Fachhochschule Aachen
Campus Jülich

Seminararbeit

Single-Page-Applikationen mit den Frameworks React und Redux

---

vorgelegt von Leon Mallmes

Jülich, den 3. Januar 2022

Fachbereich:	Medizintechnik und Technomathematik
Studiengang:	Angewandte Mathematik und Informatik
Matrikelnummer:	3232073

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Seminararbeit mit dem Thema:

_____

_____

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war. Ich verpflichte mich, ein Exemplar der Seminararbeit fünf Jahre aufzubewahren und auf Verlangen dem Prüfungsamt des Fachbereiches Medizintechnik und Technomathematik auszuhändigen.

_____ _____

(Ort, Datum) (Unterschrift des Verfassers)

Diese Arbeit wurde betreut von:

1. Prüfer:	Prof. Dr. rer. nat. Volker Sander
2. Prüfer:	Dipl. Phys. Ing. Waldemar Hinz

Sie wurde angefertigt in der Forschungszentrum Jülich GmbH in der Zentralbibliothek (ZB)

Zusammenfassung

Die Gesellschaft verlangt nicht nur immer schnellere Internetseiten, sondern beispielsweise auch Applikationen, die auf allen Geräten von Desktop-Rechnern bis hin zu mobilen Geräten, eine ähnliche Darstellung, sprich Benutzerführung aufweisen.
Dabei geht die moderne Web-Entwicklung auch immer mehr in Richtung einfacher Seiten und simpler Applikationen. Die Hauptziele bestehen daraus, die Effizienz zu steigern, schneller Antworten zurückgeben zu können und einfache Handhabung zu gewähren. Eine Möglichkeit, diese Ansichten und Ziele zu vereinen, wären die Single-Page-Applikationen(SPA).
SPA sind Webanwendungen und zeichnen sich durch zwei Merkmale besonders aus. Einerseits bestehen diese nur aus einem HTML-Dokument, andererseits können in diesem Dokument auch Inhalte dynamisch nachgeladen werden.
Dabei wird das JavaScript-Framework React häufig als Basis für eine solche Anwendung benutzt. Einen Vorteil bildet dabei die Gestaltung und das generelle Ausschauen, da dies für alle möglichen Geräte optimiert ist. Der Kern von React ist dabei die eigentliche Komponente. Durch Java-Script kommt außerdem die Möglichkeit hinzu, einen Offline-Modus zu ermöglichen.
Für das Modell React sprechen dabei nicht nur ein einseitiger Datenfluss, sondern der dennoch starke Performance-Aufbau, besonders auch bei komplexeren Anwendungen. Dabei spielt der unidirektionale Datenfluss, als Kernkonzept von React, eine wichtige Rolle, da dieser durch seine Einfachheit, den Aufbau und die Wechselwirkungen der verschiedenen Komponenten vereinfachen soll.
Bei dem vielen Vorteilen, die React mit sich bringt, fällt allerdings auch ein Thema auf, welches zu Schwierigkeiten führen kann, wenn man sich nur auf React als Framework konzentriert. Die Zustandsverwaltung von Daten ist auch in der Komponente verankert. Als Folge führt dies zu komplexeren Anwendungen und der Quellcode ist undurchsichtig. Abhilfe kann eine zentrale Zustandsverwaltung schaffen.
Aufgrund dieser Zustandsverwaltung ist es also sinnvoll React in Verbindung mit dem Framework Redux zu benutzen. Dabei hilft Redux nicht nur dabei, alle Zustandsinformationen zusammenzuhalten, sondern verfolgt außerdem das Ziel, diese für alle Komponenten der Webanwendung zugänglich zu machen.
In der Zentralbibliothek (ZB) des Forschungszentrums werden verschiedene
Web-Applikationen, die auf einer breiten Auswahl von Geräten laufen sollen, entwickelt. Dabei sind die Wissenschaftler des Forschungszentrums die Hauptnutzer, da diese Zugang zu ihrer Literatur auf verschiedenen Geräten haben möchten. Um dies nutzbar zu machen, sollte die Bedienung der Anwendungen auch möglichst ähnlich und simpel sein.

