzeitig zu betreiben. Ansonsten würden sich Umleitungen im System häufen und das Gegenteil von dem bezwecken, was angestrebt wird: eine Verschlechterung der Systemstruktur. (Roock & Lippert, 2004, S. 21)

3.1.3 Geeignete Zeitpunkte für Refactoring

Fowler (2004) ist grundsätzlich der Meinung, dass es überflüssig sei, im Voraus Zeit für Refactoring zu kalkulieren. Seine Empfehlung fällt dahingehend aus, fortlaufend Refactoring

anzuwenden, immer dann, wenn etwas anderes auf dem Zeitplan steht, das durch ein Refactoring erleichtert werden kann. Refactoring erweist sich vor allem als nützlich, wenn die Anwendung um eine neue Funktion erweitert werden soll, dies aber nicht ohne weiteres möglich ist. Nach dem Refactoring ist es dann ein Leichtes, die eingeplante Funktion und zukünftig auch weitere, ähnliche Funktionalität in die Anwendung einzubauen. (Fowler, 2004, S. 46-48)

3.1.4 Geeigneter Zeitpunkt für einen Rewrite

In bestimmten Fällen macht es mehr Sinn, den Code ganzheitlich neu zu schreiben. Beispielsweise, wenn Refactoring deutlich aufwendiger wäre als ein Neustart. Fowler (2004) gibt zu bedenken, dass eine Beurteilung schwer fällt, wann ein Neubeginn die richtige Wahl ist. Ein eindeutiges Indiz dafür, dass ein Neustart der Software guttun würde, ist, wenn die Funktionalität des Codes von vornherein nicht gewährleistet ist, denn die Voraussetzung für ein erfolgreiches Refactoring im klassischen Sinne, ist eine solide Ausgangsbasis mit eventuell sperrigem, aber funktionstüchtigem Code. (Fowler, 2004, S. 56)

Spolsky (2000) warnt davor alten Code, an dem lange gearbeitet wurde, zu verwerfen und von vorne anzufangen. Ein nachvollziehbarer Grund hierfür ist die Tatsache, dass mit der Zeit bereits viele Fehler entdeckt und behoben werden konnten, die einem neuen Entwicklerteam zu Beginn des Vorhabens womöglich gar nicht bekannt sind und eventuell ein zweites Mal gemacht werden. Einige große Unternehmen sind bereits bei einem Rewrite gescheitert. Netscape beispielsweise arbeitete nahezu drei Jahre an einer Nachfolgeversion 6.0 ihres gleichnamigen Webbrowsers, der von Grund auf neu geschrieben wurde. Während dieser langen Zeitspanne sank der Marktwert des Unternehmens stetig. Auch Microsoft startete einen Rewrite-Versuch des Textverarbeitungsprogrammes Word, ein zum Scheitern verurteiltes Projekt mit dem Namen „Pyramid“, das vorzeitig abgebrochen und verworfen wurde. Im Allgemeinen ist es leichter, Code zu schreiben als Code zu verstehen. Aus diesem Grund wird die Entscheidung für einen Rewrite oftmals zu voreilig getroffen. (Spolsky, 2000)

3.2 Integration moderner Webtechnologien

Eine weitere Möglichkeit eine Webanwendung zu überarbeiten, besteht darin eine moderne Webtechnologie einzubinden. Zu modernen Webtechnologien zählen beispielsweise JavaScript Bibliotheken und Frameworks. Auch wenn die Anwendung bisher ohne den Einsatz dieser Webtechnologie ausgekommen ist, kann die Integration einer neuen Technologie Vorteile haben. In diesem Zusammenhang stellt sich die Frage, warum nicht gleich zu Projektbeginn mit dieser Technologie gearbeitet wurde. Die Entwicklung neuer Webtechnologien macht rasante Fortschritte (Petereit, 2017). Es ist denkbar, dass es die moderne Webtechnologie beim Start

des Projektes noch nicht gab, oder dass es zu diesem Zeitpunkt noch an Erfahrungsberichten, einer umfangreichen Community oder schlicht an Bekanntheit mangelte. Im Folgenden werden Vor- und Nachteile behandelt, die die Integration moderner Webtechnologien mit sich bringen.

3.2.1 Vorteile

Der zentrale Beweggrund für das Verwenden einer modernen Webtechnologie ist die Zeitersparnis. Durch vorgefertigte Funktionen, Bausteine oder Konzepte, die komplizierte Abläufe vereinfachen, gelingt es Entwicklern schnellere Projektfortschritte zu erzielen. Im kommerziellen Bereich entspricht Zeit, Geld. Folglich geht mit der Einsparung an Zeit ein Kostenersparnis einher. Dies ist vor allem bei Firmen mit knappem Budget oder großem Zeitdruck bei der Vollendung eines Projektes vorteilhaft. (Petereit, 2016)

In erster Linie macht sich der Einsatz eines Frameworks bei größeren Anwendungen bezahlt. Ähnlich wie eine Schablone entfaltet das Framework sein ganzes Potential vor allem dann, wenn an vielen Stellen dessen Vorzüge genutzt werden. (*Frameworks*, o. D.)

Durch ein Framework kann außerdem die zu produzierende Code-Menge verringert werden, indem beispielsweise Komponenten mehrfach verwendet werden. Das sorgt für mehr Übersichtlichkeit und kommt schlussendlich der Wartbarkeit der Anwendung zugute. (Schieferbein, 2018)

3.2.2 Nachteile

Demgegenüber ist zu berücksichtigen, dass es eine gewisse Zeit kosten wird, sich in eine neue Technologie einzuarbeiten. Die angestrebte Zeitersparnis beim Entwickeln einer Anwendung tritt erst ein, wenn die Entwickler das einzubindende Framework beherrschen oder zumindest über ein Grundverständnis verfügen. (Petereit, 2017)

Grundsätzlich sind Frameworks auch für weniger erfahrene Programmierer leicht erlernbar. Kritisch betrachtet, ist das Erlernen einer modernen Webtechnologie für Laien nicht empfehlenswert. Bei Einsteigern fehlt möglicherweise das nötige Wissen über die zugrunde liegende Programmiersprache. Meist wird Anfängern deshalb gar nicht bewusst, an welchen Stellen genau die Webtechnologie Arbeit einspart. (Wießeckel, 2014)

Zudem kann eine eingebundene Webtechnologie zu Problemen führen, wenn die Weiterentwicklung eingestellt oder die Unterstützung der Technologie aufgehoben wird. Nicht nur bei Webtechnologien mit kleineren Communitys ist das eine berechtigte Sorge. Bei Webtechnologien, an denen große Entwicklerteams arbeiten und die von einer Vielzahl an Anwendern eingesetzt werden, ist die langfristige Unterstützung zwar wahrscheinlicher, jedoch nicht abgesichert. (Wießeckel, 2014)

Außerdem fällt die Auswahl einer passenden modernen Webtechnologie oftmals schwer aufgrund der enormen Angebotsvielfalt (Petereit, 2017).

3.3 Passende Wartungstechnik für das Praxisprojekt

Im Falle des Event-Editors liegt funktionstüchtiger Code vor. Aus Fowlers Sicht spräche demnach nichts gegen Refactoring (Fowler, 2004, S. 56). Allerdings dürfen die vielen Vorzüge einer modernen Webtechnologie nicht in Vergessenheit geraten, von denen auch Evidentmedia profitieren möchte. Jedoch stellt eine moderne Webtechnologie Anfordernisse, die im Code berücksichtigt werden müssen. Tatsächlich ist es gar nicht möglich, eine moderne Technologie einzubinden, ohne den Aufbau der Anwendung komplett zu überdenken. Deshalb ist in Kombination mit dem Einbinden einer neuen Technologie, ein Rewrite die einzige Möglichkeit, die Anwendung zu renovieren.

Nichtsdestotrotz geht es beim Praxisprojekt nicht darum neue Funktionalität beim Veranstaltungseditor zu implementieren. Aus diesem Grund trifft Rook und Lipperts grundlegende Definition von Refactoring (siehe 3.1) immer noch zu. Es wird sich allerdings nicht um ein Refactoring im klassischen Sinne handeln, wie es von Fowler (2004) oder Roock und Lippert (2004) im Detail beschrieben wird, mit temporären Umleitungen oder dem Befolgen der als „Mechanics“ bezeichneten Anleitungen, sondern um ein Refactoring in Form eines Rewrites.

4 React: eine moderne Webtechnologie auf dem Vormarsch

Eine angesehene, zunehmend beliebter werdende Webtechnologie ist React. Ungefähr im Jahr 2010 begann das Unternehmen Facebook mit dessen Entwicklung. Der Grund für dieses neue Projekt war, dass immer wieder Schwierigkeiten bei der Instandhaltung von Code auftraten, vor allem bei der Arbeit größerer Teams an Single-Page-Applications (SPAs). Dies beeinträchtigte Arbeitsgeschwindigkeit und -qualität. (Zammetti, 2020, S.44)

Zunächst wurde XHP[2], eine PHP-Erweiterung etabliert, die den PHP-Code einfacher lesbar machen sollte. Die Erweiterung ermöglichte die Zerlegung der Benutzeroberfläche in voneinander weitestgehend unabhängige Komponenten. 2011 entstand FaxJs[3], ein Projekt, das bereits viele Ähnlichkeiten zum späteren React aufweist. Entwickelt wurde es vom Ingenieur Jordan Walke. (Zammetti, 2020, S. 44-45)

Im April 2012 folgte ein weiterer Meilenstein in der Entwicklungshistorie von React: Facebook übernahm Instagram. Die neu akquirierte Firma stellte schon bald Ansprüche auf die Nutzungsrechte von React. Dies bewog Facebook zu dem Schritt, aus React eine Open-Source-

[2] http://hhvm.github.io/xhp-lib/about
[3] https://github.com/jordwalke/FaxJs

Anwendung zu machen und eine Community aufzubauen, die sich fortan um die kontinuierliche Weiterentwicklung von React kümmern sollte. Neben Jordan Walke war Pete Hunt ein weiterer zentraler Akteur, der die Veröffentlichung von React als Open-Source-Bibliothek im Jahr 2013 möglich machte. (Zammetti, 2020, S. 45-46)

4.1 Leistungsspektrum von React

React ist eine JavaScript Bibliothek für das Schreiben von Webanwendungen. Im Konkreten handelt es sich bei diesen um SPAs. Die Besonderheit bei SPAs liegt darin, dass sie ausschließlich im Browser ausgeführt werden. Ein Datenaustausch mit dem Server findet zwar statt, um unter anderem Bilder und Fonts einzubinden, aber es wird kein HTML-Code vom Server bezogen. Besonders vorteilhaft erweisen sich SPAs bei hochgradig dynamischen, interaktiven Webseiten. Aber auch abseits des Web kann React eingesetzt werden, beispielsweise bei der Erstellung von iOS- oder Android-Anwendungen. (Hartmann & Zeigermann, 2020, S. 1-4)

Ein Kernkonzept von React ist es, die Benutzeroberfläche in Components zu untergliedern, die in sich abgeschlossen und unabhängig sind. Neben der Herangehensweise, Components zu schreiben, stellt React weitere Dinge zur Verfügung: Props, State, den virtuellen DOM[4] und Style. (Zammetti, 2020, S. 46-47)

Darüber hinaus ist es gebräuchlich, JSX in React Anwendungen zu verwenden, eine Spracherweiterung, die HTML sehr ähnlichsieht (Stefanov, 2017, S. 65-84). Im Folgenden wird näher auf diese Utensilien eingegangen.

4.1.1 Components

Components sind die Bausteine der Anwendung. Sie können unterschiedlich komplex sein und werden beliebig zur fertigen Anwendung kombiniert (siehe Abbildung 1).

[4] Im Sprachgebrauch wird „der“ als Artikel für DOM verwendet, obwohl es genau genommen „das Document Object Model“ sein müsste.

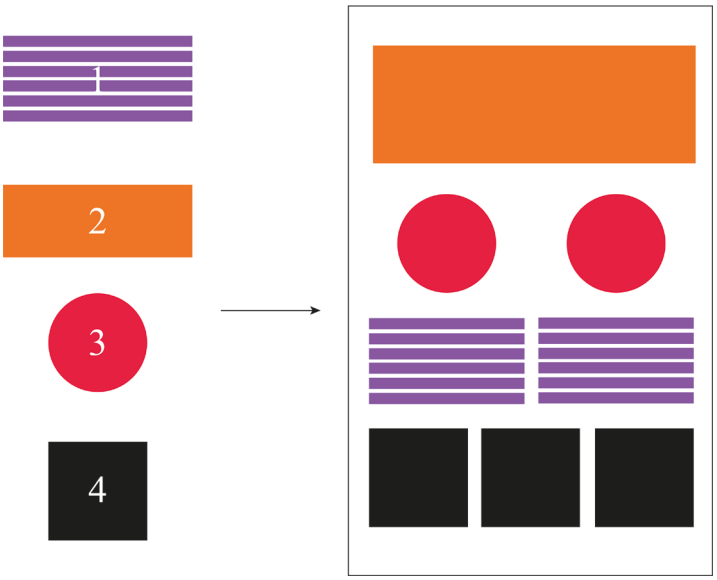

Abbildung 1: React-Components in Anlehnung an Holla, (o. D.)

Normalerweise werden kleine, einfach strukturierte Components zu komplizierteren vereinigt. Im Gegensatz zu anderen Frontend-Bibliotheken, die auf eine separierte Behandlung von Benutzeroberfläche und Logik setzen, beinhalten React-Components den Code von User Interface (UI) und Logik am gleichen Ort.

Components in React werden als deklarativ bezeichnet. Das bedeutet, dass sich die Components ähnlich wie eine mathematische Funktion verhalten. Ein bestimmter Eingabewert führt zu einem bestimmten Ergebniswert. Übertragen auf die Components bedeutet es, dass ein bestimmter Zustand einer Komponente immer zu einem bestimmten UI führt. Dieses Prinzip nennt sich „UI as a function“. (Hartmann & Zeigermann, 2020, S. 4-7)

4.1.2 Daten der Components

Bei den Daten der Components unterscheidet man zwischen zwei Arten: Props und State. Props ist die Abkürzung für Properties, was so viel bedeutet wie Eigenschaften. State kann mit „Zustand“ übersetzt werden. Props kommen von außerhalb des Components. Sie werden von dem Ort, an dem der Component eingebunden ist, an diesen weitergegeben. State auf der anderen Seite wird innerhalb des Components deklariert und steht auch nur dort zur Verfügung. (Zammetti, 2020, S. 55-59)

4.1.2.1 Props

Jeder Parent-Component kann Informationen in Form von Props an einen Child-Component weiterreichen. Die Reihenfolge ist hierbei von großer Bedeutung. Umgekehrt wäre die Weitergabe von Daten nicht möglich. Props werden also benötigt, um Informationen für untergeordnete Components verfügbar zu machen. Im Zusammenhang mit Props gibt es zwei wichtige

Vorschriften. Props werden nur zu einem bestimmten Zeitpunkt weitergereicht, nämlich in dem Moment, in dem der Child-Component erstellt wird. Außerdem sind Props anschließend unveränderbar. Das scheint zunächst sonderbar, denn die Informationen sollten flexibel und modifizierbar sein. An dieser Stelle kommt der sogenannte State ins Spiel, der das Ändern zentraler Daten ermöglicht. (Zammetti, 2020, S. 55-56)

4.1.2.2 State

Den State könnte man auch als Speicher der Components beschreiben. Nur manche Components benötigen solch einen Speicher. Man unterscheidet hierbei zwischen „stateful“ Components (Components mit Speicher) und „stateless“ Components (Components ohne Speicher). State bietet neben Props die Möglichkeit, die Components mit Informationen zu versorgen und somit auch Einfluss auf die Components auszuüben. Im Gegensatz zu Props sind Daten, die im State gespeichert werden, darauf ausgerichtet modifiziert zu werden. Ein typisches Beispiel hierfür sind Benutzereingaben. Bei Änderungen des State, nimmt React Anpassungen am virtuellen DOM vor, die dann für minimale Adaption des reellen DOM sorgen und somit zum erneuten Rendern aller betroffenen Komponenten führen. Oftmals wird State in einer Variablen mit dem Namen `state` gespeichert, die ein Objekt enthält. Diesbezüglich gibt es allerdings keine verbindlichen Vorschriften. (Zammetti, 2020, S. 57-60)

4.1.3 Virtueller DOM

Der Browser parst den Hypertext Markup Language (HTML)-Code und erstellt simultan den DOM, eine Baumstruktur, bei der jedes HTML-Tag einen Knotenpunkt bildet. Der DOM steht in engem Zusammenhang mit der Ausgabe auf dem Bildschirm und stellt eine Programmierschnittstelle (API) bereit, mithilfe derer der Seiteninhalt manipuliert werden kann. Bei einer Aktion, die eine Änderung der Seite hervorruft, wie beispielsweise einer Nutzereingabe, muss der Browser eine erhebliche Menge Arbeit verrichten, um den DOM und die Seite zu aktualisieren. Deshalb gibt es bei React das Konzept des virtuellen DOM. Es ist ein zusätzlicher DOM, welcher sich eine Ebene über dem reellen DOM befindet. React verändert nicht den reellen DOM unmittelbar. Stattdessen übermittelt React zunächst die Änderungen an den virtuellen DOM und aktualisiert diesen. Mithilfe eines Algorithmus kann dann herausgefunden werden, wie der reelle DOM mit dem geringstmöglichen Aufwand angepasst werden kann. Dadurch können Änderungen gebündelt, gleichzeitig ausgeführt und somit die Performanz verbessert werden. (Zammetti, 2020, S. 47)

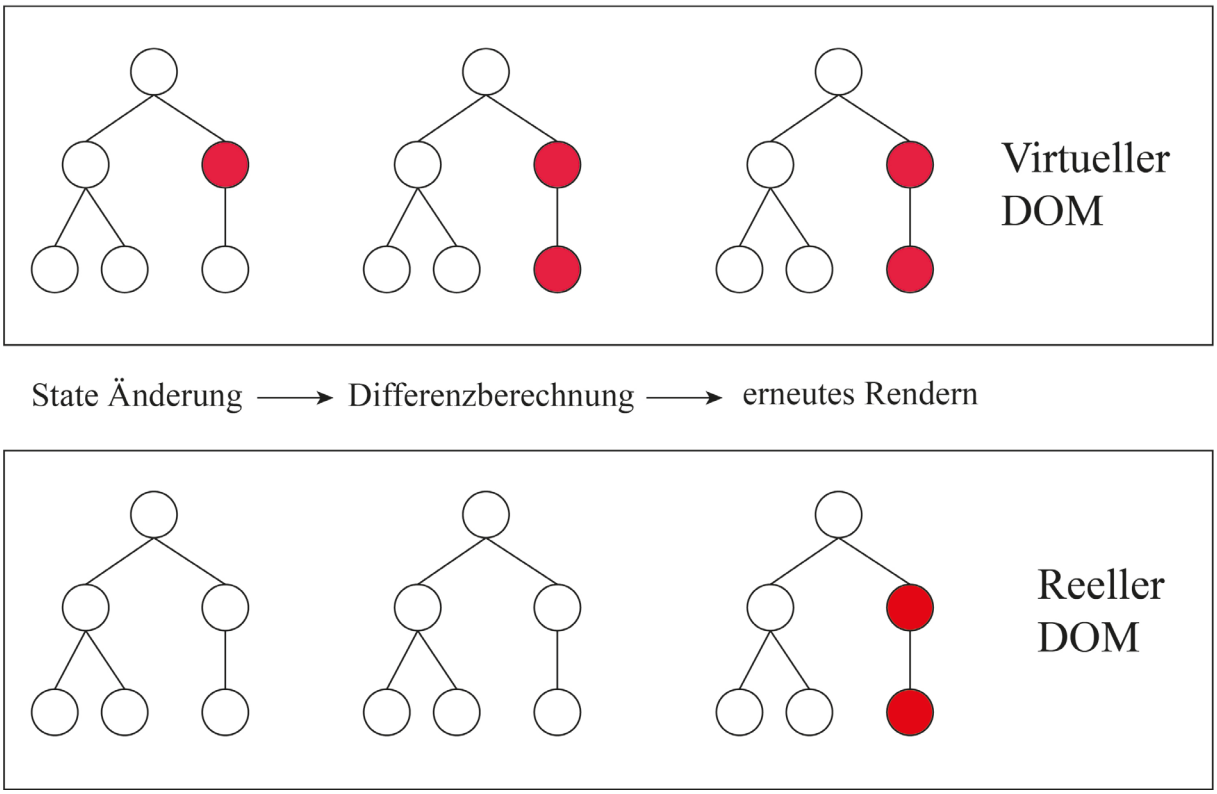

Abbildung 2: Virtueller DOM in Anlehnung an Eisenman, (2015)

Abbildung 2 verdeutlicht das Prinzip. Bei State-Änderungen wird zunächst der virtuelle DOM angepasst. Knotenpunkte mit verändertem State werden in Rot dargestellt. Dann werden die Differenzen zum ursprünglichen DOM berechnet und schließlich wird der reelle DOM in der günstigsten Weise aktualisiert.