Zuerst wird sich in dieser Arbeit genauer mit den Frameworks React und Redux auseinandergesetzt. Außerdem sollen durch eine Beispielanwendung die Frameworks React und Redux darauf untersucht werden, ob man dieses Gespann sinnvoll für die Entwicklung von Anwendungen mit ähnlicher Benutzerführung verwenden kann.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung und Motivation

Im gesamten Forschungszentrum und somit auch in der Zentralbibliothek haben die Mitarbeiter Zugriff auf das forschungszentrums-interne Intranet. Dies stellt intern nutzbare Webapplikationen dar, über die sowohl Informationen zum Forschungszentrum geteilt werden, als auch interne Dienste angeboten werden, wie zum Beispiel das Mitarbeiterportal oder auch das Management von Zertifikaten. Allerdings stellt die ZB nicht nur Dienste innerhalb des Forschungszentrums zur Verfügung, sondern auch einige Webapplikationen im World Wide Web. Grundsätzlich ist aber irrelevant, ob eine Website intern oder extern angelegt werden soll, denn die wichtigsten Aspekte spiegeln sich grundsätzlich in der Performance und in der Laufzeit einer Webapplikation wieder. Um dies zu erreichen, wird immer wieder gerne auf das Konzept der SPA's mit dynamisch nachladbaren Inhalten zurückgegriffen. Die Technik entwickelt sich stetig weiter und somit kommt es nicht nur auf die Effizienz alleine, sondern auch auf die Flexibilität der Nutzung an. Das bedeutet, dass man nicht nur von seinem Arbeitsrechner auf verschiedene Applikationen zugreifen will, sondern unter Umständen auch die Möglichkeit haben möchte, dieselbe Applikation auf einer breiteren Auswahl an Geräten aufrufen zu können. Dazu trägt zusätzlich die Corona-Pandemie bei, die auch im Forschungszentrum für eine große Ausweitung des Home-Office Konzepts gesorgt hat. Somit waren viele Mitarbeiter gezwungen, bei ihrer Arbeit viele unterschiedliche Geräte zu nutzen. Dabei ist es besonders wichtig, dass dieselbe Web-Anwendung sowohl auf dem Arbeitsrechner als auch dem kleineren Laptop, dem Tablet oder dem Handy genutzt werden kann. In der Entwicklung der SPA's kommen auch viele Frameworks zum Einsatz, um diese Entwicklung zu generalisieren. In besonderem Bezug auf die Punkte der Performance und der variablen Nutzung hat man sich darauf verständigt, die Frameworks React und Redux im Folgenden genauer zu betrachten und deren gemeinsamen Nutzen anschließend an einer Beispielapplikation zu analysieren. All das mit dem Ziel herauszufinden, ob diese Kombination wirklich die benötigten Voraussetzungen mitbringt, um auch für Anwendungen in der ZB genutzt zu werden. Bei Erfolg bestände beispielsweise Bedarf an einer, auf unterschiedlichen Geräten nutzbaren, Applikation von Julib eXtended als SPA[Jul], welche dann auch mobil laufen können soll.

Kapitel 2

Grundlagen

2.1 HTML

HTML ist die Kurzform für Hyper Text Markup Language und sorgt mithilfe einer textbasierten Beschreibung für die Strukturierung einer HTML-Datei. Mithilfe von Markup können einem Webbrowser Daten, wie zum Beispiel Texte, Hyperlinks oder Bilder, mitgeteilt und mithilfe einer Website angezeigt werden. [HTM]

HTML gibt es seit dem 3. November 1989 und wurde damals bei der Schweizer Forschungsorganisation, Europäische Organisation für Kernforschung (CERN) entwickelt, mit dem Ziel, gemeinsame Forschungsdaten mit dem Standort in Frankreich auszutauschen. Die erste Veröffentlichung fand dann im Jahr 1992 statt. Die aktuellste Version ist bekannt als HTML 5.2 und wurde am 14. Dezember 2017 veröffentlicht. Ein typisches HTML-Dokument besteht aus drei Strukturen. Dabei beginnt es mit einer Dokumenttyp-Deklaration, mit welcher man den genutzten HTML-Typ deklariert. Darauf folgt dann der HTML-Kopf (HEAD), der im Normalfall alle technischen Komponenten des Dokuments beinhaltet, und im Anschluss noch der HTML-Körper (BODY), in welchem der auf der Website darzustellende Inhalt, beispielsweise Texte und Bilder, verfasst wird. [Wik]

```
<html>
<head>
   <title>Einfaches Beispiel</title>
   <meta name="description" content="Diese Beschreibung wird nicht angezeigt,
kann aber im Google-Eintrag erscheinen."/>
</head>
<body>
  <h1>Das ist die Überschrift</h1>
  <p>Und das ist ein Absatz, der auch ein <strong>fettes</strong> Wort
enthält.</p>
  <p>Mehr dazu finden Sie auf <a href="https://de.wikipedia.org/wiki/
Hypertext_Markup_Language">Wikipedia</a></p>
</body>
</html>
```

Abbildung 2.1: Beispiel HTML-Struktur

2.2 Document Object Model (DOM)

Document Object Model (DOM) ist eine Programmierschnittstelle, welche vereinheitlicht für die Strukturierung von HTML- und XML-Dokumenten sorgt. Während die Erfindung von JavaScript selber dafür sorgte, dass nicht nur statische, sondern auch dynamische Webseiten im Browser erstellt werden konnten. Allerdings war diese Entwicklung abhängig von den jeweils entwickelten Interpretern und dynamischen Modellen, welche zwar dafür sorgten, dass gesamte JavaScript Potenzial zu nutzen, allerdings waren diese im Gesamten nicht aufeinander abgestimmt und somit sehr aufwendig. Um dies zu vermeiden und dem großen Aufwand entgegenzuwirken, wurden mit DOM erste Spezifikationen dafür entwickelt. Dieses Konzept wurde erstmals 1998 veröffentlicht und wird bis heute intensiv in der Entwicklung mit JavaScript genutzt. [DOMa]

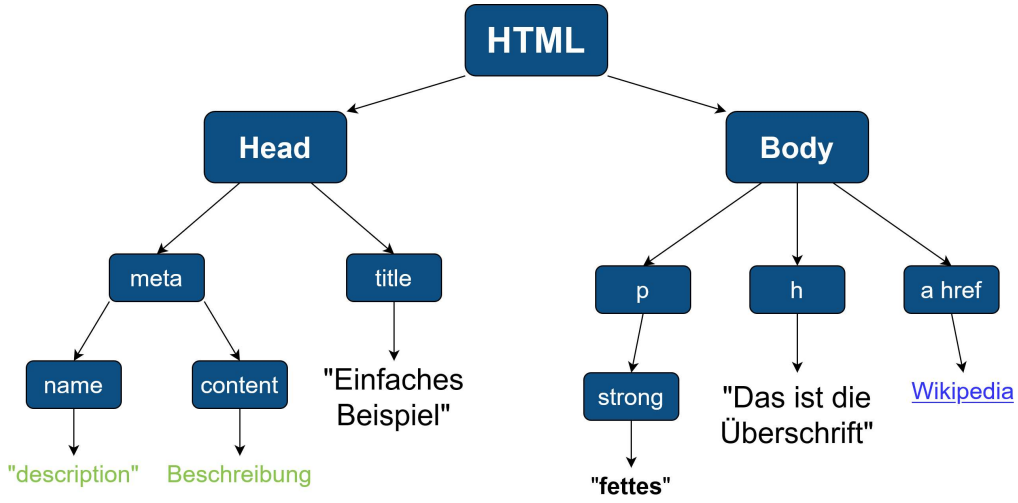

Abbildung 2.2: DOM-Baum zum HTML-Beispiel Abbildung 2.1

2.3 JavaScript und funktionale Programmierung

Die Programmiersprache JavaScript wurde wie sein Namensvetter Java im Jahr 1995 veröffentlicht und verbreitete sich seitdem ähnlich schnell. Mit Java selber ist es aber

nicht vergleichbar, da sich die beiden Sprachen dafür zu sehr unterscheiden. Aufgrund des schnellen Internetwachstums zu der Zeit verbreitete sich diese Internetsprache nicht nur extrem schnell, sondern gilt auch heute als eine der bekanntesten Sprachen und ist in jedem Browser vorzufinden.
JavaScript zeichnet sich besonders durch eine Vielseitigkeit von dynamischen Funktionen aus und ist außerdem variabel einsetzbar, da man mithilfe von JavaScript nicht nur objektorientiert, sondern auch funktional programmieren kann. Ein bekanntes Beispiel für die funktionale Programmierung stellt die Funktion höherer Ordnung *map* dar. Diese kann als Parameter zuvor definierte Funktionen sowie auch Lambda-Funktionen erhalten.

```
54  const myArray = [1, 2, 3];
55  const double = n => n * 2;
56  // Benutzung einer bereits definierten Funktion
57  myArray.map(double); // [2, 4, 6]
58  // Benutzung einer Lambda-funktion
59  myArray.map(n => n * 3); // [3, 6, 9]
```