4.1.4 Style

Das Styling kann wie gewohnt mittels Cascading Style Sheets (CSS) implementiert werden. Es besteht die Möglichkeit externe CSS-Files einzubinden, intern einen Style- Abschnitt mittels `style`-Tags zu deklarieren oder Inline-Styles zu verwenden. Bei letzterem wird das Styling direkt in JavaScript definiert. (Zammetti, 2020, S. 60-62)

4.1.5 JSX

Die Bedeutung des Kürzels JSX ist nicht eindeutig geklärt. Möglich wären die ausgeschriebenen Versionen „JavaScriptXML“ und „JavaScript Syntax eXtension“. JSX ist eine Syntaxerweiterung, die nicht zwingend mit React verknüpft werden muss. Es ist möglich, das UI eines Components zu definieren, ohne JSX zu verwenden. In der Regel wird allerdings von JSX Gebrauch gemacht, denn die JSX-Syntax ist deutlich aufgeräumter im Vergleich zu der Schreibweise, die auf JSX verzichtet. (Stefanov, 2017, S. 65-84)

Code, der JSX verwendet:

```
1. class Hello extends React.Component {
2.   render() {
3.     return <div>Hello {this.props.toWhat}</div>;
4.   }
5. }
6.
7. ReactDOM.render(
8.   <Hello toWhat="World" />,
9.   document.getElementById('root')
10.);
```

Code, der auf JSX verzichtet:

```
1. class Hello extends React.Component {
2.   render() {
3.     return React.createElement('div', null, `Hello ${this.props.toWhat}`);
4.   }
5. }
6.
7. ReactDOM.render(
8.   React.createElement(Hello, {toWhat: 'World'}, null),
9.   document.getElementById('root')
10.);
```

Damit der Browser JSX verstehen kann, muss es in gültiges JavaScript konvertiert werden. Dafür ist ein Transpiler, wie zum Beispiel Babel[5], unabdingbar. Auf den ersten Blick erkennt man die unumstrittene Ähnlichkeit zu HTML. Schaut man jedoch genauer hin, stößt man auf feine Unterschiede. So stehen die Bezeichnungen `class` und `for` beispielsweise nicht als Attributnamen zur Verfügung, denn sie sind Schlüsselbegriffe in JavaScript. An deren Stelle treten `className` und `htmlFor`. (Stefanov, 2017, S. 65-84)

4.2 Redux: eine Bibliothek für externes State-Management

Da es bei der Erstellung von Webanwendungen nicht ausschließlich um das UI geht, wurden, parallel zu React, Bibliotheken für weiterführende Bedürfnisse der Webentwicklung geschaffen. Im Zusammenspiel mit React können diese sogenannten Drittanbieter-Bibliotheken eingebunden und verwendet werden. Redux ist eine davon. (Hartmann & Zeigermann, 2020, S. 8)

Bei Redux handelt es sich um ein Tool für externes State-Management. Besonders bei größeren Anwendungen kann das State-Management kompliziert werden. Dann kann es Sinn machen,

[5] https://babeljs.io/

den State der Anwendung aus den Components auszulagern, an einen zentralen Ort, auf den alle Components Zugriff haben. Das ist vor allem bei Daten von Vorteil, die überall in der Anwendung benötigt werden. Ein Paradebeispiel hierfür ist der Login-Status eines Nutzers. (Hartmann & Zeigermann, 2020, S. 217)

Die ersten Ideen zu globalem State-Management stammen, genau wie bei React, von Facebook. Das Unternehmen hat eine Technologie entwickelt, die sich Flux[6] nennt. Die Ideen von Flux dienten verschiedenen Tools, unter anderem Redux, als Vorlage. In einigen Bereichen hat sich Redux von Flux losgelöst und unterscheidet sich teilweise deutlich von seinem Vorgänger. Mittlerweile hat sich Redux als die erste Wahl der React-Entwickler für externes State-Management herauskristallisiert. Redux kann aber auch mit anderen Bibliotheken oder Frameworks kombiniert werden. (Hartmann & Zeigermann, 2020, S. 217-219)

Redux funktioniert folgendermaßen (siehe Abbildung 3): Es gibt einen „Store“, in dem der State aus den Components gespeichert werden kann. Der Nutzer löst, durch die Interaktion mit der Anwendung, Action-Creator aus. Diese Funktionen produzieren, wie der Name bereits verrät jeweils eine Action, ein einfaches JavaScript Objekt mit einem `Type` und optional mit einem `Payload`. In der Action steht festgeschrieben, wie der State im Store modifiziert werden soll. Die Actions werden weitergeleitet an die Reducer. Man sagt auch, die Actions werden „dispatched“, was so viel bedeutet wie „versendet“. Mithilfe der Informationen aus den Actions erstellen die Reducer einen veränderten State, der dann im Store abgespeichert wird. (Hartmann & Zeigermann, 2020, S. 218- 219)

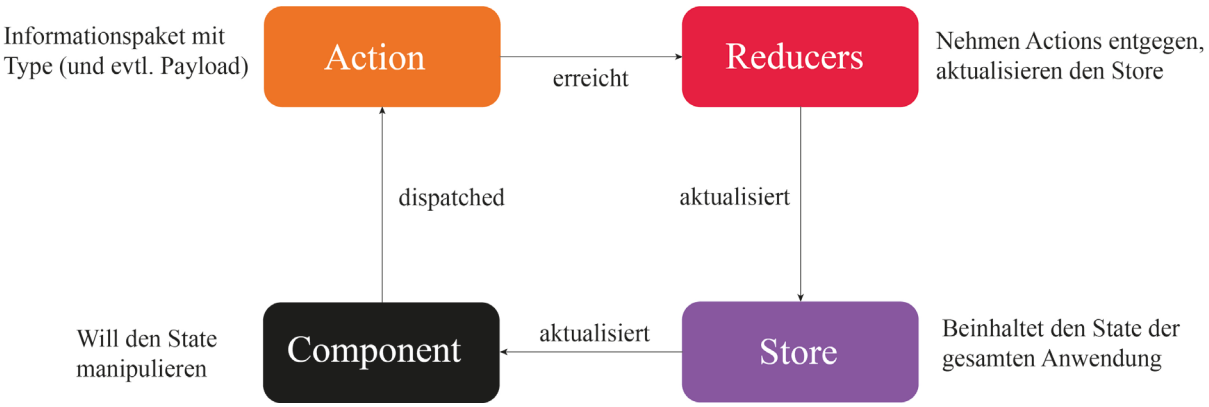

Abbildung 3: Redux in Anlehnung an Schwarzmüller, (2019)

[6] https://facebook.github.io/flux/

4.3 Steigende Beliebtheit gegenüber anderen Frameworks

Bis Ende 2013 gelang es React, anfängliche Skepsis auszuräumen. Ab diesem Zeitpunkt ging es steil bergauf. 2014 wurden die React Developer Tools veröffentlicht, eine Erweiterung der Chrome Developer Tools, mit denen ein einfaches Werkzeug für das Debuggen von React Apps geschaffen wurde. Viele Code-Editoren und integrierte Entwicklungsumgebungen (IDE) bieten seither native Unterstützung für React. Aus diesen Gründen wurde React zum Trendsetter und bei vielen bekannten Firmen beliebt. Zu diesen Firmen zählen unter anderem Netflix und Airbnb, die der Bibliothek nochmals zu einem Beliebtheitsschub verhalfen. React wächst stetig und ist heute eine der beliebtesten Bibliotheken, um clientseitige Webanwendungen zu bauen. (Zammetti, 2020, S. 43-46)

Es gibt diverse Gründe, warum sich Entwickler entscheiden, mit React zu arbeiten. Zum einen hat React den Ruf, einfach zu erlernen zu sein. Das liegt unter anderem an der stark begrenzten API (Hartmann & Zeigermann, 2020, S. 7). Darüber hinaus kann React unkompliziert in ein bereits bestehendes Projekt integriert werden, weil es selbst keine starre Architektur vorgibt. React stellt nur geringe Ansprüche an die Anwendung, in die es eingebunden wird (Hartmann & Zeigermann, 2020, S. 7). Zum dritten bietet die beträchtliche Größe des Unternehmens Facebook eine gewisse Sicherheit in Bezug auf die Weiterentwicklung und zukünftige Unterstützung der Bibliothek. (Zammetti, 2020)

Darüber hinaus wird bei Innovationen der API sehr rücksichtsvoll vorgegangen. Änderungen werden entsprechend weit im Voraus angekündigt und solange keine Alternative vorliegt, werden APIs nicht stillgelegt. Updates erfolgen kontinuierlich, jedoch immer abwärtskompatibel. So können sich die Anwender darauf verlassen, dass ihre Projekte auch nach der Einführung moderner Features noch funktionstüchtig bleiben. Die größten Konkurrenten für React sind Angular und View.

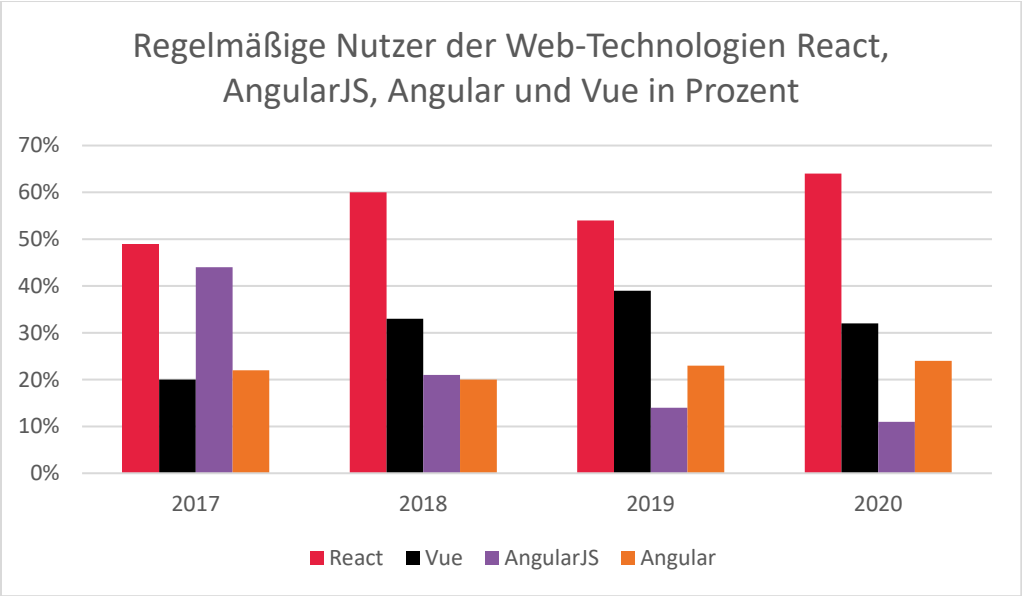

Abbildung 4: Regelmäßige Nutzer der Web-Technologien React, AngularJS, Angular und Vue in Prozent in Anlehnung an JetBrains, (2017, 2018, 2019, 2020)

Die Entwicklerumfragen von JetBrains aus den Jahren 2017 bis 2020 (siehe Abbildung 4) verdeutlichen, dass React im Vergleich zu Vue und Angular das beliebteste Web-Framework der vergangenen vier Jahre ist. In jedem Jahr gaben die meisten Befragten an, React regelmäßig zu nutzen. Nachfolgend werden die beiden React-Konkurrenten Angular und Vue kurz vorgestellt. Auch der Unterschied zwischen AngularJS und Angular wird im Folgenden erläutert.

4.3.1 Angular

Das JavaScript-Framework Angular[7] wurde von Google entwickelt und ist 2010 erschienen. In den ersten Jahren erfreute sich Angular großer Beliebtheit. Zur Aufstiegszeit von React unterlief Google allerdings ein entscheidender Fehler, durch den einige Nutzer Angular den Rücken kehrten und sich React zuwendeten. Der Übergang von Angular Version 1 (auch als AngularJS bezeichnet) auf Version 2 verlief nicht ganz reibungslos. Die neue Version war nicht kompatibel mit der älteren und hatte für einige Nutzer erhebliche Schwierigkeiten zur Folge. Angular nutzt die Programmiersprache TypeScript[8]. (Hartmann & Zeigermann, 2020, S. 15-16; Zammetti, 2020, S. 63)

[7] https://angular.io/
[8] https://www.typescriptlang.org/

4.3.2 Vue

Vue[9] ist ein weiteres JavaScript Framework, mit dem SPAs realisiert werden. Es wurde 2014 herausgebracht, nur ein Jahr später als React. Es dauerte nicht lange, bis Entwickler an der Open-Source-Software Gefallen fanden. Ebenso wie React arbeitet Vue mit Komponenten. Einige Tools, die bei React über externe Bibliotheken eingebunden werden müssen, sind in View direkt integriert. So besitzt Vue beispielsweise seinen eigenen internen Router. Im Vergleich zu Angular liefert Vue eine kleinere API. Auch die Anforderungen an die Struktur der Anwendung sind lockerer als bei Angular. (Hartmann & Zeigermann, 2020, S. 16)

5 Unit Tests

Auch das Testen einer Anwendung nimmt Einfluss auf deren Wartbarkeit und wird deshalb in dieser Arbeit behandelt. Es gibt verschiedene Arten von Tests. Eine der beliebtesten sind die sogenannten Unit Tests. Diese sind, in Anbetracht von Zeitaufwand und Nutzen, verhältnismäßig ergiebig. (Hunt & Thomas, 2004, S.1) Laut Martin (2009, S. 162) sind Unit Tests unter anderem Garant für die Wartbarkeit des Codes. Diese Behauptung begründet er damit, dass durch Unit Tests Hemmungen entfallen, den Code zu verändern, weil der Entwickler sich auf die Absicherung der Funktionalität verlassen kann.

> „Ein Unit Test ist ein Stück Quelltext, das ein Entwickler nur zum Testen eines sehr kleinen und klar umrissenen Teiles der Funktionalität geschrieben hat.“ (Hunt & Thomas, 2004, S. 3)

Hunt & Thomas (2004) konzentrieren sich zwar auf die Programmiersprache Java. Die Autoren machen allerdings deutlich, dass die von ihnen beschriebene Methodik des Testens auch in anderen Programmiersprachen anwendbar ist.

Unit Tests zu schreiben, fällt in den Zuständigkeitsbereich des Programmierers, nicht des Vorgesetzten. Setzt der Programmierer Unit Tests gekonnt ein, kann er sich oftmals Ärger ersparen und seine Arbeit dadurch vereinfachen. Unit Tests überprüfen, ob der Code genau das bewirkt, was der Entwickler im Sinn hatte. Beim Programmieren kann man sich nicht auf Vermutungen verlassen. Auch das reine Compilieren des Codes reicht nicht aus, denn es bedeutet lediglich, dass die Syntax eingehalten wurde. Ein Compiler kann aber nicht überprüfen, ob der Code das macht, was er soll. Es muss ein handfester Beweis über die Funktionsfähigkeit des Codes geliefert werden, in Form eines positiven Testergebnisses. Eine untergeordnete Aufgabe von Unit

[9] https://vuejs.org/

Tests ist es, den Teamkollegen eine Art Bericht über das Verhalten der Anwendung an die Hand zu geben. Im Gegensatz zu einem schriftlichen Protokoll hat diese Art der Dokumentation einen entscheidenden Vorteil: Durch das beständige Durchführen und gelegentliche Anpassen der Tests, bleibt sie immer aktuell. (Hunt & Thomas, 2004, S. 1-7)

Tests zu schreiben zählt meist nicht zur Lieblingsaufgabe eines Programmierers und wird deshalb zuweilen vernachlässigt. Ein beliebtes Argument, das in diesem Zusammenhang hervorgebracht wird, ist der erhöhte Bedarf an Zeit, der für das Testen eingeplant werden muss. Während das Schreiben von Tests Zeit kostet, kann andererseits Zeit für die Fehlersuche eingespart werden. Im Normalfall wird der zeitliche Mehraufwand des Testens im Laufe eines Projektes mehr als ausgeglichen. Auch das Durchlaufen der geschriebenen Tests geht im Regelfall zügig von statten. (Hunt & Thomas, 2004, S. 9-13)

Beim Ausführen der Tests ist folgendes zu beachten: Am besten führt man nicht nur den neuesten Test aus, sondern immer alle auf einmal. Dadurch kann man dem Entstehen von Fehlern in anderen als dem aktuell betrachteten Bereich entgegenwirken. (Hunt & Thomas, 2004, S. 8)

5.1 Eigenschaften guter Tests

Für die Eigenschaften guter Tests gibt es ein Merkwort: A TRIP. Dieses Akronym steht für die fünf Kriterien „Automatic“, „Thorough“, „Repeatable“, „Independet“ und „Professional“, mit denen identifiziert werden kann, ob es sich um einen guten Test handelt. (Hunt & Thomas, 2004, S. 89)

Erstens sollen Tests automatisch ausgeführt werden oder falls nicht möglich, zumindest einfach gestartet werden können. Das Testergebnis muss visuell eindeutig sein, sodass der Entwickler unwillkürlich sehen kann, ob der Test erfolgreich absolviert wurde. (Hunt & Thomas, 2004, S. 90-91)

Darüber hinaus spielt die Gründlichkeit der Tests eine wichtige Rolle. Dieser Aspekt ist eher subjektiv und steht in engem Zusammenhang mit den Ansprüchen an die Software. Mit Gründlichkeit ist gemeint, wie viel Code durch die Tests abgesichert wird. Man spricht in diesem Zusammenhang auch von „Quelltextabdeckung“. Je gründlicher die Tests, desto mehr wird getestet und desto größer ist die Abdeckung. (Hunt & Thomas, 2004, S. 91-93)

Zum dritten sind gute Tests wiederholbar. Sie müssen beliebig oft ausgeführt werden können und unter denselben Umständen immer zum gleichen Resultat führen. Die Entwicklungsumgebung sowie die Reihenfolge, in der die Tests ausgeführt werden, dürfen keinen Einfluss auf das Ergebnis nehmen. (Hunt & Thomas, 2004, S. 93)

Außerdem sind gute Tests unabhängig. Sie beschränken sich auf einen überschaubaren Bereich und stützen sich nicht auf die Ergebnisse eines zuvor ausgeführten Tests. Der Durchlauf der

Tests soll in jeder denkbaren Abfolge möglich sein, ohne die Testergebnisse zu verfälschen. (Hunt & Thomas, 2004, S. 94)

Schließlich sind gute Tests professionell. Darunter versteht man, dass ausschließlich riskante Stellen durch Tests überprüft werden sollen, denn Tests schreiben ist kein Selbstzweck. Außerdem soll der Code der Tests dem Quellcode der Anwendung qualitativ in nichts nachstehen. Auch beim Schreiben der Tests müssen elementare Prinzipien wie beispielsweise das DRY-Prinzip berücksichtigt werden. Ferner kann es auch bei Tests Sinn machen, Refactoring einzusetzen, um die Struktur des Codes qualitativ auf das nächste Level zu heben. (Hunt & Thomas, 2004, S. 95-96)

5.2 Tests in React

Wenn es um das Testen von React Anwendungen geht, ist Jest[10], ein von Facebook entwickeltes Testframework, die beliebteste Wahl. Andere Optionen sind Jasmine[11] und Mocha[12]. Zwischen diesen drei Test-Werkzeugen sind Ähnlichkeiten vorhanden, wodurch ein Wechsel zu einem anderen Tool in der Regel einfach möglich ist, sofern der Entwickler bereits mit einem der Tools vertraut ist. (Springer, 2020, S. 248-249)

In Kombination mit Jest kann die Bibliothek „React Testing Library“ (RTL) verwendet werden, die das Testen von React-Anwendungen nochmals deutlich vereinfacht (Springer, 2020, S. 274). Beide Tools werden im Folgenden vorgestellt.

5.2.1 Jest

Im Falle von Jest wird nicht auf eine richtige Browserumgebung gesetzt, in dem die Tests ausgeführt werden können, sondern auf jsdom[13]. Hierbei handelt es sich um einen Ersatz für die Browserumgebung, welche einige, Differenzen im Vergleich zu einem echten Browser aufweist: mit jsdom wird eine aufwendige Instandhaltung der Testumgebung redundant und Tests laufen schneller ab, weil zusätzliche Daten entfallen, die für einen echten Browsers benötigt werden. (Springer, 2020, S. 248-249)

Unit Tests mit Jest konzentrieren sich auch auf einen überschaubaren, klar abgegrenzten Teil der Funktionalität. Im Kontext von React beziehen sie sich auf einen einzelnen Component oder eine Funktion. Die Testdatei kann entweder im gleichen Ordner wie der Component abgespeichert werden oder in einem gesonderten Ordner für Tests. (Springer, 2020, S. 250) Als

[10] https://jestjs.io/
[11] https://jasmine.github.io/
[12] https://mochajs.org/
[13] https://github.com/jsdom/jsdom

Dateiendung wird `test.js` an den Dateinamen angehängt. So sind sie auf einen Blick von regulären Dateien differenzierbar.
Um Tests zu schreiben, kann entweder die `test`- oder die `it`-Funktion verwendet werden. Sie erfüllen denselben Zweck und sind austauschbar. Die Basisfunktion, ganz gleich welche von beiden ausgewählt wird, erwartet zwei Argumente: zum einen eine kurze Aussage über den Testgegenstand, die immer mit ausgegeben wird und im Fehlerfall Aufschluss darüber geben soll, an welcher Stelle das Problem aufgetreten ist, zum anderen eine Callback-Funktion, die die eigentliche Logik des Tests beinhaltet. Diese richtet sich nach dem Triple-A-Schema. (Springer, 2020, S. 252-254)

Abbildung 5: Triple-A-Schema

Die drei As stehen für „Arrange“, „Act“ und „Assert“ (siehe Abbildung 5). Sie stellen drei wichtige Stationen bei der Zusammenstellung eines Tests dar. Zunächst müssen Vorbereitungen der Umgebung getroffen werden. Im Konkreten handelt es sich darum, die Interaktion zwischen Test und Testgegenstand zu ermöglichen, indem beispielsweise der zu testende Component dupliziert beziehungsweise zur Verfügung gestellt wird. Danach wird das Ereignis ausgeführt, das getestet werden soll. Im letzten Schritt werden Erwartungen und Ergebnis verglichen. (Springer, 2020, S. 252-254) Ein Test, bei dem nach dem Triple-A-Schema vorgegangen wird, kann wie folgt aussehen:

```
1. import Calculator from './calculator';
2.
3. it('should add 1 and 1 and return 2', () => {
4. const calculator = newCalculator();
5. const result = calculator.add(1, 1);
6. expect(result).toBe(2);
7. });
```

Jest bietet verschiedene „Matcher“ an, von denen man den passenden auswählen und für den Abgleich einsetzen kann. Darüber hinaus besteht die Möglichkeit „Matcher“ ins Gegenteil zu kehren oder maßgeschneiderte „Matcher“ selbst zu schreiben. Außerdem können mehrere Tests zu einer Testsuite zusammengefügt werden, die je nach Bedarf als Gruppe ausgelassen oder ausgeführt werden kann. (Springer, 2020, S. 254-255)

5.2.2 React Testing Library[14]

RTL ist eine Bibliothek, die das Testen von React-Components noch zusätzlich vereinfachen soll. Sie bietet eine praktische `render`-Funktion an, sowie diverse Funktionen, die das Lokalisieren von Elementen erleichtern. Darüber hinaus ist es mit der RTL möglich, Benutzerinteraktionen zu simulieren. (Springer, 2020, S. 274-278)

RTL stellt für das Testen einen virtuellen DOM zur Verfügung. Die Bibliothek fördert eine bestimmte Art zu testen: nämlich das Testen aus Nutzer Perspektive. Beim Testen mit RTL geht es vornehmlich um das Testen der Funktionalität und nicht um das Überprüfen der Implementierung. Deshalb verzichtet die Bibliothek darauf, Zugriff auf State oder Props der Components zu gewährleisten. Getestet wird ausschließlich mithilfe der DOM-Elemente. (Norbu, 2020)

6 Praktische Anwendung: Veranstaltungseditor des Monte-Tools

An Montessorischulen wird von den Eltern der Schüler erwartet, sich für die Schulgemeinschaft zu engagieren. Bei gemeinschaftlichen Aktionen oder schulischen Veranstaltungen gibt es immer genug Arbeit, die von freiwilligen Elternteilen übernommen werden soll. Vor noch nicht langer Zeit wurde dieses Konzept mithilfe von physischen Aufgaben-Listen umgesetzt, in die sich die Eltern eintragen konnten. Diese konventionelle Organisationsform zog einige Nachteile nach sich, die in den meisten Fällen bei Papier-lastiger Büroarbeit eintreten. Zettel gehen verloren oder werden nicht auf Anhieb gefunden. Die Kontrolle darüber, welche Familie schon genügend Aufgaben übernommen hat und welche noch nicht, ist mithilfe eines Ordners voller Listen auch eher mühsam und zeitintensiv.

Diesem Problem hat sich die Firma Evidentmedia, eine Agentur für Kommunikation und Visualisierung, angenommen. Sie entwickelte die Webanwendung Monte-Tool[15]. Mittlerweile wird die Software von verschiedenen Montessorischulen in Bayern genutzt.

6.1 Funktionalität des Veranstaltungseditors

Die Webanwendung beinhaltet einen Veranstaltungseditor, der das Herzstück des Tools bildet. Die Beschreibung der Funktionalität wird sich im Folgenden auf diesen elementaren Teil der Anwendung beschränken. Außerhalb des Veranstaltungseditors gibt es weitere Funktionen, die den Rahmen der Arbeit an dieser Stelle überschreiten würden.

[14] https://testing-library.com/docs/react-testing-library/intro/
[15] https://www.monte-tool.de

Wie der Name bereits verrät, spielen Veranstaltungen eine zentrale Rolle. Hierarchisch betrachtet steht die Veranstaltung an oberster Stelle. Ihr untergeordnet sind Aufgaben, die wiederum Dienste beinhalten.

Mit dem Veranstaltungseditor kann ein Administrator Veranstaltungen anlegen. Beispielsweise besteht die Möglichkeit für das geplante Sommerfest oder den anstehenden Weihnachtsbasar ein Event zu erstellen. Zusammen mit einer Veranstaltung können Aufgaben angelegt werden, die bei dem jeweiligen Event anfallen. „Brezen schmieren“ könnte eine Aufgabe beim Sommerfest darstellen. Zu jeder Aufgabe gehören ein oder mehrere Dienste, die von den Familien übernommen werden müssen. Gibt es vier Dienste für die Aufgabe „Brezen schmieren“, werden vier Freiwillige benötigt, um diese Aufgabe auszuführen. Die Eltern können diese Dienste bequem von zu Hause buchen. Wurden die Dienste zufriedenstellend erledigt, obliegt es dem Administrator, die ausgeführten Dienste zu bestätigen. Die aufgewendeten Arbeitsstunden werden dem entsprechenden Nutzerkonto gutgeschrieben. Man sieht also, der Veranstaltungseditor ist viel mehr als ein herkömmlicher Terminkalender. Vor allem die Buchungs- und Bestätigungsoption des Veranstaltungseditors ist außergewöhnlich und maßgeschneidert auf die Bedürfnisse der Montessorischulen.

Um eine genauere Vorstellung von einzelnen Vorgängen bei der Arbeit mit dem Veranstaltungseditor zu bekommen, werden diese im Folgenden näher dargestellt.

6.1.1 Anlegen einer Veranstaltung

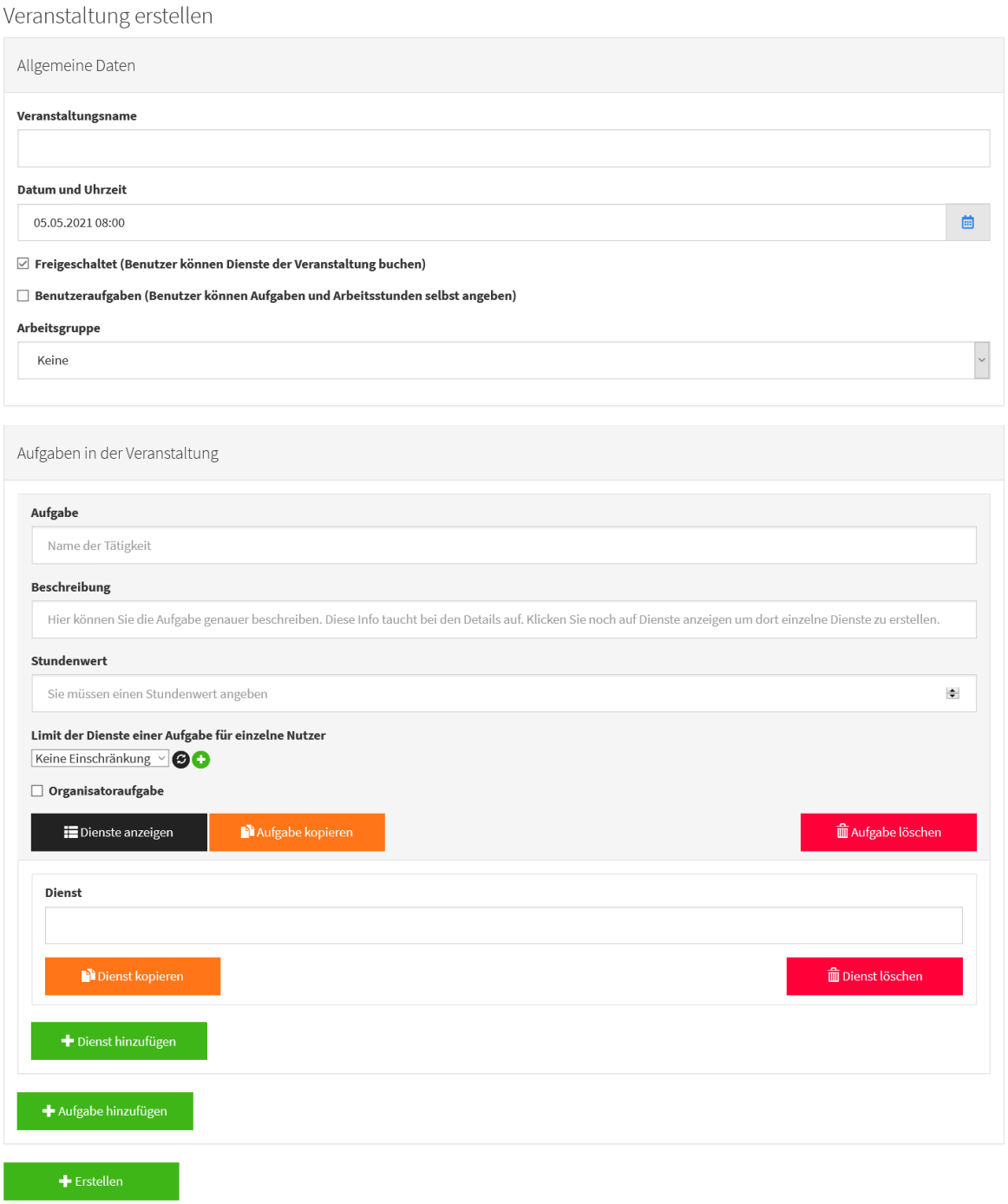

Abbildung 6: Benutzeroberfläche Veranstaltung erstellen

Um eine Veranstaltung anzulegen, muss ein Formular mit entsprechenden Daten befüllt werden (siehe Abbildung 6). Zunächst besteht die Option, allgemeine Daten zur Veranstaltung in die Formularmaske einzutragen. Zu diesen grundlegenden Daten zählen der Veranstaltungsname, sowie das Datum und die Uhrzeit der Veranstaltung. Des Weiteren gibt es die Möglichkeit, die Dienste der Veranstaltung für die Buchung der Nutzer freizuschalten oder vorerst zu sperren.

Darüber hinaus können die Aufgaben zu „Benutzeraufgaben“ konvertiert werden. Das bedeutet, dass fortan nicht mehr der Administrator den Stundenwert einer Aufgabe festlegt, sondern, dass dieses Privileg nun den Eltern zu Teil wird. Falls eine Veranstaltung nur die Personen einer bestimmten Arbeitsgruppe betrifft, kann diese Arbeitsgruppe ausgewählt werden. Daraufhin wird der Personenkreis, für den das Dienste Buchen freigeschaltet ist, auf die Mitglieder der Arbeitsgruppe beschränkt.
Für das Anlegen von Aufgaben und Diensten gibt es ebenfalls Formularfelder. Für die Aufgabe kann ein Name, eine Beschreibung und ein Stundenwert vergeben werden. Falls Limitierungen existieren, die verhindern sollen, dass Nutzer zu viele gleichartige Dienste für sich beanspruchen, können diese ausgewählt werden. Sind andernfalls noch keine vorhanden oder wären zusätzliche Limitierungen wünschenswert, können diese erstellt und zu den Auswahlmöglichkeiten hinzugefügt werden. Des Weiteren kann eine Aufgabe als „Organisatoraufgabe“ ausgezeichnet werden. Dem Nutzer, der diese Aufgabe bucht, wird die Befugnis übertragen, über das Dashboard Dienste anderer Nutzer zu bestätigen.
Um die Übersichtlichkeit der Seite zu wahren, reicht ein Mausklick und die Dienste werden ein- bzw. ausgeblendet. Mithilfe der Buttons können Dienste und Aufgaben schnell und einfach kopiert, hinzugefügt oder gelöscht werden.
Es ist Vorgabe, dass jede Veranstaltung mindestens eine Aufgabe beinhalten muss. Jede Aufgabe wiederum, muss mindestens einen Dienst beinhalten. Für die Anzahl der Aufgaben oder der Dienste einer Aufgabe gibt es kein Limit. Alle Daten, die beim Erstellungsprozess einer Veranstaltung in das Formular eingetragen werden, werden nach Betätigung des Veranstaltung-Erstellen-Buttons an einen Server geschickt und in einer relationalen Datenbank gespeichert.

6.1.2 Editieren einer Veranstaltung

Darüber hinaus können die Datensätze einer bereits angelegten Veranstaltung nach Belieben bearbeitet und somit auch zu einem späteren Zeitpunkt verändert oder um zusätzliche Informationen ergänzt werden. Das Eingabeformular ist in diesem Fall mit den zugehörigen Daten der Veranstaltung bereits vorausgefüllt.

6.1.3 Buchungsprozess

Die Eltern können sich selbstständig von zu Hause aus für Dienste eintragen. Es besteht darüber hinaus die Möglichkeit als Administrator Dienste einem Profil zuzuweisen. Hierbei kann der Administrator aus einer Liste aller in Frage kommender Nutzer, den gewünschten Namen anwählen. Tritt dabei ein Versehen auf, kann die Buchung wieder gelöscht und ein anderer Nutzer

kann ausgewählt werden. Wurde der Dienst zufriedenstellend erledigt, wird der Dienst vom Administrator bestätigt.

6.1.4 Änderungsanfragen

Außerhalb des Veranstaltungseditors können Eltern mit dem Monte-Tool Änderungsanfragen stellen. Wenn Sie beispielsweise länger oder kürzer als geplant für einen Dienst benötigt haben, können sie diese Abweichung zusammen mit einer Begründung einreichen. Der Administrator bearbeitet die Anfragen. Er kann sie bestätigen oder ablehnen.

Es ist wichtig, die Änderungsanfragen oder „change requests“, wie sie auch genannt werden, im Zusammenhang mit dem Event-Editor anzusprechen, denn dort müssen Informationen zu bestehenden Änderungsanfragen angezeigt werden. Zum einen wird ein allgemeiner Hinweis bei der übergeordneten Aufgabe eingeblendet (siehe Abbildung 7), zum anderen ein etwas detaillierterer Hinweis direkt unter der betroffenen Buchung (siehe Abbildung 8).

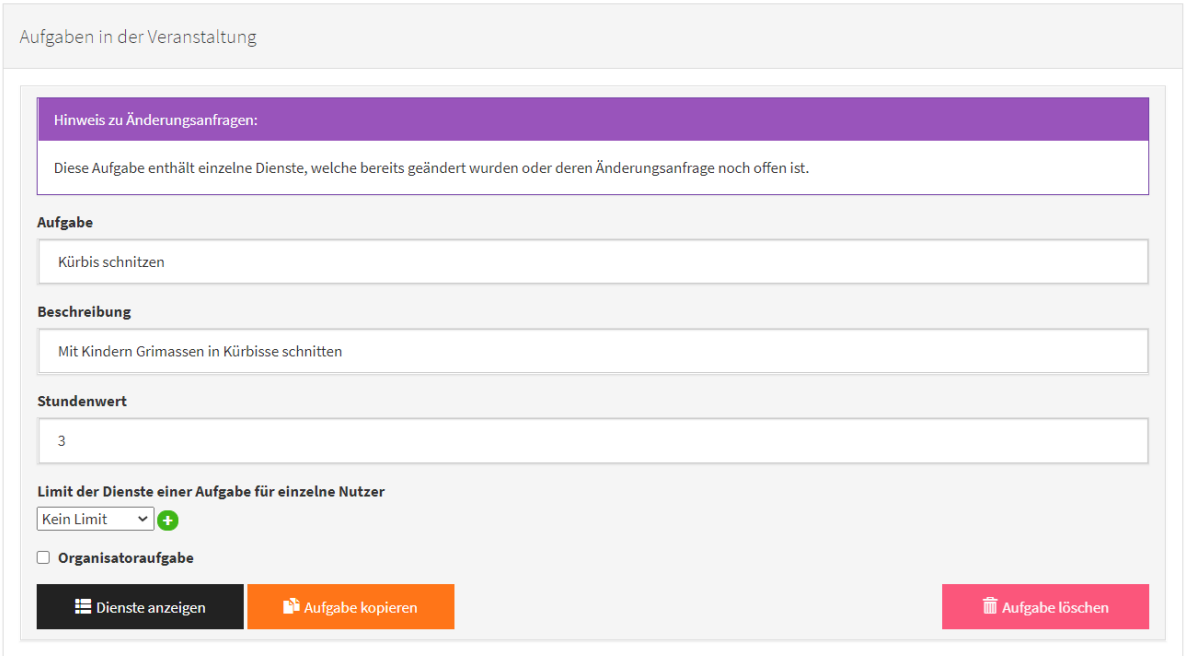

Abbildung 7: Hinweis zu Änderungsanfrage bei übergeordneter Aufgabe

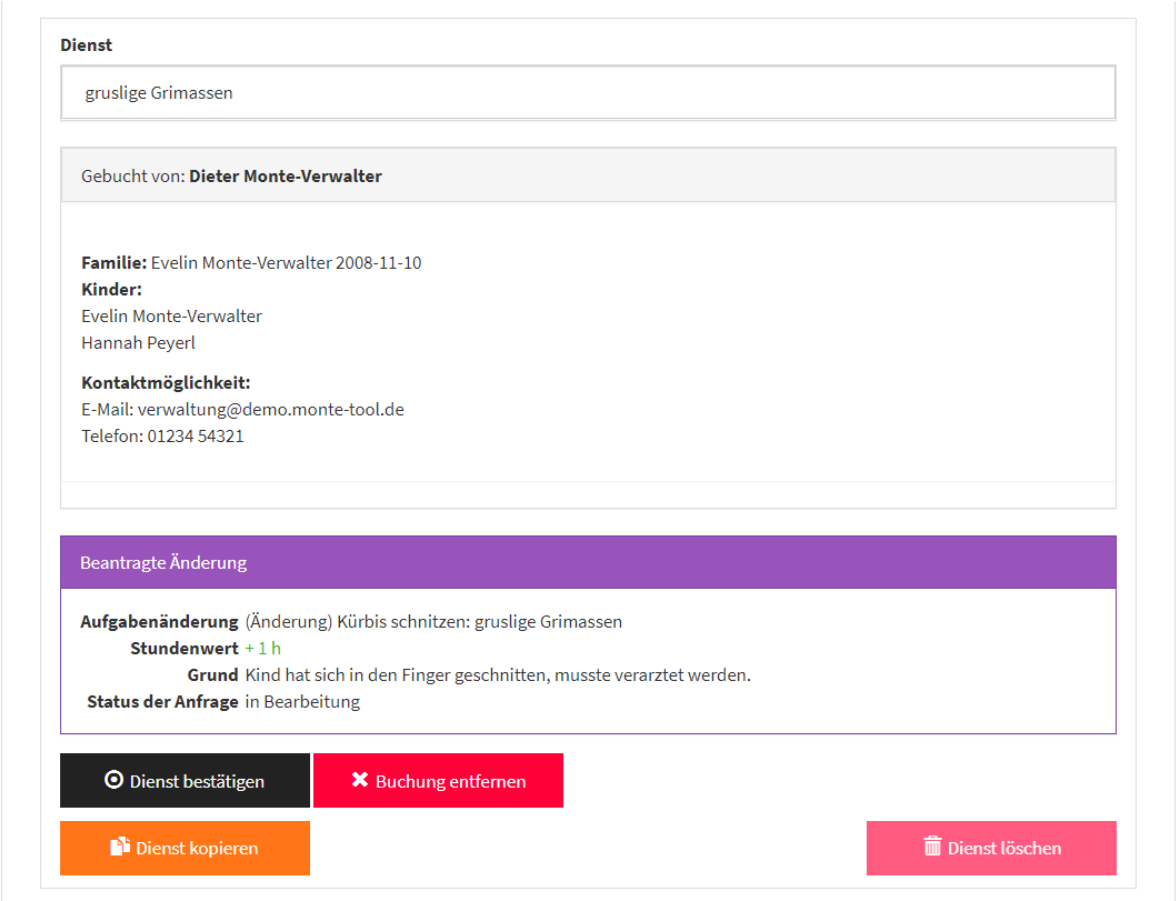

Abbildung 8: Hinweis zu Änderungsanfrage direkt unter betroffener Buchung

6.2 Verwendete Tools

Die Webanwendung Monte-Tool wurde mithilfe des Frameworks CakePHP[16] entwickelt. Die Daten werden aus einer angebundenen MySQL-Datenbank bezogen. Weitere Technologien, die bei der Programmierung genutzt wurden, sind JQuery[17], eine JavaScript Bibliothek und Bootstrap, ein Frontend-CSS-Framework. Nachfolgend wird ein tieferer Einblick in diese Technologien gewährt.