Abbildung 2.3: Funktionale Programmierung mit *map* [Fun]

Des Weiteren kann JavaScript auch in HTML eingebunden werden und dafür den DOM durchsuchen und diesen zur Laufzeit manipulieren. Aufgrund dieser Manipulation rendert der Browser dann erneut und aktualisiert den manipulierten Inhalt. [Aug]
Darunter fallen beispielsweise auch einfache Aktionen wie ein Button-Click und eine darauf folgende Reaktion des Programms sowie auch eine damit verknüpfte Funktion, die dafür sorgt, dass auch auf der Webseite selber eine Ausgabe sichtbar wird.

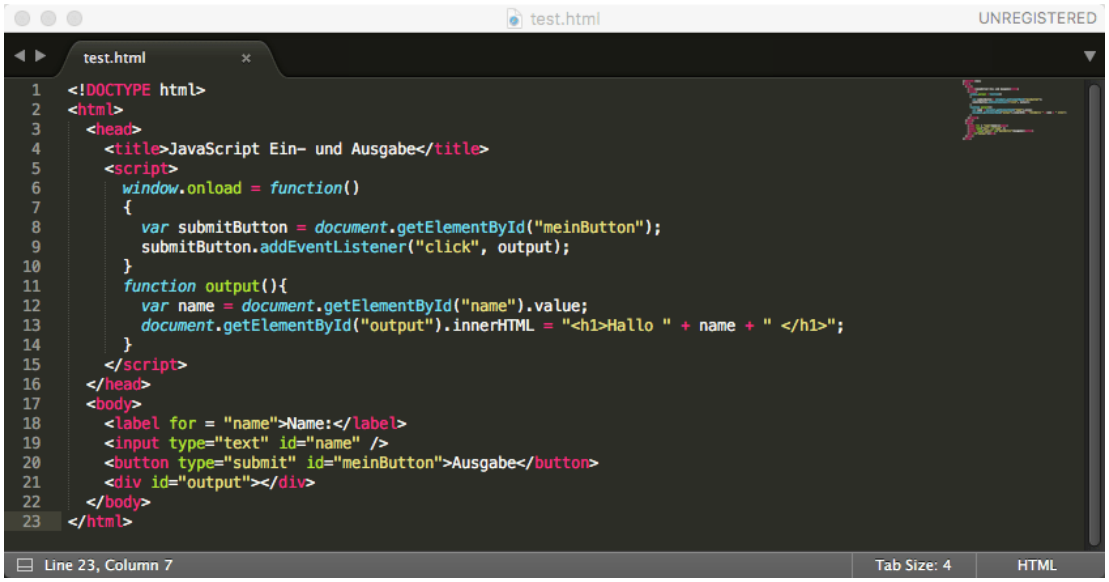

Abbildung 2.4: JavaScript-Button in HTML eingebunden

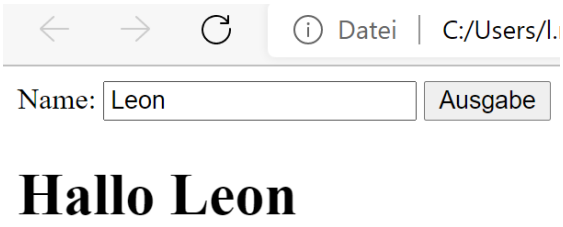

Abbildung 2.5: Beispiel-Eingabe und Ausgabe zum Beispielcode

2.4 HTTP-Protokoll und REST-API

Hypertext Transfer Protocol (HTTP) im deutschen auch Hypertext-Übertragungsprotokoll sorgt, wie der Name schon erkennen lässt, für die Übertragung von Daten, genauer Hypertext-Dokumenten. Meist wird dies verwendet, um solche Dokumente aus dem World Wide Web in einem Webbrowser neu zu laden. Die Transaktion des Protokolls besteht dabei nur aus kurzen Abläufen, welche sich aus verschiedenen Headern zusammensetzen. Die Header werden in Request, die Anfrage vom Client an den Server, Response, die Antwort vom Server und Entity Header, normalerweise GET-Request, unterschieden. Die gesamten HTTP-Methoden, werden anwendungstechnisch nochmal in verschiedene, sogenannte HTTP-Verben differenziert. Die HTTP-Verben bilden die Grundlage für die Architektur der REST-API. Diese werden in der nachfolgenden Tabelle noch einmal aufgelistet. [Spi19, S. 7-12]