6.2.1 CakePHP

CakePHP ist ein Framework, das die Entwicklung von Webanwendungen mit der Programmiersprache PHP erleichtert. Es wurde dem beliebten Framework Ruby on Rails[18] nachempfunden. (Ammelburger & Scherer, 2009, S. 1-2)

[16] https://cakephp.org/
[17] https://jquery.com/
[18] https://rubyonrails.org/

> „Ein Framework, frei übersetzt eine ‚Rahmenstruktur' ist im Prinzip eine Art Code-Gerüst, das für die Entwicklung von Applikationen verwendet wird. Ein Framework selbst ist kein fertiges Programm, sondern stellt vielmehr einer Art Grundlage dar, auf der ein neues Programm entwickelt wird. In der Regel wird durch ein Framework ein bestimmter Stil oder ein bestimmter Aufbau vorgegeben, nach dem sich der Entwickler richten sollte." (Ammelburger & Scherer, 2009, S. 2)

Bei CakePHP äußert sich diese Vorgabe durch das Model-View-Controller (MVC)-Schema. Dieses Design-Pattern wird nicht ausschließlich von CakePHP verwendet, sondern findet sich auch in anderen Frameworks wieder. Durch die MVC-Architektur erfolgt eine klare Trennung der Funktionen einer Anwendung. Die Aufspaltung erfolgt nach den drei Bereichen Daten (Model), Ansicht (View) und Steuerung (Controller). Das Model kümmert sich um das Beschaffen der Daten, beispielsweise aus einer relationalen Datenbank. Darüber hinaus führt es Änderungen an den Daten durch. Die View ist zuständig für das Anzeigen der Daten. Der Controller beinhaltet die Logik der Applikation. Er ist Bindeglied und Informationsübermittler zwischen Model und View und nimmt Benutzerinteraktionen entgegen. Model und View hingegen tangieren sich kaum. Der Sinn hinter dieser Aufteilung ist es, die Applikation in selbstverantwortliche Abschnitte aufzugliedern, bei der jeder für sich modifiziert werden kann. (Ammelburger & Scherer, 2009, S. 6-10)

6.2.2 MySQL, JQuery und Bootstrap

Zur Verwaltung der Daten des Monte-Tools wird eine MySQL-Datenbank verwendet, die mithilfe der Open-Source-Software phpMyAdmin[19] bearbeitet werden kann. MySQL kann in die Kategorie relationaler Datenbanksysteme eingeordnet werden. Bei diesen werden die Daten in Tabellenform gespeichert, wobei jede Tabellenzeile eindeutig unterschieden werden kann. Um Operationen auf den Datensätzen auszuüben, wird die Abfragesprache Structured Query Language (SQL) verwendet. (Reimers & Thies, 2010, S. 186-190)

JQuery wird beim Monte-Tool eingesetzt, um einen Teil der Benutzerlogik zu implementieren. Im Vergleich zu Vanilla JavaScript kann mit JQuery die Länge des zu schreibenden Quellcodes verkürzt und Operationen, wie beispielsweise das Reagieren auf Events, können vereinfacht werden.

Für das Styling wird auf Bootstrap zurückgegriffen. Bootstrap liefert vorgefertigte Rastersysteme, an die man das Layout der Seite ausrichten kann und die sich hervorragend für einen

[19] https://www.phpmyadmin.net/

responsiven Aufbau der Seite eignen. Außerdem stellt Bootstrap CSS-Klassen bereit, die für ein attraktives Aussehen von beispielsweise Buttons eingesetzt werden können.

6.2.3 Integration von React

Auch wenn React meistens mit SPAs in Verbindung gebracht wird, kann die JavaScript Bibliothek auch in eine bereits bestehende Anwendung, wie beispielsweise das Monte-Tool integriert werden. React wird dann lediglich für den Veranstaltungseditor verwendet, während der Rest der Anwendung nach wie vor mit der Programmiersprache PHP und dem Framework CakePHP realisiert wird. Abbildung 9 soll verdeutlichen, an welcher Stelle React in die Anwendung integriert wird und mit welchen Teilen der Anwendung React in Verbindung steht.

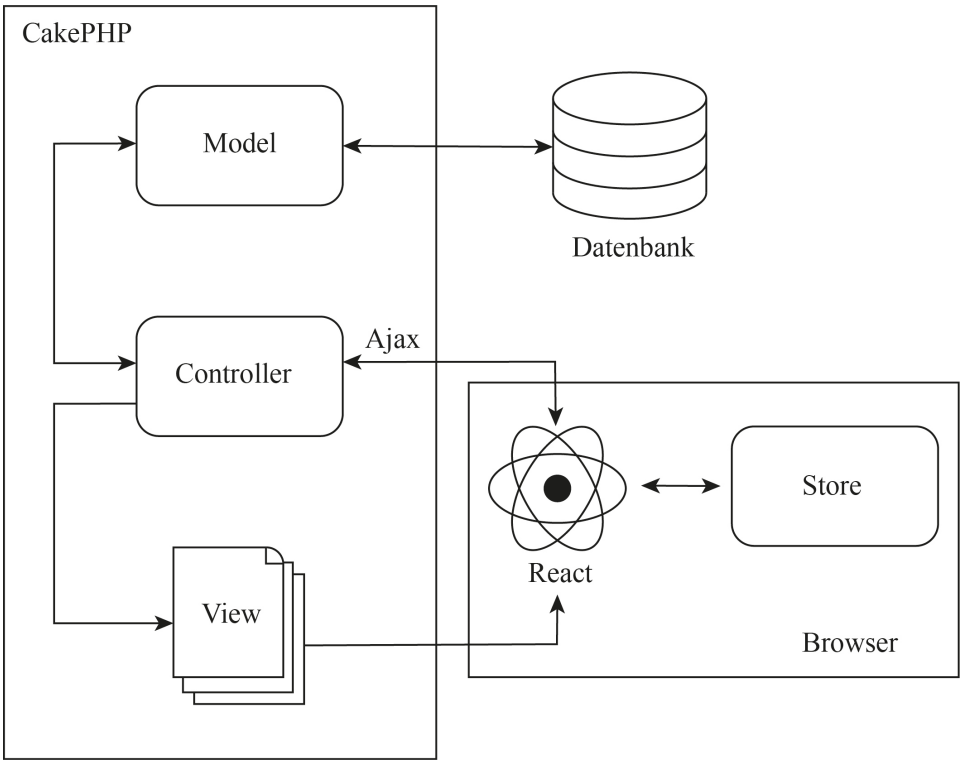

Abbildung 9: Integration von React in das Monte-Tool

React wird zwischen der View und dem Controller von CakePHP in das Monte-Tool integriert. Im Controller wird eine Funktion aufgesetzt, die die View mit allen benötigten Daten aus dem Model versorgt. In der View von CakePHP wird React eingebunden. React läuft clientseitig. Das bedeutet, dass die Seite, die der Benutzer sehen kann, im Browser generiert wird und nicht vom Server an den Browser geschickt werden muss.

React verwaltet alle Daten mithilfe des Stores. Das stellt einen entscheidenden Vorteil im Gegensatz zur Vorgängerversion dar, bei der benötigte Daten in der HTML-Struktur des Formulars mühsam gesucht und selektiert werden mussten. Benötigt React weitere Informationen aus

der Datenbank, beispielsweise Nutzerdaten, ruft React eine passende Funktion im Controller auf, die die Daten beschaffen kann. Auch für das Abspeichern von Nutzereingaben interagiert React mit dem Controller. Im Unterschied zu vorher wird kein Formular an den Server geschickt. Stattdessen wird mittels Ajax eine Funktion im Controller aufgerufen, die die Daten in die Datenbank speichert. Ein Vorteil dieser Vorgehensweise ist, dass bei einem Ausfall des Servers die ursprüngliche Seite noch angezeigt werden kann. Würde ein Formular abgeschickt werden und der Server lieferte keine Antwort, wäre die alte Seite scheinbar verloren. Mittlerweile sind Browser allerdings so intelligent, dass mittels der „Zurück"-Funktion der linken Pfeiltaste des Browsers die Seite trotzdem wiederhergestellt werden kann.

6.3 Probleme der bisherigen Softwarestruktur

Das zentrale Problem besteht darin, dass der Code des alten Veranstaltungseditors mittlerweile maximal unübersichtlich ist. Es gibt lediglich zwei große Dateien, die als Templates fungieren und das UI beschreiben: `add.ctp` und `edit.ctp`. Dadurch, dass es keine Untergliederung des UIs gibt, haben diese Dateien eine stattliche Länge. Zusätzlich wurde eine Datei angelegt, um mithilfe von JQuery Operationen auf ausgewählten DOM-Elementen auszuführen. In dieser Datei findet keine thematische Untergliederung statt. Alle Funktionen reihen sich aneinander. Aufgrund der langen Files kostet es mehr Zeit, bis man den entsprechenden Code-Abschnitt gefunden hat, nach dem man sucht. So wird die Wartung der Anwendung komplizierter und anstrengender. Mit dem Konzept von React, das UI in modularen, schlanken Components zu organisieren, soll für mehr Übersichtlichkeit und eine bessere Wartbarkeit gesorgt werden.
Ein weiteres Problem beim alten Event-Editor ist, dass die Anzeige der Daten und die konzeptionellen Daten nicht getrennt voneinander behandelt werden. Änderungen an einem Element werden aufgrund von dessen Position im DOM oder aufgrund von CSS-Eigenschaften wie Klassen ausgeführt. Das Problem, das dadurch entsteht, ist folgendes: Änderungen des DOM oder des CSS könnten die Funktionalität der Anwendung gefährden.
Als Beispiel dient folgendes Code-Fragement aus der Datei `edit.ctp`:

```
1. function admSetServiceDone(elem) {
2.     var workID = $(elem).attr('data-workid');
3.     if($(elem).hasClass('btn-default')) {
4.             $.getJSON(admSetDoneUrl+"/"+workID+"/1", function(data){
5.                 if(data.error == false) {
6.                     $(elem).removeClass('btn-default').addClass('btn-suc-
   cess');
7.                     $(elem).html('<span class="glyphicon glyphicon-ok-
   circle"></span> Dienst bestätigt');
8.                 } else {
9.                     alert("Leider ist ein Fehler aufgetreten. Versuchen Sie
   es zu einem anderen Zeitpunkt noch einmal.");
10.                }
11.            });
12. …
```

Mittels der Funktion `admSetServiceDone` soll eine Buchung bestätigt werden können. Die Funktion ist sehr stark von der CSS-Eigenschaft `btn-default` abhängig. Nur wenn diese Klasse vorhanden ist, kann die Buchung bestätigt werden. Sollte später einmal das Styling des Buttons geändert werden, muss beachtet werden, dass die Funktion ebenfalls angepasst werden muss, um weiterhin zu funktionieren.

In React auf der anderen Seite wird der Entwickler zur Abstraktion von Daten gezwungen. Konzeptionelle Daten werden im Store gespeichert und sind strikt von der Anzeige getrennt. Wenn Daten geändert werden, werden diese nicht über die Position des Elements im DOM oder über CSS-Eigenschaften angesprochen. Dadurch können Änderungen des CSS oder des DOM ohne Risiko durchgeführt werden.

Des Weiteren werden im alten Veranstaltungseditor beim Betätigen des Speichern-Buttons, der als Submit-Button fungiert, alle Inputs gleichzeitig an den Server übermittelt. Bei dem Versuch, besonders üppige Veranstaltungen mit vielen Aufgaben und Diensten abzuspeichern, kann es zu Problemen mit dem Server kommen, woraufhin eine Fehlermeldung vom Typ „E_WARNING“ angezeigt wird (*Liste Der Php.Ini-Direktiven*, o.D.). Ein Grund dafür ist das Prinzip von CakePHP mit sogenannten „hidden“, also versteckten Eingabefeldern zu arbeiten. Diese Eingabefelder können weder vom Nutzer gesehen noch bearbeitet werden. Sie dienen lediglich dazu, Informationen zu lagern und zu übermitteln. Ein typisches Beispiel hierfür sind ID-Felder. IDs sollen vom Ottonormalverbraucher weder eingesehen noch verändert werden können. Es ist aber trotzdem wichtig, dass auch diese Informationen an den Server geschickt und in die Datenbank gespeichert werden. Dadurch, dass zu jeder Aufgabe und jedem Dienst noch versteckte Felder hinzukommen, kumuliert die Input-Anzahl rasch. Wenn die Anzahl einen bestimmten Wert überschreitet, tritt das Server-Problem auf. In der Konfigurationsdatei

`php.ini` wird durch die Variable `max_input_vars` ein Wert für die maximal mögliche Anzahl an Inputs festgelegt, die gleichzeitig an den Server geschickt werden dürfen. Standardmäßig wird der Wert auf 1000 begrenzt. (*Liste Der Php.Ini-Direktiven*, o.D.)
Diese Begrenzung wurde als Sicherheitsvorkehrung eingerichtet. Sie soll vor mutwilligen Angriffen des Servers schützen und einen „Denial of Service“ (DOS) verhindern. Wird die Limitierung überschritten, werden alle überschüssigen Inputs außer Acht gelassen. Eine Lösung des Problems ist das Heraufsetzen des verantwortlichen Variablen-Wertes in der Konfigurationsdatei. Wenn das Problem bekannt ist, lässt es sich gut bewältigen. Eventuell kann es bei einer alternativen Herangehensweise zukünftig vollständig beseitigt werden. Dies wird sich später im Abschnitt der Beschreibung des neuen Speicherprozesses (siehe 6.4.4.8) noch genauer zeigen.

6.4 Erarbeitungsprozess der Lösung

Zunächst muss genauer betrachtet werden, welche Funktionalitäten die neue Software-Version beinhalten soll. Dafür muss die alte Version eingehend inspiziert und jegliche Funktionalität extrahiert werden. Daraus kann zunächst ein übersichtliches Use-Case-Diagramm erstellt werden. Dann geht es an die Aufgliederung in Components. Im Speziellen um die Beantwortung der Frage, welche Components erstellt werden müssen und wie diese zusammenhängen. Noch bevor es an das Schreiben des Codes geht, müssen grundlegende Entscheidungen getroffen werden. Anschließend erfolgen die Code-Implementierung und die Spezifikation von Unit Tests. Nachfolgend wird jeder Schritt für sich im Detail beleuchtet.

6.4.1 Grafische Visualisierung des Codes

In einem Anwendungsfalldiagramm (auch bekannt als Use-Case-Diagramm) kann übersichtlich veranschaulicht werden, was die Anwendung aus Sicht des Anwenders leisten soll. Hierbei handelt es sich um ein Unified-Modeling-Language(UML)-Diagramm. (*Das Use-Case-Diagramm (Anwendungsfalldiagramm) in UML*, 2020)

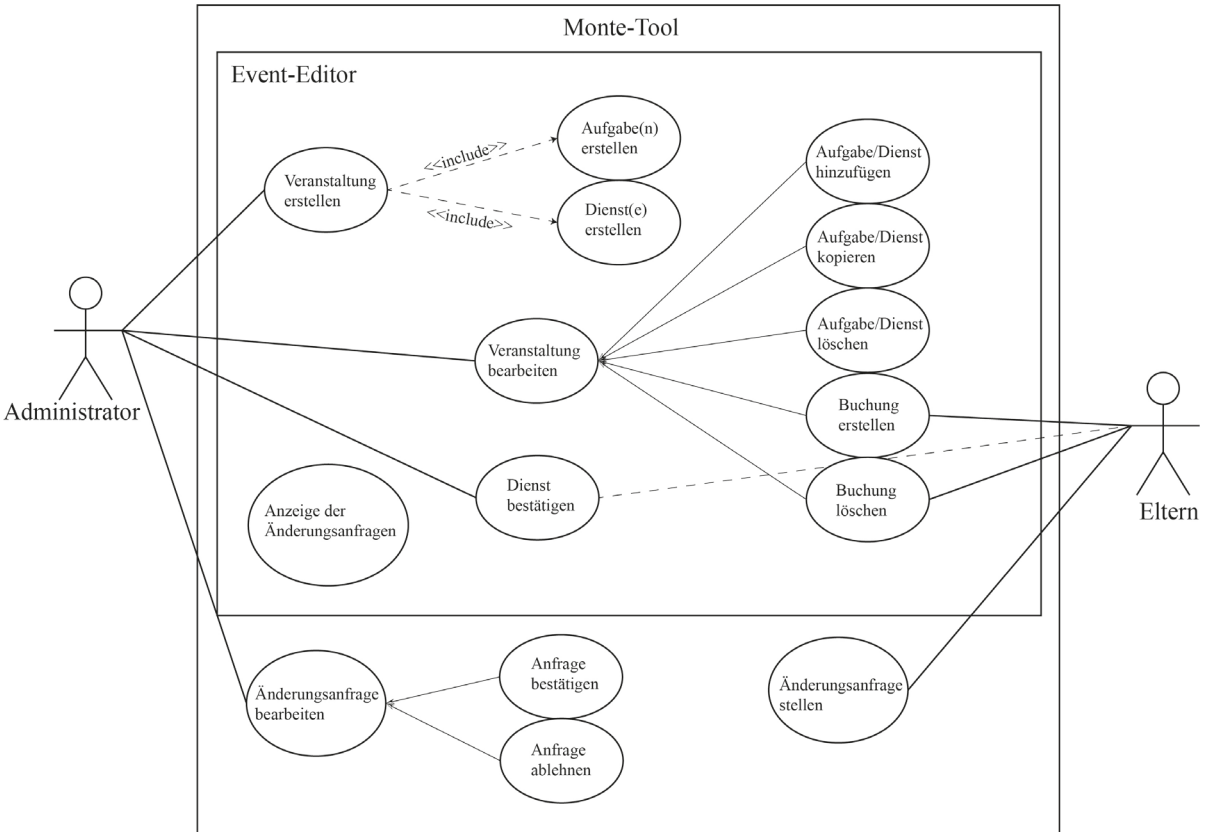

Abbildung 10: Use-Case-Diagramm

In dem Diagramm (siehe Abbildung 10) gibt es zwei Akteure, die als Anwender des Systems agieren. Der Administrator wird auf der linken Seite dargestellt, die Eltern der Montessori-Schulkinder auf der rechten Seite. Das System, mit dem die Akteure interagieren wird als Rechteck visualisiert. In diesem Fall ist das der Event-Editor, der Teil des Monte-Tools ist. Innerhalb des Systems werden verschiedene Anwendungsfälle und deren Beziehungen mittels Ellipsen und Pfeilen veranschaulicht. Die Linien zwischen Akteur und Anwendungsfall verdeutlichen, welche Akteure mit welchen Use Cases in Verbindung stehen.

Der Administrator kann mithilfe des Systems eine Veranstaltung erstellen. Zu diesem Use Case gehören automatisch die Anwendungsfälle „Aufgabe(n) erstellen“ und „Dienst(e) erstellen“ dazu. Das wird durch eine „include“-Beziehung verdeutlicht. Darüber hinaus kann er eine Veranstaltung bearbeiten. Dieser Use Case untergliedert sich in verschiedene Optionen (siehe Abbildung 10). Des Weiteren ist er in der Lage, einen erledigten Dienst zu bestätigen. Die Eltern auf der anderen Seite können für sich Dienste buchen oder Buchungen löschen. In Ausnahmefällen, also beim Übernehmen einer „Organisatoraufgabe“, erhalten Eltern das Privileg, Dienste zu bestätigen. Außerhalb des Event-Editors besteht für Eltern die Möglichkeit, Änderungsanfragen zu stellen, die dann von einem Administrator bearbeitet bzw. entweder bestätigt oder

abgelehnt werden. Diese Anwendungsfälle wurden mit in das Diagramm aufgenommen, da sie Einfluss auf den Event-Editor nehmen, denn die Informationen bezüglich der Änderungsanfragen werden im Event-Editor angezeigt. Das Monte-Tool beinhaltet, unabhängig vom Event-Editor viele weitere Use Cases, die aber hier nicht von Bedeutung sind. Aus diesem Grund ist das Rechteck, welches das Monte-Tool darstellt, nach unten hin geöffnet.

6.4.2 Übersichtliche Gliederung durch Components

Die Stärke von Components besteht in ihrer Wiederverwendbarkeit. Einmal geschrieben können sie beliebig oft an verschiedenen Stellen der Anwendung eingebunden werden.
Wenn bestimmte Bausteine einer Anwendung mehrmals vorkommen sollen, lediglich mit unterschiedlichen Inhalten, liegt der Gedanke nahe, aus diesem Baustein einen Component zu machen. Wenn eine grobe Ahnung darüber besteht, welche Components für die Anwendung gebraucht werden, muss festgelegt werden, wie diese ineinander zu verschachteln sind. Um das zu visualisieren, eignet sich ein „Component-Tree“ (siehe Abbildung 11), ein Baumdiagramm, das die unterschiedlichen Hierarchien der Components veranschaulicht.

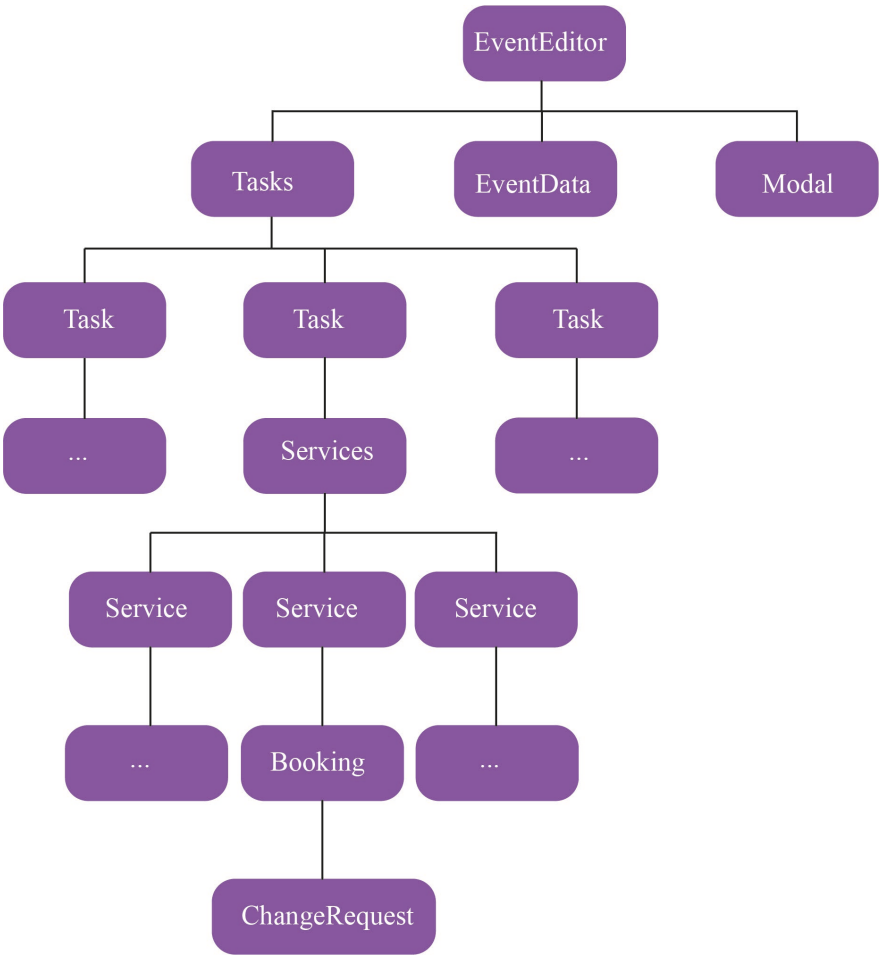

Abbildung 11: Component-Tree

Ganz oben befindet sich der „Root-Component“. Dieser verzweigt sich immer weiter nach unten. Im Falle des Projektes heißt er `EventEditor`. Darunter angeordnet sind die `Tasks` und der `EventData`-Component. Innerhalb des `Tasks`-Component werden dynamisch einzelne `Task`-Components eingebunden. Dies geschieht mit Hilfe der `map`-Funktion. Innerhalb jedes `Task`-Components befindet sich der `Services`-Component. Aus Gründen der Übersichtlichkeit wird die weitere Aufgliederung des Baumes nur an einer Stelle vorgenommen. Bei den `Services` wird das gleiche Prinzip angewendet, wie bereits zuvor bei den `Tasks`. Je nachdem wie viele Dienste es gibt, wird die passende Anzahl an `Service`-Components dynamisch gerendert. Die dazugehörige Funktion sieht wie folgt aus:

```
1. (task.services).map((id) => {
2.       return (
3.           <Service
4.            service={services[id]}
5.            key={id}
6.            task={task}
7.            />
8.        );
9. })
```

Das Modal, das eingeblendet wird, um Benutzer für eine Buchung auszuwählen wird relativ hoch in der Hierarchieebene eingegliedert, denn es muss im Vordergrund, außerhalb der Bootstrap-Panels, eingeblendet werden. Der Component-Tree, der zu Beginn eines Projektes erstellt wird ist keinesfalls in Stein gemeißelt. Während dem Implementierungsprozess können weitere Components sinnvoll erscheinen und hinzugefügt werden. Auch andere Anpassungen, beispielsweise an der Hierarchie können nachträglich noch vorgenommen werden.

6.4.3 Grundlegende Entscheidungen

Sollen ausschließlich „Functional Components“ verwendet werden oder auch „Class Components“? Wird der State ohne Redux verwaltet oder doch lieber mit dem State-Management-Tool? Diese Fragen stellen sich, noch ehe mit dem Schreiben des Codes begonnen wird. Im Folgenden werden verschiedene Optionen in Erwägung gezogen und Entscheidungen werden getroffen.

6.4.3.1 Ausschließlich „Functional Components“ oder mitunter „Class Components“

In React gibt es zwei verschiedene Möglichkeiten einen Component zu schreiben: entweder in Form einer Funktion oder in Form einer Klasse. Ersterer wird auch als „Functional Component“

bezeichnet, letzterer als „Class Component“. Die unterschiedlichen Schreibweisen werden an folgenden zwei Beispielen verdeutlicht. (Yamazaki, 2020)
Functional Component:

```
1. import React from "react";
2.
3. const FunctionalComponent = () => {
4.  return <h1>Hello, world</h1>;
5. };
```

Class Component:

```
6. import React, { Component } from "react";
7.
8. class ClassComponent extends Component {
9.  render() {
10.   return <h1>Hello, world</h1>;
11. }
12.}
```

Bevor die Version React 16.8 erschienen ist, war es nicht möglich State in „Functional Components“ zu verwenden. Um State verwalten zu können, musste man sich also gezwungenermaßen für „Class Components“ entscheiden. Damals war es gängige Praxis beide Arten von Components einzusetzen. „Functional Components“ für Components, die ohne State auskommen und „Class Components“ für Components mit State. Eine entscheidende Neuerung, die mit der Version 16.8 verbunden war, nennt sich React Hooks. Mithilfe von Hooks ist es seither möglich ausschließlich „Functional Components“ zu schreiben, egal ob mit oder ohne State.
Der Trend geht in Richtung der modernen Herangehensweise mit ausschließlich „Functional Components“. Das hat mehrere Gründe. Zum einen ist die Schreibweise für „Functional Components“ einfacher und kürzer, was der Verständlichkeit zugutekommt. Zum anderen kann von der Weiterentwicklung der Hooks profitiert werden, denn React arbeitet an Neuerungen und Verbesserungen im Zusammenhang mit Hooks. (Yamazaki, 2020) Ein weiterer Vorteil ist, dass Components nicht konvertiert werden müssen, wenn sich im Nachhinein herausstellt, dass ein Component wider Erwarten State verwenden sollte.
Auf der anderen Seite ist es aber auch legitim, sich für die traditionelle Variante zu entscheiden. React hat zugesichert, dass eine Unterstützung von „Class Components“ auf absehbare Zeit bestehen bleibt (Yamazaki, 2020). Diese Zusicherung ist auch für ältere Anwendungen von großer Bedeutung, die auf die Verwendung von „Class Components“ setzen. Ein weiterer möglicher Beweggrund für „Class Components“ könnte sein, dass sich Entwickler an die

ursprüngliche Herangehensweise gewöhnt und mit ihr angefreundet haben. Die modernen Hooks könnten noch unbekannt sein und für Verständnisschwierigkeiten sorgen.
In erster Linie kommt es bei der Entscheidung aber auf Konsistenz an. Wenn man sich auf eine Arbeitsweise festgelegt hat, sollte man diese auch konsequent verfolgen. In einem Entwicklerteam ist es wichtig, dass sich die Mitarbeiter untereinander abstimmen und eine gemeinsame Entscheidung treffen. Beim Monte-Tool wurde bereits ein anderer Teil der Anwendung mit Hooks und ausschließlich „Functional Components“ umgesetzt. Um konsistent daran anzuschließen, ist es sinnvoll, sich ebenfalls für den modernen Ansatz zu entscheiden.

6.4.3.2 Verwendung von Redux

Ob es sinnvoll ist, Redux zu verwenden, hängt davon ab, an welcher Stelle man Zugriff auf den State haben möchte. Soll der State nur innerhalb desselben Components, in dem der State erstellt wird, und des Child-Components direkt darunter zur Verfügung stehen, lässt sich das auch ohne Redux bewerkstelligen. In diesem Fall können die Daten aus dem State ganz einfach via Props weitergegeben werden. Auch an tiefer liegende Components können die Props weiter durchgereicht werden. Je länger die Component-Kette ist, die das Datenpaket durchlaufen muss, desto unübersichtlicher wird das State-Management. Was ist außerdem, wenn man auf einer höher gelegenen Ebene im Component-Tree auf den State zugreifen möchte? Ohne Redux wäre das eine sehr komplizierte Angelegenheit. Der Vorteil von Redux besteht darin, dass der State an einem zentralen Ort abgespeichert wird, dem sogenannten Store. Auf diesen hat jeder Component Zugriff, ganz gleich an welcher Stelle des Component-Trees er sich befindet. Allerdings gibt es auch einen negativen Aspekt: Beim Erstellen von Actions und Reducern ist deutlich mehr Schreibaufwand nötig als bei der Verwendung des internen States.
Im konkreten Fall des Praxisprojektes hat es sich als sinnvoll erwiesen, Redux zu verwenden. Im Folgenden wird ein Beispiel näher beschrieben, das den Vorteil der Implementierung von Redux im Event-Editor verdeutlichen soll:
Der Nutzen von Redux zeigt sich beispielsweise an der booleschen Variable `showServices`, die Indikator dafür ist, ob die Dienste aus- oder eingeklappt sind. Würde diese Eigenschaft lediglich im lokalen State definiert werden, könnte man nur innerhalb desselben Components Änderungen daran durchführen. Mit Redux wird es möglich, an jeder beliebigen Stelle die `showServices`-Eigenschaft anzupassen. Dadurch wird es, realisierbar, die Dienste nicht nur bei Klick auf den Dienste-Anzeigen-Button im `Task`-Component auszuklappen, sondern beispielsweise auch bei Klick auf den Speichern-Button im `SaveButton`-Component. Das ist vor allem hilfreich, wenn der Nutzer vergessen hat im Bereich der Dienste ein Eingabefeld

auszufüllen. Statt den Speichervorgang anzustoßen, kann der Nutzer auf die fehlende Eingabe aufmerksam gemacht und die betreffenden Dienste können ausgeklappt werden.
Abgesehen von diesem konkreten Fallbeispiel ist es für den Entwickler angenehm, nicht überlegen zu müssen, in welchem Component welche Daten des States am geschicktesten initialisiert werden sollten. Mit Redux geschieht das für alle Daten in den Reducern.
Der Import-Editor, ein weiterer Teil des Monte-Tools wurde bereits vom Evidentmedia-Team unter Einbeziehung von React und Redux programmiert. In Anbetracht der Kontinuität innerhalb einer Anwendung, bietet sich deshalb auch im Event-Editor die Arbeit mit Redux an. So verwenden verschiedene Teile des Monte-Tools dieselben Strukturen und Konzepte. Unterschiedlichen Mitarbeitern fällt es dann leichter, Parallelen zwischen den Abschnitten zu ziehen und sich in verschiedenen Teilen des Codes zurechtzufinden.

6.4.4 Code-Implementierung

Der nächste Abschnitt widmet sich dem eigentlichen Rewrite-Prozess. Stück für Stück werden einzelne Lösungsschritte genauer erklärt und bei passenden Gelegenheiten wird auf konkrete Beispiele aus dem Quellcode eingegangen.
Da das UI von „Veranstaltung bearbeiten“ und „Veranstaltung erstellen“ sehr ähnlich ist, hat es Sinn gemacht, sich zunächst auf einen der beiden Dialoge zu konzentrieren. Aufgrund der zusätzlichen Schwierigkeit, passende Daten aus der Datenbank zu beziehen, war es förderlich, zunächst mit dem etwas komplexeren Dialog „Veranstaltung bearbeiten“ zu starten.

6.4.4.1 State-Organisation mit Redux

Um auf Daten zuzugreifen, die bereits in der Datenbank gespeichert sind, müssen diese vom Events-Controller an die View und von dort aus weiter an React übergeben werden. Zu Beginn ist der Store noch leer – In React wird der Store mit den Daten befüllt. Eine Visualisierung des Stores findet sich in den Redux Developer Tools. In den Reducern wird der Anfangszustand des Stores bestimmt. Hierbei gibt es zahlreiche Möglichkeiten, die Daten zu strukturieren. Wie beim Einräumen eines Schrankes, kann der Entwickler selbst entscheiden, welche und wie viele Daten er mit welchem Ordnungssystem hineinlegen möchte. Jeder Reducer bildet einen Unterpunkt im Store. Beim Event-Editor haben sich zunächst vier Stück angeboten: `eventData`, `tasks`, `services` und `user`. Dementsprechend tauchen diese Bezeichnungen als Gliederungspunkte im Store auf (siehe Abbildung 12). Im Laufe des Projektfortschrittes erschienen weitere Reducer und somit Gliederungspunkte sinnvoll: `UI`, `load`, `changeRequests` und `validation`.

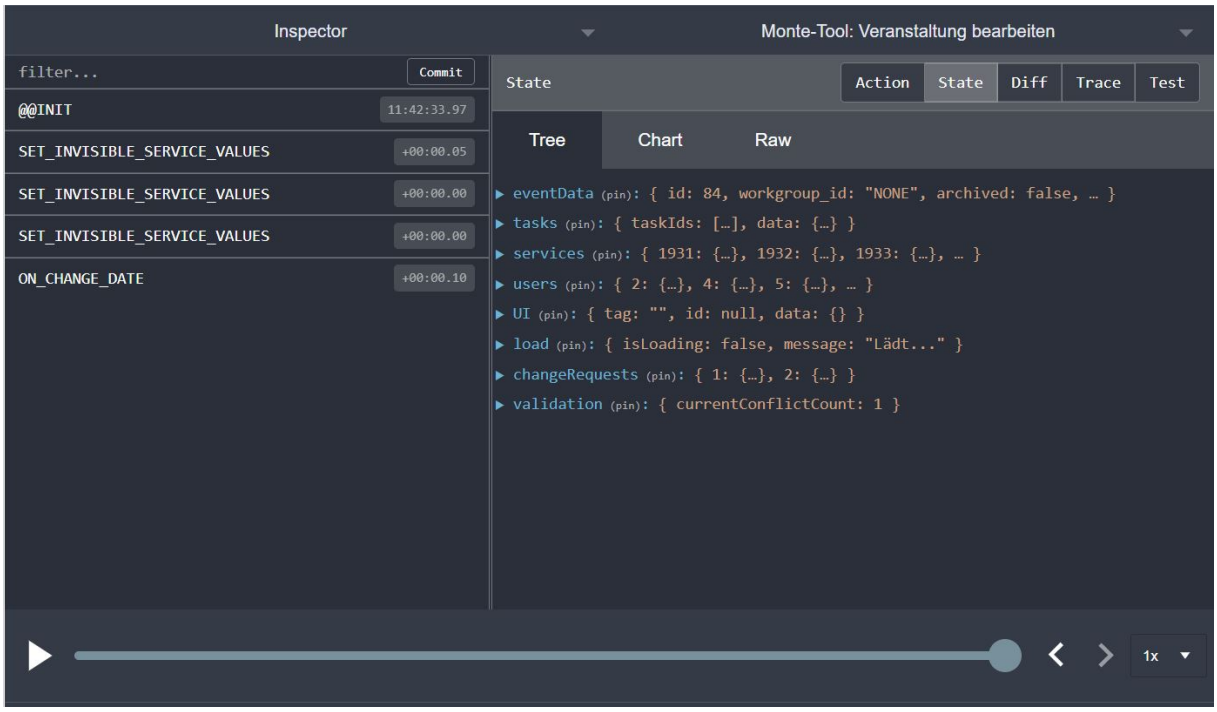

Abbildung 12: Gliederungspunkte im Store, zugeklappt

Die `Tasks` untergliedern sich in die beiden Unterpunkte `taskIds` und `data`. Ersterer beinhaltet ein Array aller Aufgaben-IDs. Das ist relevant für die Reihenfolge, in der die Aufgaben angezeigt werden. Es wird benötigt, um kopierte Aufgaben an der richtigen Stelle in das Formular einzufügen. Letzterer enthält einzelne Knotenpunkte, benannt nach den jeweiligen IDs, wobei jeder davon die Daten zu einer Aufgabe beinhaltet. Die Untergliederung der `Services` erfolgt nach dem gleichen Prinzip.

Nach eingehender Überlegung hat sich gezeigt, dass es keinen Sinn hat, die Daten zu allen Events aus der Datenbank zu holen und im Store abzuspeichern. Das wäre nur dann notwendig, wenn eine Übersicht aller Events angezeigt werden sollte, was aber nicht der Fall ist. Also werden lediglich die Daten des aktuell angezeigten Events im Store benötigt.

Des Weiteren kann der Datenspeicher entlastet werden, indem einige `User`-Daten erst „just in time“, also erst, wenn sie wirklich gebraucht werden, vom Server geholt werden. Für den Anfangszustand werden lediglich die Daten der Nutzer in den Store gespeichert, die zur ausgewählten Arbeitsgruppe passen. Ändert sich die Auswahl der Arbeitsgruppe, wird der Datenbestand im Store passend erweitert.

Um Doppelungen und demnach Redundanz zu vermeiden, wird darauf verzichtet, innerhalb einer Aufgabe die dazugehörigen Dienste mit aufzuführen, die ja bereits innerhalb ihres eigenen Gliederungspunktes zu finden sind. Lediglich ein Array mit den IDs der Dienste befindet sich an dieser Stelle, um die Verknüpfung der Aufgabe mit den dazugehörenden Diensten nicht zu

verlieren. Das folgende Code-Beispiel verdeutlicht, die Struktur einer Aufgabe, die im Store gespeichert ist:

```
1.  '295': {
2.          id: 295,
3.          event_id: 50,
4.          name: 'Reinigung in der RB1 nach Sommerferien',
5.          description: 'Tätigkeit: Abstauben und säubern des Materials und der
    Regale',
6.          created: null,
7.          modified: '2021-05-06T08:32:37+02:00',
8.          services: [
9.            1149,
10.           1150,
11.           1151,
12.           1867
13.         ],
14.         containsChangeRequest: true,
15.         hasBookedWork: true,
16.         isNew: false,
17.         showServices: false,
18.         value: 2
19.       }
```

In diesem Zusammenhang ist es wichtig zu beachten, dass, wenn beispielsweise ein neuer Dienst erstellt wird, Änderungen am State innerhalb zwei verschiedener Reducer vorgenommen werden müssen. Zum einen muss im `Services`-Reducer ein neuer Dienst hinzugefügt werden und zum anderen muss im `Tasks`-Reducer die Liste der dazugehörigen Dienst-IDs angepasst werden.

6.4.4.2 Handhabung der Benutzereingaben

Eine essenzielle Funktionalität des Veranstaltungseditors ist, dass der Benutzer Eingaben in den Inputs machen kann. Diese werden dann im Store abgespeichert und die View aktualisiert sich bei jedem Tastendruck. Das Prinzip, das beim Umgang mit den Benutzereingaben angewendet wird, nennt sich „Two way binding“ (siehe Abbildung 13). Dieses besagt, dass auf der einen Seite eine Änderung am Model, also an den Daten, eine Änderung der View zur Folge hat. Auf der anderen Seite führt eine Änderung an der View zu einer Anpassung des Models.

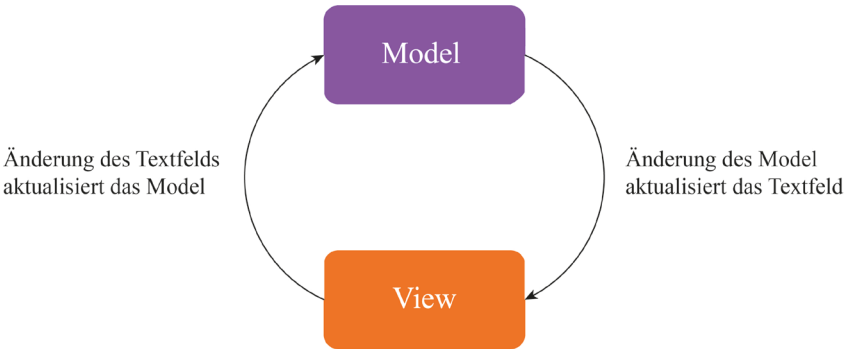

Abbildung 13: Two way Binding

Möchte der Nutzer der Anwendung ein bereits vorhandenes Event editieren, müssen die Eingabefelder mit den Werten aus dem Store vorausgefüllt werden. Aus diesem Grund besitzt jedes Eingabefeld ein `value`-Attribut. Mithilfe des `useSelector`-Hooks werden die benötigten Daten aus dem Store geholt und als `value` eingesetzt. Tippt nun der Nutzer etwas in das Eingabefeld, sieht er noch keine Veränderung. Es fehlt die `onChange`-Eigenschaft. `OnChange`, also beim Eintippen eines neuen Inhalts, muss eine Action dispatched werden, mitsamt der Benutzereingabe als `Payload`. Unter `Payload` versteht man in diesem Zusammenhang zusätzliche Informationen, die zusammen mit der Action an den Reducer übergeben werden. In den betroffenen Reducern kann dann der Store gemäß der Benutzereingabe angepasst werden. Dadurch ändert sich auch der `value` des Eingabefeldes und somit das, was im Feld angezeigt wird.