HTTP Verben	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/id)
POST	Create	201 (Created), 'Location' header with link to /customers/id containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

Tabelle 2.1: HTTP-Verben [Ver]

2.5 AJAX und JSON

Asynchronous JavaScript and XML (AJAX) ist eine asynchrone Übertragung von Daten zwischen Client und Server. Die Datenübertragung bei AJAX funktioniert mittels eines HTTP-Request, wobei aber nur jeweils die Bereiche einer Seite nachgeladen werden, die auch aktualisiert werden müssen. AJAX basiert dabei auf der Sprache JavaScript und kann dadurch mit HTML in Webseiten eingebaut werden. JavaScript sorgt für die Funktionalität eines AJAX-Aufrufs, der beispielsweise in Form eines Button-Click-Events auftreten kann. Innerhalb eines AJAX-Aufrufs werden die HTTP-Verben angewendet, bevor der Aufruf dann an einen WWW-Server weitergeleitet und dort mithilfe einer Controller-Logik initialisiert und ausgewertet wird. [AJA]

Auch JavaScript Object Model (JSON), was mithilfe von JavaScript übersetzt und geparst werden kann, spielt in Kombination mit AJAX eine wichtige Rolle. [JSO] Die Interaktion zwischen der Anwendung und dem Web-Server, welche über HTTP-Anfragen und mithilfe der DOM-Manipulation durch AJAX gestaltet wird, wird in der nächsten Abbildung noch einmal verbildlicht.

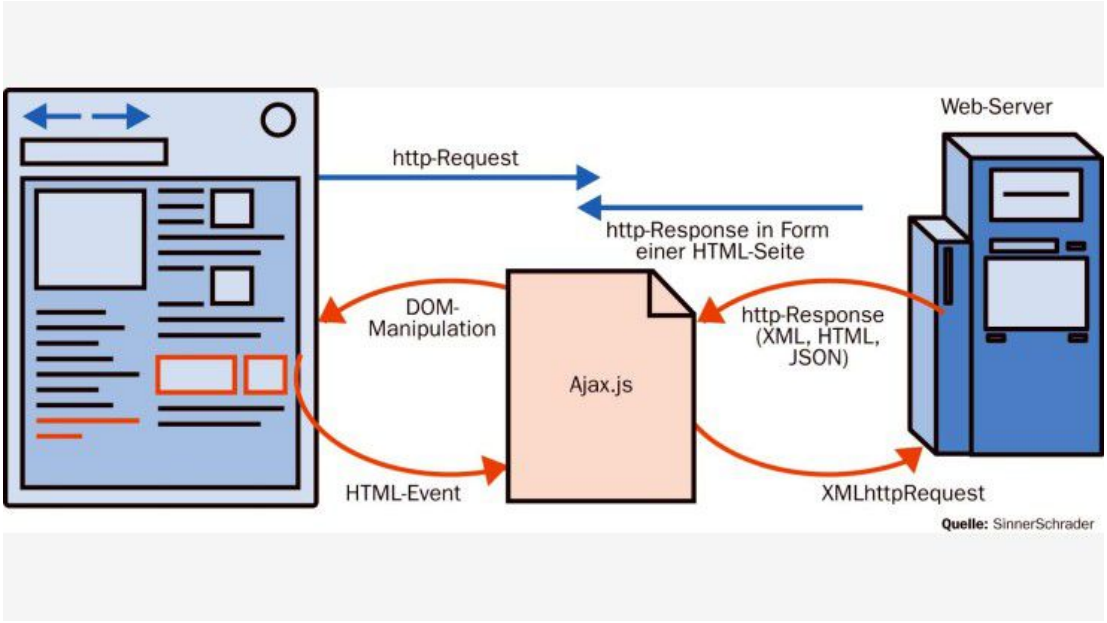

Abbildung 2.6: Konzept von AJAX [Hey]