Um diesen Prozess zu veranschaulichen, wird im Folgenden auf ein konkretes Beispiel eingegangen. Betrachtet wird das Eingabefeld für den Veranstaltungsnamen, welches sich im `EventData`-Component befindet:

```
1.  <label>Veranstaltungsname</label>
2.  <div className="panel panel-default">
3.        <input className="form-control"
4.               type="text"
5.               name="Veranstaltungsname"
6.               value={eventData.name}
7.               onChange={(event)=>
8.               dispatch(actionCreators.onChangeName(event.target.value))}>
9.         </input>
10. </div>
```

Tippt der Benutzer in dieses Feld, wird die Action `onChangeName` aufgerufen. An diese wird die Benutzereingabe als Payload verschickt. In der dazugehörigen Action wird der `Type` auf `ON_CHANGE_NAME` gesetzt:

```
1. import * as actionTypes from './actionTypes';
2.
3. export const onChangeName = (userInput) => {
4.     return {
5.         type: actionTypes.ON_CHANGE_NAME,
6.         userInput
7.     }
8. }
```

Im `Switch-Case-Statement` des `eventData`-Reducers wird abgefragt, um welchen `Type` es sich handelt und infolgedessen nur der Abschnitt mit dem entsprechenden `Type` ausgeführt:

```
1. const eventDataReducer = (state = initialState, action) => {
2.     switch(action.type) {
3.         case actionTypes.ON_CHANGE_NAME: {
4.             return {
5.                 ...state,
6.                 name: action.userInput
7.             }
8.         }
```

Im Reducer wird definiert, wie der Store aktualisiert werden muss. Hierbei ist es wichtig, „immutable“ vorzugehen, was im Deutschen mit „unveränderlich“ übersetzt werden kann. Das bedeutet, es muss zunächst eine Kopie des States erstellt werden, bevor dieser verändert werden darf. Eine Kopie kann mithilfe des Spread-Operators, dargestellt durch drei Punkte, erzeugt werden. Das Duplikat kann nun ohne Weiteres modifiziert werden. Durch die Änderung des Stores ändert sich auch der Veranstaltungsname, den der Benutzer im gleichnamigen Eingabefeld sehen kann.
Bei jedem einzelnen Input des Event-Editors muss dementsprechend vorgegangen werden. Deshalb gibt es für jedes Eingabefeld eine Action, die bei Veränderung dispatched wird. Damit keine Tippfehler bei der Definition der `Types` auftreten, werden diese im Dokument `actionTypes.js` definiert und an den Stellen, an denen sie benötigt werden, importiert. In dieser Datei erhält man einen Überblick, wie viele Actions es gibt, die `onChange` ausgeführt werden.

```
1.  //onChange Handler
2.  export const ON_CHANGE_NAME = 'ON_CHANGE_NAME';
3.  export const ON_CHANGE_DATE = 'ON_CHANGE_DATE';
4.  export const ON_CHANGE_USERCLAIMABLE = 'ON_CHANGE_USERCLAIMABLE';
5.  export const ON_CHANGE_BOOKABLE = 'ON_CHANGE_BOOKABLE';
6.  export const ON_CHANGE_TASK_NAME = 'ON_CHANGE_TASK_NAME';
7.  export const ON_CHANGE_TASK_DESCRIPTION = 'ON_CHANGE_TASK_DESCRIPTION';
8.  export const ON_CHANGE_HOUR_VALUE = 'ON_CHANGE_HOUR_VALUE';
9.  export const ON_CHANGE_CONTROLS_EVENT = 'ON_CHANGE_CONTROLS_EVENT';
10. export const ON_CHANGE_SERVICE_NAME = 'ON_CHANGE_SERVICE_NAME';
11. export const ON_CHANGE_WORKGROUP = 'ON_CHANGE_WORKGROUP';
12. export const ON_CHANGE_SERVICETYPES = 'ON_CHANGE_SERVICETYPES';
13. export const ON_CHANGE_SERVICETYPE_NAME = 'ON_CHANGE_SERVICETYPE_NAME';
14. export const ON_CHANGE_SERVICETYPE_MAX_COUNT = 'ON_CHANGE_SERVICE-
    TYPE_MAX_COUNT';
```

6.4.4.3 Buchungsprozess

Beim Buchungsprozess spielen das Benutzerauswahl-Modal und der `Booking`-Component eine entscheidende Rolle. Ersteres ist ein Dialogfenster, welches eingeblendet werden kann, um einen Benutzer für einen Dienst auszuwählen.

6.4.4.3.1 Dialogfenster („Modal“)

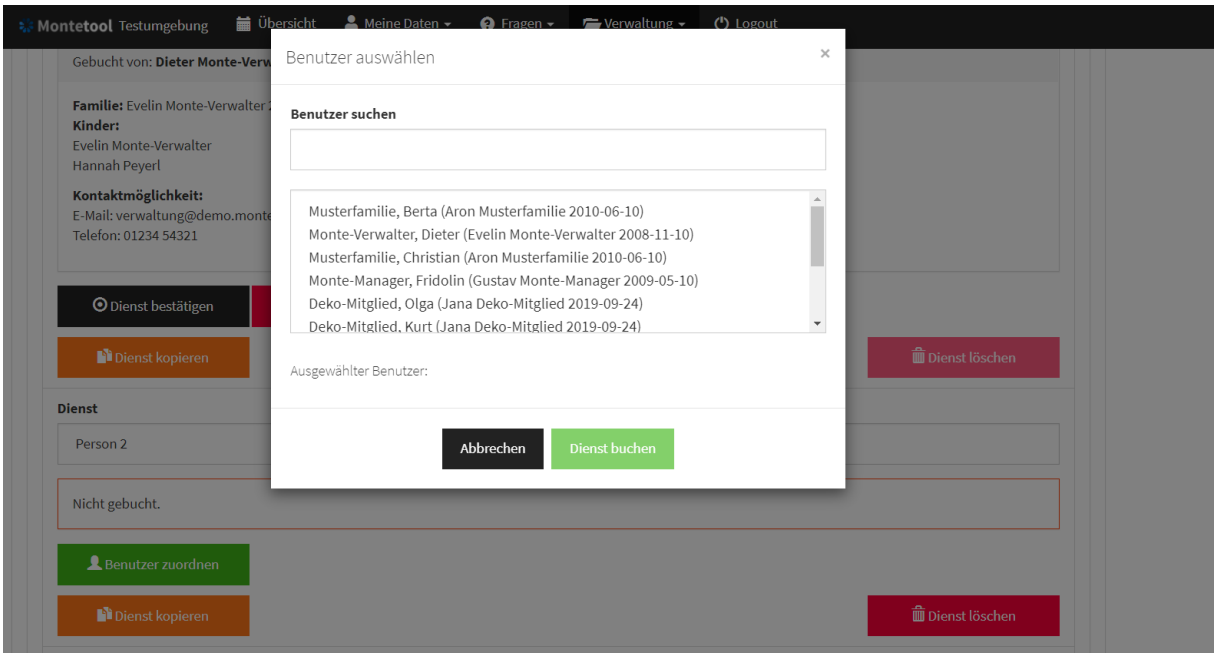

Abbildung 14: Benutzerauswahl Modal

Das Dialogfenster (siehe Abbildung 14) besteht aus dem Rahmen-Component Modal, dem der `UserSelection`-Component als Kind-Component übergeben wird. Im Äußeren der beiden

wird mithilfe des `useEffect`-Hooks dafür gesorgt, dass die Seite im Hintergrund nicht bewegt werden kann, solange der Dialog eingeblendet wird:

```
1. useEffect(() => {
2.     document.body.classList.add('modal-open');
3.     return function () {
4.         document.body.classList.remove('modal-open');
5.     }
6. }, [])
```

Der innere Component enthält die eigentliche Substanz: Informationen darüber, welche `User` zur Auswahl stehen und welcher `User` ausgewählt wurde.

Beim Einblenden des Dialogfensters wird die entsprechende Dienst-ID in den Store gespeichert, die später für das Buchen des Dienstes benötigt wird. Solange noch kein Benutzer ausgewählt wurde, bleibt der Dienst-Buchen-Button gesperrt. Hierfür wird der interne State des `UserSelection`-Components verwendet: ganz konkret die Variable `bookable`.

Beim Klicken des Dienst-Buchen-Buttons wird die Action `bookService` ausgelöst. Diese Action nimmt Einfluss auf den `Services`-Reducer. Dort wird ein neuer Work beim entsprechenden Dienst erstellt.

6.4.4.3.2 `Booking`-Component

Bei der Initialisierung des `Services`-Reducers erhält jeder Dienst die Eigenschaft `userId`. Diese ist `null`, wenn es keine Buchung des Dienstes gibt. Gibt es hingegen eine Buchung, enthält `userId` einen Zahlenwert. Im `Booking`-Component wird von dieser Logik Gebrauch gemacht. Je nachdem, ob eine Buchung vorhanden ist, werden unterschiedliche JSX-Abschnitte gerendert. So werden bei bestehender Buchung Informationen des Buchenden angezeigt. Zudem gibt es Buttons, mit denen die Buchung bestätigt oder entfernt werden kann (siehe Abbildung 15).

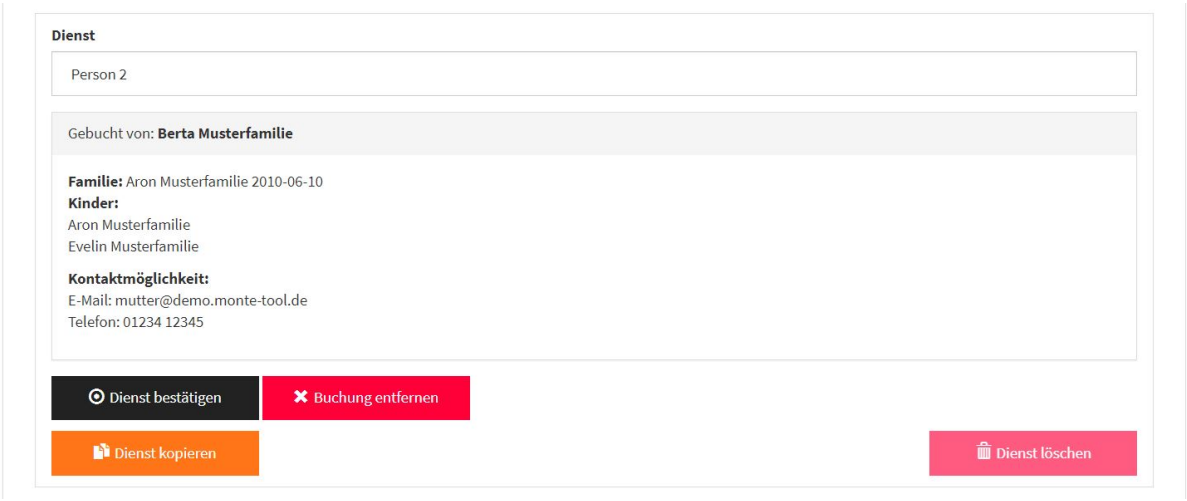

Abbildung 15: Dienst mit Buchung

Bei keiner vorliegenden Buchung wird diese Information im UI kommuniziert und der Benutzer-Zuordnen-Button wird zur Verfügung gestellt, um eine Buchung zu erstellen (siehe Abbildung 16).

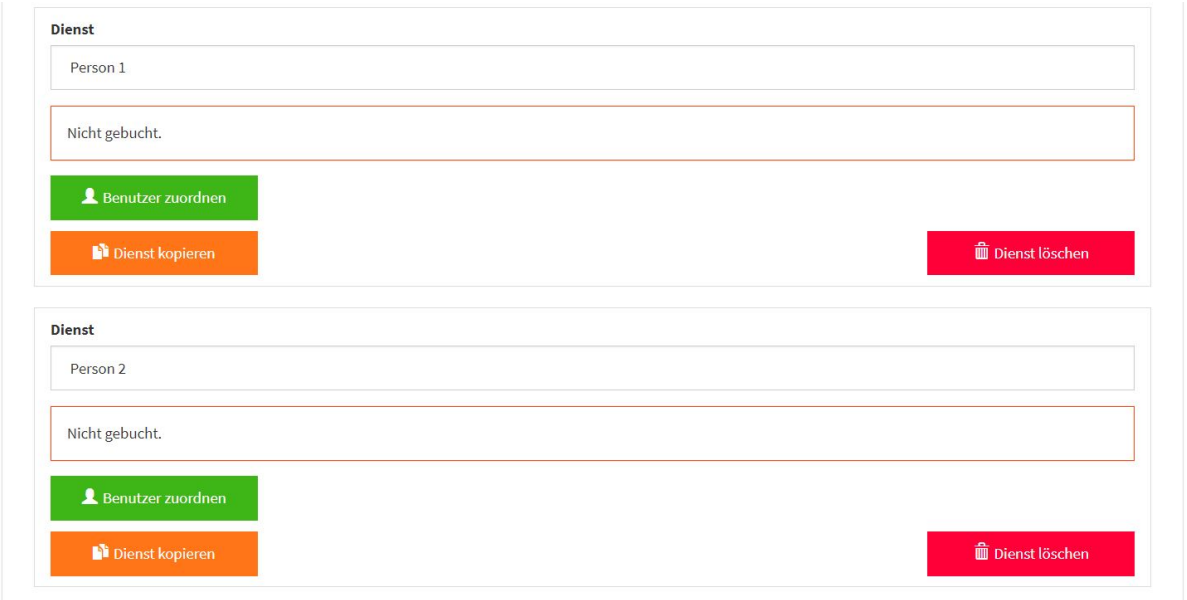

Abbildung 16: Dienste ohne Buchung

6.4.4.4 Anzeige der Änderungsanfragen

Um die passenden Hinweise anzeigen zu können, wenn eine Änderungsanfrage gestellt wurde, werden in der Datei `EventsController.php` die benötigten Änderungsanfragen zusammengestellt.

```
1.  $neededChangeRequests = [];
2.          foreach($event['tasks'] as $task) {
3.              foreach($task['services'] as $service) {
4.                if($service['containsChangeRequest'] === true) {
5.                  foreach ($service['works'] as $work) {
6.                    foreach ($changeRequests as $changeRequest) {
7.                      if($work['id'] == $changeRequest['origin_work_id'] ) {
8.                        array_push($neededChangeRequests, $changeRequest);
9.                      }
10.                   }
11.                 }
12.               }
13.             }
14.         }
```

Danach werden die Daten über die View an React übergeben und in den Store geschrieben. Ob ein entsprechender Hinweis mit den Daten aus dem Store angezeigt wird, entscheidet die boolesche Variable `containsChangeRequest`. Je nachdem, ob die Anfrage noch bearbeitet wird, bereits bestätigt oder abgelehnt wurde, werden dazu passende Informationen angezeigt.

6.4.4.5 Einbinden weiterer Modals

Im Laufe des Projekes hat sich gezeigt, dass neben dem Benutzerauswahl-Modal noch weitere Modals benötigt werden. Unter anderem ein Modal zum Erstellen von Limitierungen, Modals mit Auswahlmöglichkeiten oder Informationen für den Nutzer. Aus diesem Grund häuften sich die Modals im Root-Component, wobei jedes von Ihnen seine eigenen Actions zum Ein- und Ausblenden erhielt. Hierfür konnte eine bessere, allgemeingültige Lösung gefunden werden: Zum einen werden fortan die Modals an der Stelle im Component-Tree eingebunden, an der sie aufgerufen werden können. Damit die Modals trotzdem im Vordergrund in Erscheinung treten, wird mit „Portals“ gearbeitet. Diese bieten die Möglichkeit, einen Component an einer bestimmten Stelle im DOM zu rendern, die von dem eigentlichen Ort abweicht, an dem der Component eingebunden wird. Damit ist es möglich, die Modals auf der ersten Ebene innerhalb des HTML `body`-Tags einzublenden.

Jedes Modal besteht aus einem Wrapper-Component, dem ein Child-Component untergeordnet wird. Um fortan nur noch eine Action für das Einblenden aller Modals verwenden zu können, wird folgendes Prinzip angewendet: Beim Dispatchen der allgemeingültigen Action `showModal` wird ein Tag in den Store geschrieben. Beim Einbinden des Wrapper-Components wird diesem mithilfe der Props ebenfalls ein Tag übermittelt. Der Wrapper-Component prüft dann, ob die Props, die er erhalten hat mit dem Eintrag im Store übereinstimmen und rendert seinen

Inhalt nur in diesem Fall. In anderen Worten ausgedrückt verwaltet der Wrapper-Component jedes Modals seinen Anzeigezeitpunkt selbst. Durch die Verallgemeinerung der Actions zum Ein- und Ausblenden, kann im Actions- sowie im Reducer-File viel Code eingespart werden. Die Verallgemeinerung kommt der Übersichtlichkeit und somit auch der Wartbarkeit der Anwendung zugute. Außerdem ist es zukünftig einfach, neue Modals einzubinden, denn alle Actions sowie der Wrapper-Component sind bereits vorhanden und können wiederverwendet werden.

6.4.4.6 Verbesserung der User Experience

Wenn Buttons oder Dropdown-Listen gesperrt sind, besteht die Gefahr, dass der Nutzer nicht versteht, warum. Aus diesem Grund ist es sinnvoll, mittels Meldungen dem Benutzer die Gründe für eine Sperrung näher zu bringen.
Im Veranstaltungseditor werden die Buttons „Dienst löschen“ und „Aufgabe löschen“, deaktiviert, wenn eine zugehörige Buchung existiert. Mittels der `title`-Eigenschaft des Inputs kann dem Nutzer bei dem Fall eines gesperrten Buttons die Ursache kommuniziert werden. Die Botschaft wird eingeblendet, sobald der Nutzer den Cursor über den Button bewegt.
Auch die Arbeitsgruppe kann nur verändert werden, solange keine Buchungen vorhanden sind. Hierauf weist eine Meldung direkt unter dem Select-Feld hin.
Außerdem besteht die Möglichkeit, den Nutzer vor einem versehentlichen Löschen eines Datensatzes zu bewahren. Egal ob Aufgabe, Dienst oder Buchung – bevor eine endgültige Löschung durchgeführt wird, erscheint ein Hinweis mit einer Nachfrage, die abermals bestätigt werden muss.

6.4.4.7 Clientseitige Formular Validierung

Bevor das Formular an den Server geschickt wird, muss festgestellt werden, ob die Formularfelder mit den richtigen Werten befüllt wurden. Hierfür ist die clientseitige Formular Validierung zuständig. Man unterscheidet zwei verschiedene Arten. Zum einen die standardmäßig verfügbare Formular Validierung, die mit HTML5-Attributen arbeitet, zum anderen eine, an seine Bedürfnisse angepasste Validierung, implementiert mit JavaScript. (*Client-Side Form Validation*, o.D.)
Der Vorteil der bereits existierenden Browser Validierung ist, dass dafür kein großer Implementierungsaufwand nötig ist. Man ordnet den Eingabefeldern lediglich entsprechende Attribute zu, wie z.B. `minLength` oder `required`. Bei Inputs, die beispielsweise ein `required`-Attribut erhalten haben, wird der Browser automatisch prüfen, ob auch tatsächlich ein Eingabewert vorhanden ist. Falls nicht wird eine Fehlermeldung ausgegeben.

Im Zusammenhang mit der bereits verfügbaren Browser Validierung gibt es aber auch Nachteile. Zum einen ist sie in verschiedenen Browsern nicht konsistent. Das Aussehen der automatisierten Nutzer-Benachrichtigungen variiert von Browser zu Browser. In als veraltet geltenden Browsern, wie zum Beispiel im Internet Explorer 9, ist die eingebaute Validierung sogar gar nicht verfügbar (*Can I Use… Support Tables for HTML5, CSS3, Etc*, o.D.).

Zum anderen kann der Browser nur das Validieren, was Teil des DOM ist. Im Falle des neuen Event-Editors gibt es in diesem Zusammenhang ein erhebliches Problem. Es ist möglich, einige Eingabefelder einzuklappen. Im Konkreten handelt es sich um die Eingabefelder aller Dienste. Eine Besonderheit von React ist, dass diese eingeklappten Felder im DOM nicht mehr auftauchen. Deshalb würde die Browser Validierung diese Felder nicht berücksichtigen. Aus diesem Grund macht es Sinn, eine eigene Validierung zu implementieren.

Das Formular des Event-Editors wird auf zwei Ebenen validiert. Zum einen findet eine Validierung auf Input-Ebene statt, zum anderen auf Formular-Ebene. Für die Validierung auf Input-Ebene kommt ein `Input`-Component zum Einsatz. Dieser erhält die benötigte Validierungsfunktion als Prop. Mithilfe dieser ermittelt der Component seine eigene Validität. Damit der Nutzer in Ruhe seine Eingabe tätigen kann, wird die Validität erst ermittelt, nachdem der Nutzer das zu bearbeitende Feld wieder verlassen hat. Fällt das Ergebnis der Validität-Prüfung negativ aus, wird unter dem entsprechenden Input eine in Rot geschriebene Aufforderung erscheinen. Dies kann in Abbildung 17 betrachtet werden.

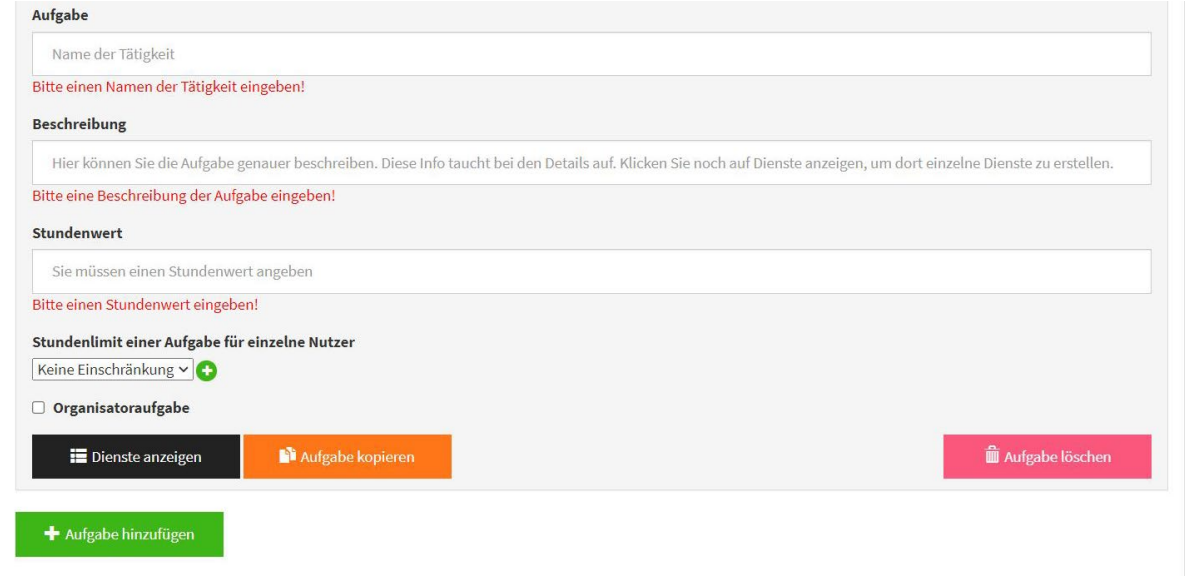

Abbildung 17: Validierung auf Input-Ebene

Die Validierung auf Formular-Ebene besteht aus einer Überprüfung der Einträge im Redux Store. Hierbei wird sowohl auf Richtig- als auch auf Vollständigkeit ausgewählter Einträge geprüft. Ausgelöst wird diese Validierung durch das Klicken des Speichern-Buttons. Der Nutzer wird mittels Modal über fehlende Eingaben informiert (siehe Abbildung 18).

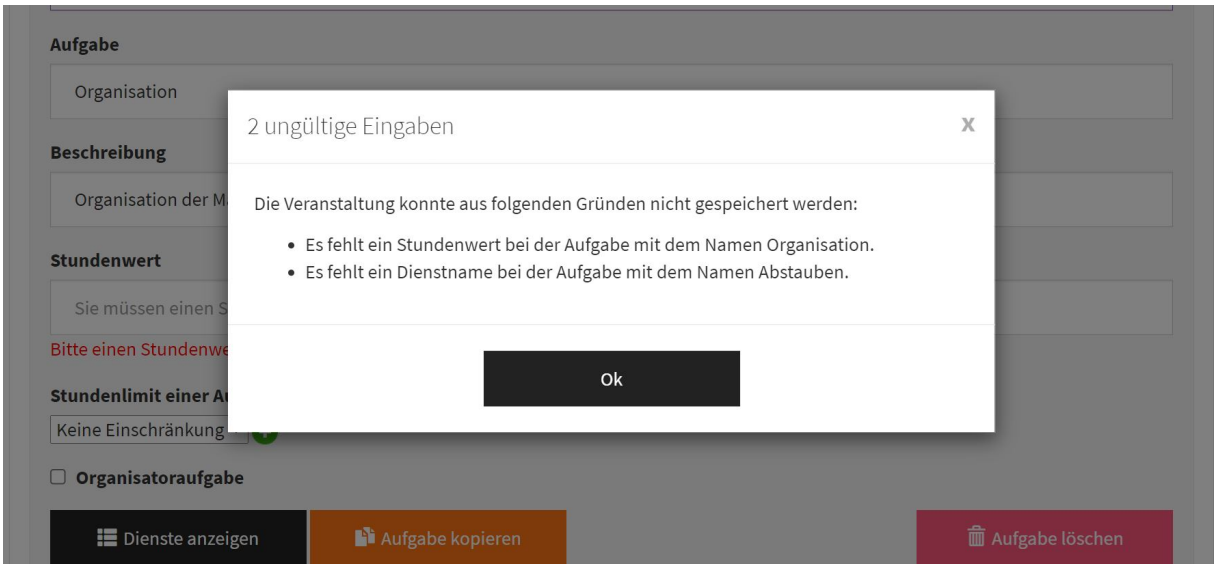

Abbildung 18: Rückmeldung der Validierung auf Formular-Ebene mittels Modal

Alternativ wäre es denkbar gewesen, fortlaufend die Daten im Store zu validieren und den Speichern-Button so lange zu sperren, bis ein valides Formular vorläge. Diese Variante wurde nicht ausgewählt, da durch das ständige Überprüfen des Stores unverhältnismäßig viel Rechenleistung aufgewendet werden müsste.

6.4.4.8 Abspeichern der Formulardaten

Wenn das Ergebnis der clientseitigen Formularvalidierung bestätigt, dass alle benötigten Eingaben vom Benutzer getätigt wurden, können die Daten an den Server geschickt werden. Dies geschieht bei Klick auf den Speichern-Button, der sich innerhalb des `SaveButton`-Components befindet und eine `fetch`-Methode aufruft.

In rudimentärer Form sieht diese wie folgt aus:

```
1.  //submit data to the backend
2.  const onSubmit = () => {
3.      fetch(__data__.paths.submitPath, {
4.          method: 'POST',
5.          headers: {
6.              'Content-Type': 'application/json'
7.          },
8.          body: JSON.stringify(preparedData)
9.      })
10.     .then(res => res.json())
11.     .then(function (data) {
12.         console.log(".then data", data);
13.     }).catch(function (error) {
14.         console.warn('Something went wrong.', error);
15.     });
```

Diese schickt die Daten aus dem Store im JSON-Format an eine Funktion im Controller. Hierbei handelt es sich um die Funktion `saveEvent()`. Im Controller müssen die Daten dann so umgebaut werden, dass sie mit den Daten in der Datenbank übereinstimmen. Das ist wichtig, um die Funktion `patchEntity()` ausführen zu können. Diese Funktion vergleicht eine alte und eine neue Instanz der Entität und aktualisiert lediglich die Eigenschaften, die sich verändert haben. Im Zuge des Datenumbaus müssen alle in React neu generierten IDs gelöscht werden, bei denen es sich genau genommen um Universally Unique Identifier (UUID) handelt. UUID werden bei allen Aufgaben, Diensten und Buchungen eingesetzt, die vom Nutzer neu angelegt werden. Beim Abspeichern der Daten in die Datenbank wird MySQL feststellen, dass es sich um neue Instanzen handelt und wird mittels Autoincrement neue IDs generieren, die noch nicht in der Datenbank verwendet werden.

Schließlich können mittels der `save`-Methode die aktualisierten Datensätze in die Datenbank gespeichert werden. Wenn das Speichern erfolgreich war, wird diese Information an den `SaveButton`-Component übermittelt. Der Nutzer gelangt auf die Übersichtsseite der Veranstaltungen und erhält eine Meldung über den erfolgreich ausgeführten Speichervorgang. War das Speichern nicht erfolgreich, wird herausgefunden, an was es gelegen hat, um auch in diesem Fall eine passende Meldung ausgeben zu können. Beispielsweise wäre es denkbar, dass die Verbindung zum Server nicht aufgebaut werden konnte.

Durch sogenannte Transactions kann außerdem verhindert werden, dass nur ein Teil der Daten in die Datenbank gespeichert wird. Bei Auftreten eines Fehlers wird der gesamte

Speichervorgang rückgängig gemacht, sodass wieder der Ausgangszustand vor Betätigen des Speichern- Buttons vorliegt.
Mit der Begrenzung der Eingabefelder durch die Variable `max_Input_Vars` (siehe 6.3) gibt es keine Probleme mehr, da die Daten nicht mehr in Form eines Formulars übermittelt werden und dadurch die Einschränkung der Eingabefelder entfällt.

6.4.4.9 Abfangen von Edge Cases

Unter Umständen kann es zu Ungereimtheiten kommen, wenn zwei Administratoren ein Event gleichzeitig bearbeiten. Wenn Admin eins das Event abspeichert, driften bei Admin zwei der State im Backend und der State im Frontend auseinander. Der Backend-State beinhaltet bereits alle Änderungen von Admin eins, aber der State im Frontend bei Admin zwei ist noch der Ursprüngliche. Es stellt sich die Frage, wie man am besten mit dieser Situation umgeht.
Da diese Situation nur sehr selten eintreten wird, spricht man von einem sogenannten „Edge Case“. Unter Edge Cases versteht man Grenzfälle, die nur in Ausnahmesituationen eintreten.
Im alten Event-Editor gab es bisher noch keine Logik, die dafür zuständig war, diese Art von Edge Cases abzufangen. Es macht Sinn darüber nachzudenken, wie Edge Cases abgefangen werden können, um auf alle Eventualitäten vorbereitet zu sein.
Eine denkbare Lösung wäre es, das gleichzeitige Bearbeiten des Events zu verbieten. Hierfür könnte man eine `wirdEditiert`-Eigenschaft einfügen und betroffenes Event so lange sperren, wie der Editiervorgang andauert. Problematisch wird dieser Lösungsansatz, wenn der Admin, während dem Editiervorgang andere Dinge dazwischen erledigt und darüber das Editieren vergisst. Das Event bliebe im Hintergrund geöffnet und somit auch für andere Nutzer gesperrt.
Für den Event-Editor wurde ein alternativer Lösungsansatz umgesetzt, bei dem das gleichzeitige Bearbeiten gestattet ist. Im Backend wird identifiziert, ob der State auseinanderdriftet, und somit ein gleichzeitiger Bearbeitungsvorgang stattgefunden hat. Wenn ja wird zwischen drei verschiedenen Fällen unterschieden. In Fall eins tangieren sich die Bearbeitungen nicht. Dann kann der Konflikt ohne Nutzereingabe behoben werden. Im zweiten Fall gibt es Überschneidungen der Nutzereingaben, die im Backend identifiziert werden können. Dann kann der Nutzer über ein Modal die für ihn richtige Änderung auswählen und übernehmen. Der dritte Fall tritt ein, wenn der Konflikt im Backend nicht eindeutig identifiziert und somit nicht durch den Nutzer behoben werden kann. Unter diesen Umständen muss die Seite neu geladen werden und Änderungen gehen verloren. In diesem Zusammenhang ist es aber wichtig, sich zu verdeutlichen, dass ein gleichzeitiges Editieren äußerst selten eintreten wird und ein nicht identifizierbarer Konflikt dementsprechend noch viel seltener.

In Abbildung 19 sieht man das Modal, welches dem Nutzer die Möglichkeit gibt, im Backend identifizierte Konflikte zu beheben.

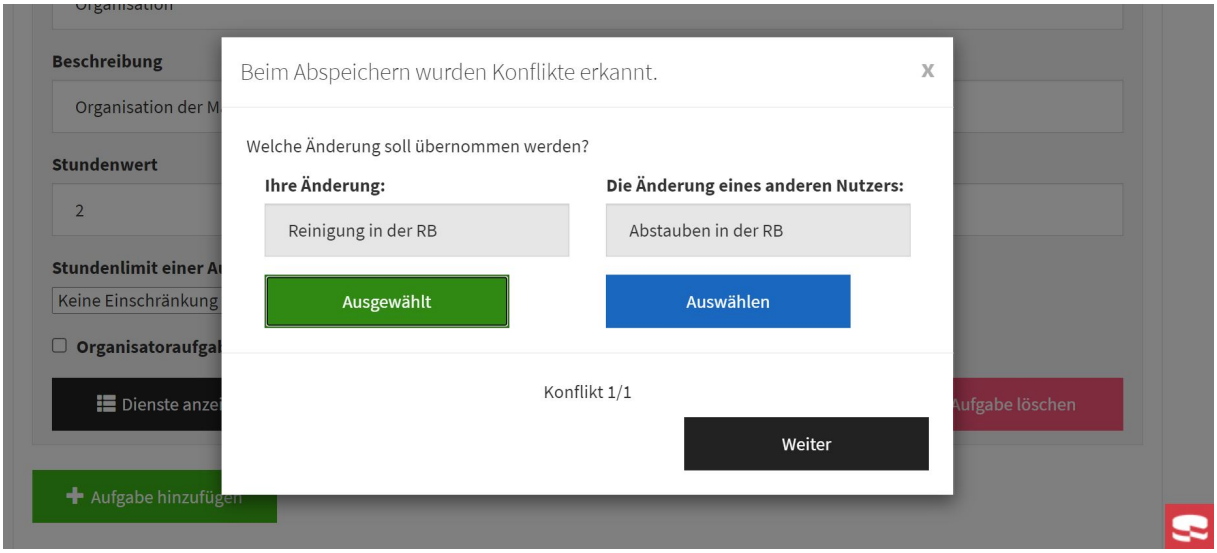

Abbildung 19: Edge-Case-Modal

6.4.4.10 Anpassungen für Veranstaltung-Erstellen-Dialog

Wie bereits zu Beginn des Abschnitts 6.4.4 erwähnt, müssen für den Veranstaltung-Erstellen-Dialog noch ein paar Anpassungen getroffen werden. Der Dialog benötigt seine eigene Funktion im Controller, sowie ein Template-File, in der React eingebunden wird. Anstatt auf ein bereits vorhandenes Event aus der Datenbank zuzugreifen, wird im Controller eine neue Instanz eines Events erstellt. Abgesehen davon, funktioniert der Veranstaltung-Erstellen-Dialog auf die gleiche Weise wie der Veranstaltung-Bearbeiten-Dialog. Es müssen ebenfalls vorhandene Arbeitsgruppen und Limitierungen im Controller zusammengestellt und an React weitergereicht werden, sodass die Select-Felder mit den richtigen Auswahlmöglichkeiten befüllt werden können. Außerdem werden die gleichen Components wie beim Veranstaltung-Bearbeiten-Dialog verwendet.

6.4.5 Spezifikation von Unit Tests

Für das Testen des Event-Editors wurden zunächst drei wichtige Themengebiete ausgewählt, zu denen Tests geschrieben wurden. Bei diesen drei Themengebieten handelt es sich um das korrekte Anzeigen der Änderungsanfragen, das Buchen von Diensten im Zusammenhang mit der eingestellten Arbeitsgruppe, sowie das richtige Anzeigen und Aktualisieren des Stundenwert-Eingabefeldes.

Beim Themengebiet Änderungsanfragen kann beispielsweise überprüft werden, ob der `ChangeRequest`-Component innerhalb des `Booking`-Component angezeigt wird, wenn es

eine Änderungsanfrage gibt. Des Weiteren kann getestet werden, ob der `ChangeRequest` die richtige Überschrift sowie den zutreffenden Status in der richtigen Farbe aufweist. Zum Thema Buchung von Diensten gilt es zum Beispiel zu prüfen, ob das Arbeitsgruppen-Select-Feld gesperrt wird, wenn die Veranstaltung mindestens eine Buchung aufweist oder ob die zur Auswahl stehenden Nutzer der ausgewählten Arbeitsgruppe entsprechen. Im Zusammenhang mit dem Stundenwert-Input kann unter anderem inspiziert werden, ob das Eingabefeld zum richtigen Zeitpunkt an der richtigen Stelle gerendert wird. Darüber hinaus wäre das korrekte Initialisieren und Aktualisieren der Werte im Store ein weiterer, interessanter Testgegenstand in Bezug auf den Stundenwert-Input. Im Anhang (siehe Tabelle 1, Tabelle 2) werden alle Tests aufgeführt, die im Zuge des Praxisprojektes geschrieben worden sind.

Für das Testen wird neben Jest (siehe 5.2.1) die React Testing Library (siehe 5.2.2) verwendet. Um mit dem Schreiben von Tests beginnen zu können, werden zwei externe Dateien benötigt. Zum einen muss eine vereinfachte Kopie des Redux Stores vorliegen, den man als initialen State für die Tests importieren kann. Wenn die Anwendung von einem Provider umschlossen wird, wird außerdem eine angepasste `render`-Methode benötigt. Ein Provider wird in Applikationen verwendet, die mit Redux arbeiten. Der Provider sorgt dafür, dass der Store für jeden Component zugänglich ist. Unter Einbeziehung dieser beiden externen Files können die eigentlichen Tests verfasst werden.

Es können zwei verschiedene Arten von Tests unterschieden werden: Component-Tests und Reducer-Tests. Bei einem Component-Test wird typischerweise der Component gerendert. Daraufhin wird beispielsweise ein Click-Event ausgeführt. Dann wird überprüft, ob ein bestimmtes DOM-Element vorhanden ist, beziehungsweise ob es eine bestimmte Eigenschaft aufweist.

Als Beispiel kann folgender Test betrachtet werden:

```
1.      it("the user you click on should be displayed under 'Ausgewählter Be-
    nutzer'", () => {
2.          initialTestState.UI.data.serviceId = 5;
3.          initialTestState.UI.data.taskId = 22;
4.          renderWithStore(<UserSelection />, { initialState: initialTestState
    });
5.          // get li element
6.          const listItems = screen.getAllByRole("listitem");
7.          const listItemAnke = listItems.find(item => item.textContent ===
    "Brenner, Anke (Monte Brenner 2015-08-01)");
8.          const listItemLena = listItems.find(item => item.textContent ===
    "Meier, Lena (Susi Meier 2013-08-14)");
9.
10.         // click on li element
11.         fireEvent.click(listItemAnke);
12.
13.         const span = screen.getByTestId("selected-user");
14.         expect(span).toHaveTextContent(
15.             "Brenner, Anke (Monte Brenner 2015-08-01)"
16.         );
17.
18.         fireEvent.click(listItemLena);
19.         expect(span).toHaveTextContent("Meier, Lena (Susi Meier 2013-08-
    14)");
20.     });
```

Zunächst werden wichtige Ergänzungen an den Daten im initialen State vorgenommen, bevor der `UserSelection`-Component gerendert wird. Danach werden zwei verschiedene Listenelemente auf der Seite gefunden. Eines davon wird angeklickt. Danach wird kontrolliert, ob das mittels Test-ID gefundene `span`-Element den richtigen Text anzeigt. Dasselbe wird auch bei dem anderen Listen-Element durchgeführt und überprüft.

Reducer-Tests sind eine andere Art von Tests. Sie agieren von React losgelöst. Es wird ein initialer State festgelegt und eine Action dazugegeben. Das Ergebnis wird dann mit einer Erwartung über den aktualisierten State verglichen. Ein Beispiel dafür stellt der nachfolgende Test dar:

```
1.      it("addService should affect servicesReducer and tasksReducer correctly",
    () => {
2.          expect(
3.              rootReducer(initialTestState, {
4.                  type: types.ADD_SERVICE,
5.                  taskId: 22,
6.                  id: "NEW_SERVICE_ID",
7.              })
8.          ).toEqual({
9.              ...initialTestState,
10.             //new service Id must be added to the array
11.             tasks: {
12.                 ...initialTestState.tasks,
13.                 data: {
14.                     ...initialTestState.tasks.data,
15.                     22: {
16.                         ...initialTestState.tasks.data[22],
17.                         services: [5, 6, "NEW_SERVICE_ID"],
18.                     },
19.                 },
20.             },
21.             services: {
22.                 //one new service must be added
23.                 ...initialTestState.services,
24.                 NEW_SERVICE_ID: {
25.                     id: "NEW_SERVICE_ID",
26.                     task_id: 22,
27.                     servicetype_id: null,
28.                     name: "",
29.                     value: "",
30.                     controlsEvent: false,
31.                     servicetype: null,
32.                     works: [],
33.                     containsChangeRequest: false,
34.                     userId: null,
35.                     isNew: true,
36.                 },
37.             },
38.         });
39.     });
```

Dem Root-Reducer, der die Kombination aus allen Reducern darstellt, wird die Action `addService` übergeben. Dann wird das zu erwartende Ergebnis aufgeschrieben. Darunter fallen alle Daten, die der Root-Reducer nach der Ausführung der Action beinhalten sollte.
Dann wird überprüft, ob die prognostizierte Struktur des Reducers mit dem tatsächlichen Ergebnis übereinstimmt.