2.6 Frameworks

Frameworks sind ähnlich wie API's, wobei ein Framework technisch gesehen eigentlich eine API beinhaltet. Im Grunde sind es Plattformen und dienen der Entwicklung von Softwareanwendungen. Dabei können Frameworks sowohl verschiedene Klassen und Funktionen umfassen, genauso wie auch gesamte Codebibliotheken oder Compiler. All diese werden dann für die Interaktion mit der Systemsoftware und für deren Entwicklungsprozesse genutzt. Das gesamte Konzept des dadurch definierten Entwicklungsprozesses unterstützt speziell die Programmierer, damit sie mithilfe von Frameworks nicht jedes Mal alles wieder erneut entwickeln müssen. [Fra]

Kapitel 3

Web-Anwendungen

3.1 Multiple Page Application (MPA)

Multiple Page Application (MPA) sind als traditionelle Webanwendungen bekannt und sorgen dafür, dass eine Seite komplett neu geladen wird, falls ein Nutzer mit dieser interagiert hat. Der Prozess von MPA's ist sehr zeitintensiv, indem bei jedem Aktualisierungsschritt eine neue Seite vom Server angefordert wird. Im Anschluss muss die Antwort an den Client zurückgegeben und im Browser neu gerendert werden. AJAX hat mit der Aktualisierung einzelner Komponenten zwar eine Zeitersparnis in das System eingebracht, allerdings wird das gesamte System dadurch nur noch viel komplexer, da an jede Seite der MPA eine Anfrage gesendet wird und die Komponente erst nach Beendigung dieses Prozesses aktualisiert werden kann. [Qua]