6.5 Ergebnis

Nach Abschluss des Praxisprojektes liegt ein funktionstüchtiger Veranstaltungseditor vor, der alle Funktionalitäten der Ausgangsversion besitzt. Lediglich das Abfangen von Edge Cases, die auftreten können, wenn zwei Administratoren eine Veranstaltung gleichzeitig bearbeiten, ist zusätzlich implementiert worden (siehe 6.4.4.9). Schlussendlich wurden mehr Components erstellt, als ursprünglich eingeplant waren.

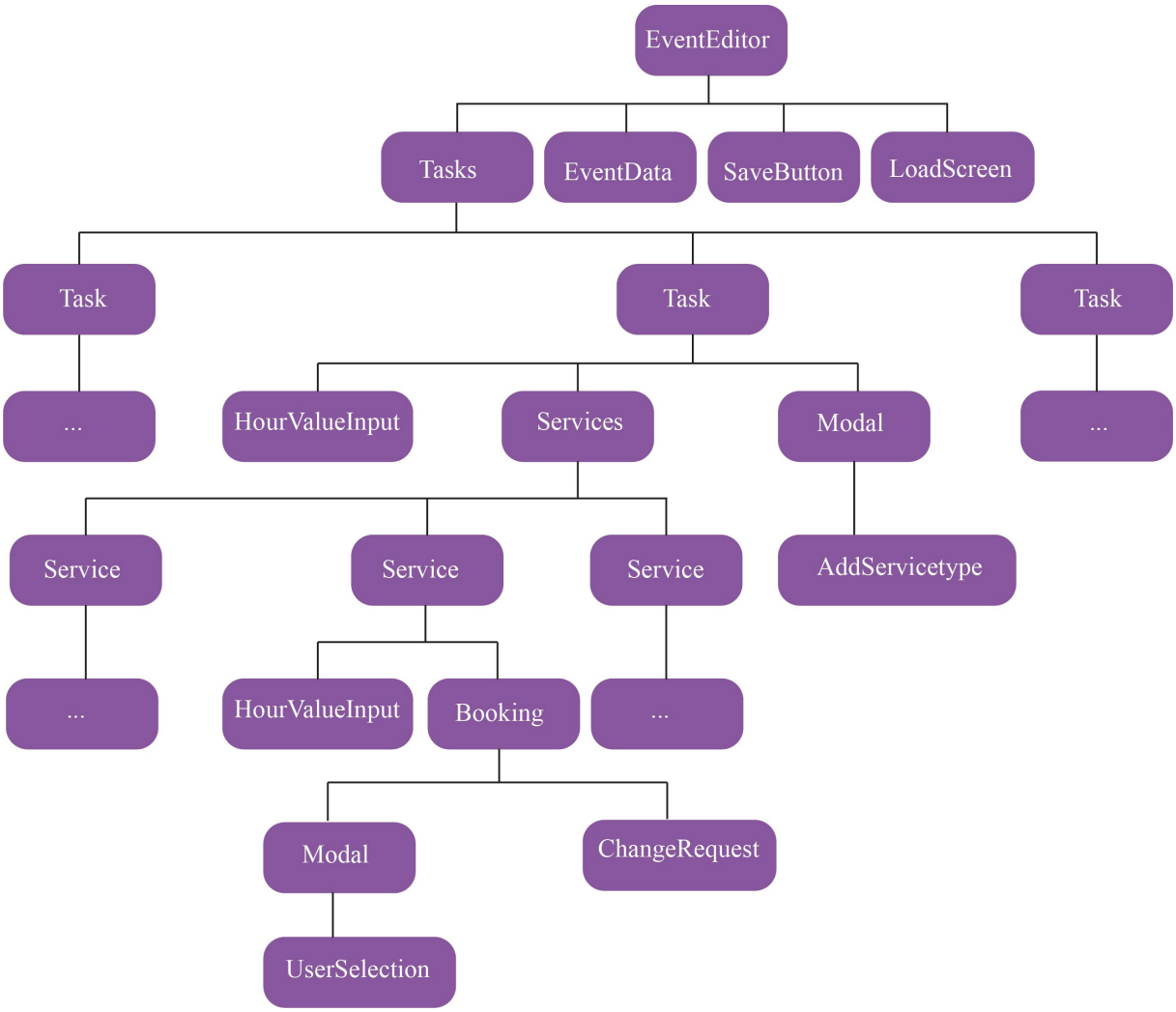

Abbildung 20: aktualisierter Component-Tree

In Abbildung 20 werden aus Übersichtlichkeitsgründen nicht alle Components inkludiert. Trotzdem kann man einige wichtige Ergänzungen im Vergleich zur Ausgangsversion erkennen. Direkt unter dem Root-Component sind ein `SaveButton`-Component sowie ein `LoadScreen`-Component hinzugekommen. Beispielhaft werden zwei wichtige Modals dargestellt: das Benutzerauswahl-Modal und das Modal zum Erstellen von Limitierungen. Sie befinden sich auf unterschiedlichen Ebenen des Component-Trees und werden nicht, wie anfangs

vermutet, auf der Ebene direkt unter dem Root-Component eingegliedert. In der Anwendung kommen noch weitere Modals zum Einsatz, die aber nicht berücksichtigt wurden, um die Übersichtlichkeit der Grafik zu wahren. Des Weiteren erhielt das Stundenwert-Eingabefeld seinen eigenen Component und wird je nach Status der Benutzeraufgaben-Checkbox entweder im `Service`- oder im `Task`-Component inkludiert.

Ein ausgewählter Teil besonders wichtiger Funktionalität ist mittels Unit Tests abgesichert. Insgesamt sind es 36 Tests, die alle mit einem positiven Ergebnis abschließen. Das Einbauen neuer Fehler würde sofort erkannt werden, wenn ein Test fehlschlägt. Durch die Tests kann man darauf vertrauen, dass der Code das macht, was von ihm erwartet wird.

Im Vergleich zu vorher ist der Code übersichtlicher. Das liegt unter anderem am Einsatz modularer, wiederverwendbarer Components. Dadurch gibt es zwar eine größere Anzahl an Dateien für das User Interface, diese sind dafür in sich kompakter und leichter zu überschauen. Zudem ist der Code leichter verständlich. Zur Verständlichkeit tragen die wiederkehrenden Konzepte von React bei, wie zum Beispiel das globale State-Management mit Redux, das Mappen eines Arrays auf Components oder andere HTML-Elemente, sowie die einfach zu lesende Spracherweiterung JSX. Das Beste an der neuen Version des Veranstaltungseditors ist allerdings, dass er leicht erweiterbar ist. Egal nach welcher Änderung die zukünftigen Monte-Tool-Anwender verlangen, sie wird relativ einfach in die Anwendung integriert werden können. Beispielsweise ist es ein Leichtes, ein weiteres Modal einzubauen oder weitere Eingabefelder, die von der bereits implementierten Validierung überprüft werden können.

7 Fazit

Diese Arbeit behandelte die Forschungsfrage „Ist ein Refactoring mithilfe moderner Webtechnologien geeignet, um die Wartbarkeit einer Webanwendung zu verbessern?“. Um diese Frage beantworten zu können, wurde bei dem Veranstaltungseditor des Monte-Tools ein Rewrite unter Einbeziehung der JavaScript Bibliothek React durchgeführt.

Zusammenfassend lässt sich sagen, dass die Wartbarkeit einer Anwendung ein wichtiger Einflussfaktor der Qualität von Software ist, um den sich fortlaufend gekümmert werden muss. Beim Veranstaltungseditor des Monte-Tools gab es Probleme im Bereich der Wartbarkeit. Lange Files mit kompliziertem, schwer erweiterbarem Code stellten die Firma Evidentmedia vor Herausforderungen. Durch geeignete Wartungstechniken, wie dem Refactoring oder der Integration einer modernen Webtechnologie, kann Einfluss auf die Wartbarkeit genommen werden. Wenn man von den Vorteilen einer modernen Webtechnologie profitieren möchte, wie beispielsweise von der erheblichen Zeitersparnis durch vorgefertigte Funktionen und Konzepte

bleibt keine andere Wahl als vom klassischen Refactoring abzuweichen und stattdessen einen Rewrite durchzuführen, bei dem von Beginn an die Strukturvorgaben der Technologie berücksichtigt werden können. Trotzdem will dieser Schritt wohl überlegt sein, da bereits namhafte Firmen bei derartigen Vorhaben gescheitert sind (Spolsky, 2000).

Bei der Auswahl einer passenden modernen Webtechnologie fiel die Entscheidung auf die in den letzten Jahren immer populärer werdende JavaScript Bibliothek React. Diese liefert einige, der Wartbarkeit dienliche Konzepte. Zum einen werden übersichtliche, klar abgrenzbare Components geschrieben, die beliebig oft wiederverwendet werden können. Auch die Spracherweiterung JSX fördert die Verständlichkeit durch seine gute Lesbarkeit und Kürze.

Durch den Einsatz von React konnte außerdem das Problem der Vermischung von konzeptionellen Daten und Anzeige behoben werden. Der Store bietet die Möglichkeit, einfach auf benötigte Daten zuzugreifen und diese anzupassen, ohne sich auf die Position im DOM oder CSS-Eigenschaften zu beziehen. Auch das stellt eine Verbesserung der Wartbarkeit dar, denn Änderungen des CSS oder des DOM können dadurch leichter implementiert werden ohne unerwünschte Nebeneffekte hervorzurufen. Durch die Implementation von Unit Tests und das regelmäßige Ausführen von diesen kann außerdem der Einbau neuer Fehler verhindert werden.

Diese positiven Entwicklungen machen deutlich, dass sich die Wartbarkeit des Monte-Tool-Veranstaltungseditors im Vergleich zur Vorgängerversion verbessert hat und ein Refactoring mithilfe einer modernen Webtechnologie demzufolge geeignet ist, um die Wartbarkeit einer Webanwendung zu verbessern.

Im Hinblick auf die Zukunft ist anzumerken, dass sich das volle Potential der neu erlangten, verbesserten Wartbarkeit noch unter Beweis stellen muss: in welchem Maße der Grad der Wartbarkeit gesteigert werden konnte wird sich bei der täglichen Arbeit des Evidentmedia-Teams bei der Weiterentwicklung des Veranstaltungseditors noch zeigen. Außerdem schadet es nicht, zusätzliche Unit Tests zu implementieren, um die Quelltextabdeckung noch auszuweiten.

Ferner sind während der Umsetzung des Praxisprojektes weitere Verbesserungspotentiale des Veranstaltungseditors zum Vorschein gekommen. Hierzu zählt die Implementation einer Suchfunktion, mit der konkrete Buchungen bestimmter Nutzer aus dem gesamten Formular gefiltert werden können. Außerdem könnte über ein neues Design des Veranstaltungseditors nachgedacht werden, bei welchem die Buchungen in einer zusätzlichen Spalte neben den Diensten angezeigt werden, anstatt darunter. Zum dritten besteht noch ein Verbesserungspotential im Bereich der Barrierefreiheit wodurch, auch Menschen mit Screenreadern, der Zugang zur Anwendung ermöglicht werden könnte.

Doch egal welche Herausforderungen zukünftig auf die Anwendung zukommen werden, die Anpassungen werden durch die vollzogene Verbesserung der Wartbarkeit leichter fallen als noch bei der Vorgängerversion.

8 Literaturverzeichnis

Ammelburger, D. & Scherer, R. (2009). *Webentwicklung mit CakePHP: Rapid Web Development mit dem MVC-Framework* (1. Aufl. korrigiert). O'Reilly. *

Bommer, C., Spindler, M. & Barr, V. (2008). *Softwarewartung: Grundlagen, Management und Wartungstechniken* (1. Aufl.). dpunkt. *

Can I use… Support tables for HTML5, CSS3, etc. (o.D.). Abgerufen 21. Mai 2021, von https://caniuse.com/form-validation

Client-side form validation. (o.D.). MDN. Abgerufen 21. Mai 2021, von https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation#validating_forms_using_javascript

Eisenman, B. (2015). *Learning React Native*. O'Reilly. *

Fowler, M. (2004). *Refactoring: Wie Sie das Design vorhandener Software verbessern* (1. Aufl.). Addison-Wesley. *

Frameworks. (o. D.). TenMedia. Abgerufen 21. Mai 2021, von https://www.tenmedia.de/de/glossar/entwicklungsframeworks

Hartmann, N. & Zeigermann, O. (2020). *React: Grundlagen, fortgeschrittene Techniken und Praxistipps - mit TypeScript und Redux* (2. Aufl.). dpunkt. *

Holla, S. (o. D.). *Chapter 3 – Understanding ReactJS Components*. Abgerufen 21. Mai 2021, von https://www.techdiagonal.com/reactjs_courses/beginner/understanding-reactjs-components/

Hunt, A. & Thomas, D. (2004). *Unit-Tests mit JUnit: Pragmatisch programmieren*. Hanser. *

International Organization for Standardization (2011). *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models* (ISO/IEC 25010:2011)

JetBrains. (2017). *JavaScript in 2017 - The State of Developer Ecosystem by JetBrains*. Abgerufen 27. Mai 2021, von https://www.jetbrains.com/research/devecosystem-2017/javascript/

JetBrains. (2018). *JavaScript in 2018 - The State of Developer Ecosystem by JetBrains*. Abgerufen 27. Mai 2021, von https://www.jetbrains.com/research/devecosystem-2018/javascript/

JetBrains. (2019). *JavaScript 2019 - The state of Developer Ecosystem in 2019 Infographic*. Abgerufen 27. Mai 2021, von https://www.jetbrains.com/lp/devecosystem-2019/javascript/

JetBrains. (2020). *JavaScript Programmierung - Infografik zum Stand des Entwickler-Ökosystems 2020 | JetBrains: Developer Tools for Professionals and Teams*. Abgerufen 20. Mai 2021, von https://www.jetbrains.com/de-de/lp/devecosystem-2020/javascript/

Liste der php.ini-Direktiven. (o.D.). Abgerufen 21. Mai 2021, von https://www.php.net/manual/de/ini.list.php

Martin, R. C. (2009). *Clean Code: Refactoring, Patterns, Testen und Techniken für sauberen Code* (1. Aufl.). mitp. *

Norbu, T. (2020, 16. Juni). *React Testing Library Vs. Enzyme*. Abgerufen 03. Mai 2021, von https://medium.com/wesionary-team/react-testing-library-vs-enzyme-afd29db380ac

Petereit, D. (2016, 09. Juli). *Wozu Frameworks gut sind*. Abgerufen 03. März 2021, von https://t3n.de/news/wozu-frameworks-gut-721842/

Petereit, D. (2017, 24. März). *Für jede Kleinigkeit ein Framework: Warum immer mehr Entwickler von Javascript genervt sind*. Abgerufen 03. März 2021, von https://t3n.de/news/fuer-kleinigkeit-framework-immer-808303/

Reimers, S. & Thies, G. (2010). *PHP 5.3 & MySQL 5.5: das umfassende Handbuch* (3. aktualisierte und erw. Aufl.). Galileo Press. *

Roock, S. & Lippert, M. (2004). *Refactorings in großen Softwareprojekten: Komplexe Restrukturierungen erfolgreich durchführen* (1. Aufl.). dpunkt. *

Schieferbein, A. (2018). *Reaktive JavaScript Frameworks - Trend in der Entwicklung von Benutzeroberflächen*. Abgerufen 25. Mai 2021, von https://www.kreitiv.de/reaktive-javascript-frameworks/

Schwarzmüller, M. (2019, 19. Februar). *Redux vs React's Context API*. Abgerufen 25. Mai 2021, von https://academind.com/tutorials/reactjs-redux-vs-context-api/

Spolsky, J. (2000, 06. April). *Things You Should Never Do, Part I*. Abgerufen 25. Mai 2021, von https://www.joelonsoftware.com/2000/04/06/things-you-should-never-do-part-i/

Springer, S. (2020). *React: Das umfassende Handbuch* (1. Aufl.). Rheinwerk Computing. *

Stefanov, S. (2017). *Durchstarten mit React: Web-Apps einfach und modular entwickeln* (1. Aufl.). dpunkt. *

Das Use-Case-Diagramm (Anwendungsfalldiagramm) in UML. (2020, 24. März). Ionos. Abgerufen 25. Mai 2021, von https://www.ionos.de/digitalguide/websites/web-entwicklung/anwendungsfalldiagramm/

Visser, J. (2016). *Building maintainable software: Ten guidelines for future-proof code* (1. Aufl.). O'Reilly. *

Wießeckel, T. (2014, 03. Juni). *Frameworks oder nicht – das ist hier die Frage*. Abgerufen 03. März 2021, von https://entwickler.de/online/frameworks-oder-nicht-das-ist-hier-die-frage-143605.html

Yamazaki, S. (2020, 18. August). *Understanding Functional Components vs. Class Components in React*. Abgerufen 25. Mai 2021, von https://www.twilio.com/blog/react-choose-functional-components

Zammetti, F. (2020). *Modern Full-Stack Development: Using TypeScript, React, Node.js, Webpack, and Docker* (1. Aufl.). Apress. *

Anhang

Anhangsverzeichnis

Anhang 1: Digitaler Anhang 64
Anhang 2: Testgegenstände 65
Anhang 2.1: Component-Tests 65
Anhang 2.2 Reducer-Tests 67

Anhang 1: Digitaler Anhang

An dieser Stelle befindet sich eine SD-Karte mit folgenden Daten:

- Eine Anleitung zum Einrichten des Monte-Tools
- Eine SQL-Datei einer Demo Datenbank
- Der Ordner monte_rework mit allen benötigten Code-Dateien
- Ein Ordner mit allen verwendeten Internetquellen
- Ein Ordner mit den Screenshots der Anwendung
- Diese Arbeit als PDF-Datei

Anhang 2: Testgegenstände

Anhang 2.1: Component-Tests

Tabelle 1: Component-Tests

Nr.	*Beschreibung des Testgegenstandes*	*Getesteter Ausschnitt der Anwendung*
1	should render Change Request if containsChangeRequest is true	Booking-Component
2	should be the headline “Beantragte Änderung” if accepted = null	ChangeRequest-Component
3	should be the headline “Abgelehnte Änderung” if accepted = false	ChangeRequest-Component
4	should be the headline “Bestätigte Änderung” if accepted = true	ChangeRequest-Component
5	should have the status “in Bearbeitung” if accepted = null	ChangeRequest-Component
6	should have the status “bestätigt” written in #3fb618 (green color) if accepted = true	ChangeRequest-Component
7	should have the status “abgelehnt” written in red if accepted = false	ChangeRequest-Component
8	hourValue should have a plus sign and a green color (#3fb618) if the value is greater than 0	ChangeRequest-Component
9	hourValue should have a plus sign and a green color (#3fb618) if the value is 0	ChangeRequest-Component
10	hourValue should have a minus sign and a red color if the value is smaller than 0	ChangeRequest-Component
11	workgroup select should be enabled if there is no booking	EventData-Component
12	workgroup select should be disabled if there is at least one booking	EventData-Component

13	should render hourValueInput if userclaimable is true	Service-Component
14	should not render hourValueInput if userclaimable is false	Service-Component
15	should NOT disable the delete Service Btn if there are two Services left without works	Service-Component
16	should disable the delete Service Btn if the service has a work even if there is more than one service for the task	Service-Component
17	should disable the delete Service Btn if there is ONLY one service left	Service-Component
18	should render hourValueInput if userclaimable is false	Task-Component
19	should not render hourValueInput if userclaimable is true	Task-Component
20	should render Message if there is a changeRequest	Task-Component
21	should NOT render Message if there is NO changeRequest	Task-Component
22	should toggle the controlsEvent checkbox when clicking on it	Task-Component
23	in the userSelection should only be users with the matching workgroup	UserSelection-Component
24	should only display the user searched for	UserSelection-Component
25	the user you click on should be displayed under “Ausgewählter Benutzer”	UserSelection-Component

Anhang 2.2 Reducer-Tests

Tabelle 2: Reducer-Tests

Nr.	*Beschreibung des Testgegenstandes*	*Getesteter Ausschnitt der Anwendung*
26	if userclaimable = true, onChangeHourValue should update service value but not task value	Action onChangeHourValue im rootReducer
27	if userclaimable = false, onChangeHourValue should update task value and all related service values	Action onChangeHourValue im rootReducer
28	bookService	Action bookService im rootReducer
29	deleteBooking	Action deleteBooking im rootReducer
30	should set the initial service values equal to the task value if !userclaimable	Action setInvisibleServiceValues im servicesReducer
31	Should NOT set the initial service values equal to the task value if userclaimable	Action setInvisibleServiceValues im servicesReducer
32	copyTask	Action copyTask im rootReducer
33	addTask	Action addTask im rootReducer
34	copyService	Action copyService im rootReducer
35	addService	Action addService im rootReducer
36	updateUsers	Action updateUsers im usersReducer