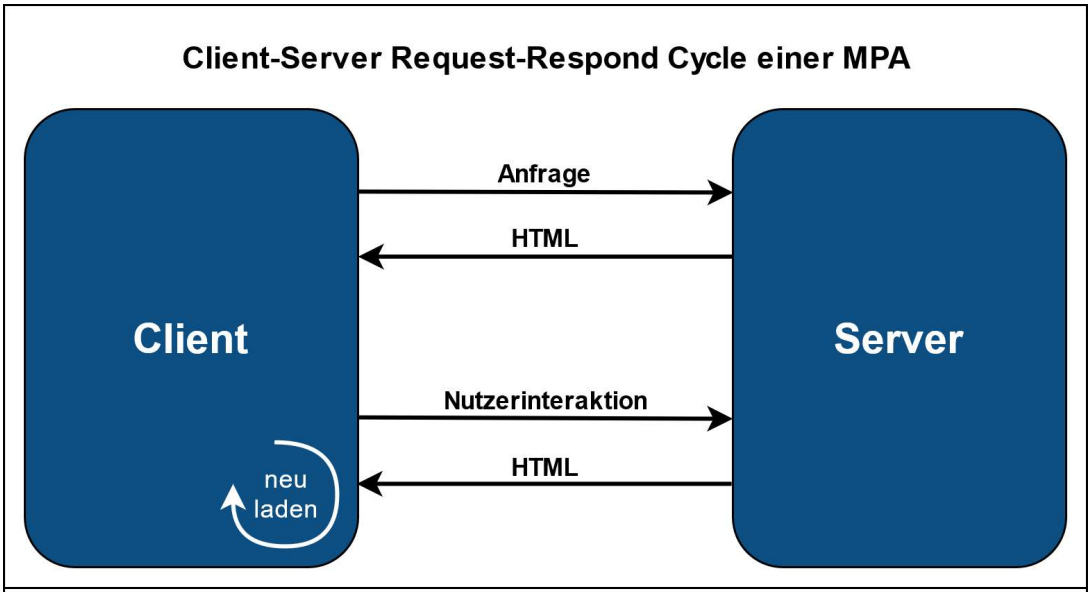

Abbildung 3.1: Klassisches MPA-Konzept

3.2 Single-Page-Applikationen (SPA)

Im Gegensatz zur klassischen MPA, setzt sich eine Single-Page-Applikation (SPA) nur aus einem einzigen HTML-Dokument zusammen. Aufgrund dessen bezeichnet man SPA's auch als Alternative zur klassischen MPA. Dabei bietet besonders das dynamische Nachladen des einen HTML-Dokuments einen Vorteil im Vergleich zur klassischen Form, bei welcher es in Bezug auf viele Nutzeranfragen deutlich schneller zu Problemen kommen kann, da dort gleich mehrere HTML-Dokumente, welche alle miteinander verlinkt sind, neu geladen werden müssen. Ein weiterer Vorteil ergibt sich durch die Möglichkeit auch offline durch die Applikation navigieren zu können. Dies erreicht man durch das Zwischenspeichern der Daten im Browser ('Web-Storage'). [SPA]

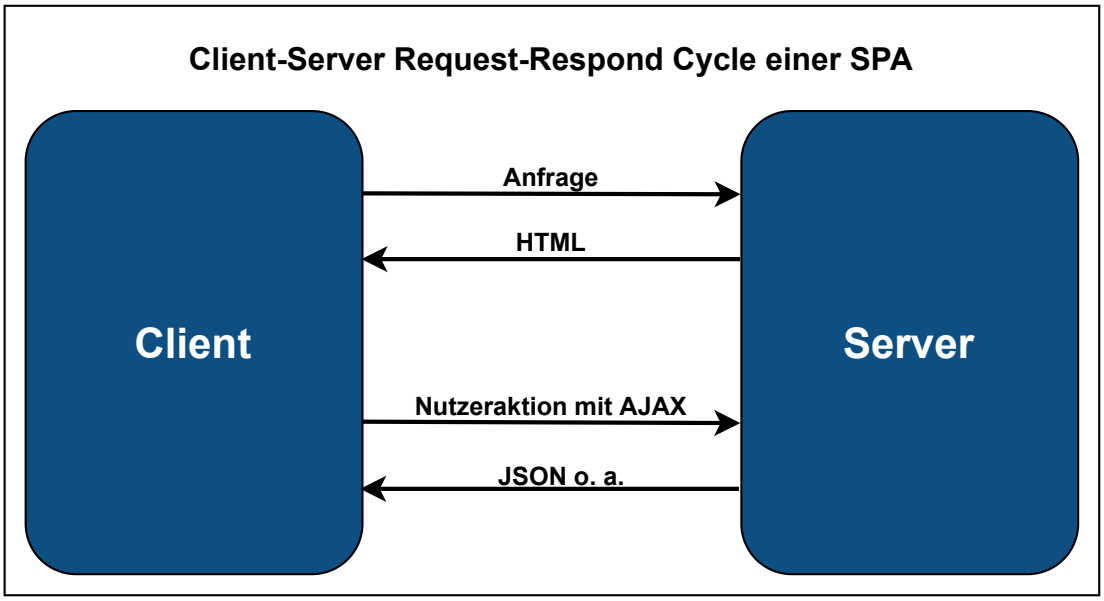

Abbildung 3.2: SPA-Konzept

3.3 MPA vs. SPA

Sowohl die klassischen MPA's als auch die moderneren SPA's haben verschiedene Vor- aber auch Nachteile. Die größten Vorteile der MPA's lassen sich bei den einfacheren Entwicklungsmöglichkeiten und den damit verbundenen unkomplizierten Optimierungsverfahren festmachen. Dies setzt sich daraus zusammen, dass bei der Entwicklung meist deutlich weniger Technologien als bei SPA's benötigt werden und man an den einzelnen, unterschiedlichen Seiten einer MPA auch jeweilige Tags und Optimierungen einbauen kann. Aber auch wenn die Optimierungsmöglichkeiten bei SPA's schwerer umsetzbar sind, ist besonders die Geschwindigkeit solcher Applikationen ein immenser Vorteil. Da die Seiten, wie bereits erklärt, nur dynamisch nachgeladen und nicht komplett neu ge-

laden werden müssen, sorgt dies für eine erheblich optimierte Arbeitsgeschwindigkeit. Dies wird auch um einiges durch die breite Nutzung vom kompakten JSON-Format bei der Kommunikation verstärkt. Besonders bezüglich mobiler Applikationen sind sie damit den MPA's weit voraus. Gesteigert wird diese Geschwindigkeit in der Entwicklung zusätzlich durch die bereitgestellten Werkzeuge in der Entwicklung, die in Form von bereitgestellten Bibliotheken und Frameworks, eine sehr effiziente und vor allem schnelle Entwicklung einer Applikation ermöglichen. Somit sind SPA's auch im Bereich Entwicklung im Vorteil.[MPA] Diese ganzen Vor- und Nachteile werden in der folgenden Tabelle Abbildung 4.4 noch belegt. Außerdem gibt es auch noch zwei Aspekte, in denen keine der beiden Applikationsarten von sich aus gewinnt, ohne zusätzliche Hilfen einzubinden.[Tab]

Charakteristisch	Gewinner
Geschwindigkeit und Leistung	**SPA** Dynamisches Laden von Inhalten eliminiert das Neuladen von Seiten und verkürzt die Ladezeiten.
Entwicklung	**SPA** Trotz des größeren Tech-Stack dauert das Entwickeln, Testen und Starten einer einseitigen Web-App viel weniger Zeit als das Entwickeln, Testen und Starten einer mehrseitigen App. Es ist nicht erforderlich, Code zu schreiben und eine Benutzeroberfläche für mehrere Seiten zu entwerfen.
Navigation	**MPA** Um eine SPA zu erstellen, in dem Benutzer einfach hin und her navigieren sowie Links zu einem bestimmten Ort auf der Website freigeben können, müssen Entwickler API's verwenden.
Skalierbarkeit	**MPA** MPA's sind unbegrenzt skalierbar, während Ihre Entwickler zum Skalieren eines SPA möglicherweise große Codeblöcke neu schreiben müssen.
Sicherheit	**Binden** Während die Leute immer darauf hinweisen, dass SPAs Cross-Scripting-Angriffen ausgesetzt sind, weisen MPA's auch Sicherheitslücken auf, einschließlich Schwächen gegen Injektionen, die XSS ähnlich sind. Der Schlüssel hier ist, diese Schwächen zu kennen und Schutz einzubauen.
Anpassungsfähigkeit	**SPA** Single-Page-Anwendungen sind von Natur aus flexibler in der Gestaltung. Sie lassen sich einfacher vom Desktop auf das Handy und umgekehrt anpassen. Darüber hinaus kann ein SPA-Backend für eine mobile App wiederverwendet werden, normalerweise zusammen mit dem Oberflächendesign.
SEO	**Binden** Um SPA's SEO-freundlich zu machen, müssen Ihre Entwickler serverseitiges Rendering verwenden und die Tags von Anfang an im Auge behalten. Aber es ist nicht schwer und auch kein langwieriger Prozess. Heutzutage ist dies also kein großer Nachteil für SPA's im Vergleich zu MPA's.

Tabelle 3.1: Vorteile MPA und SPA [Tab]

3.4 Typische Anwendungsfälle einer SPA

Bekannte Beispiele, in denen mit SPA's gearbeitet wird, sind unter anderem Twitter, Google Maps oder auch Gmail. SPA's sind in vielen Fällen sinnvoll anwendbar und besonders die genannten Beispiele profitieren dabei von einer reduzierten Client-Server-Kommunikation, da diese bei Anwendungen mit vielen Nutzern schnell ausgelastet sein kann. Außerdem bieten hier die Offline-Funktionen von SPA's, insbesondere bei den den Mails, einen weiteren großen Vorteil. Diese Funktionen ermöglichen es, bereits empfangene Nachrichten noch lesen zu können, obwohl der Nutzer nicht mehr mit dem Internet verbunden ist. SPA's sind auch bei anderen Anwendungsfällen sinnvoll einsetzbar, wie beispielsweise bei kleineren Landingpages, sowie besonders auch bei Applikationen mit einem hohen Grad an Interaktion zwischen Client und Server. [SPA]

Kapitel 4

React als SPA-Framework

4.1 Einleitung

Das Framework React ist eine JavaScript-Bibliothek, welche von Facebook entwickelt wurde, um Benutzeroberflächen zu erstellen. Die von dem Softwareingenieur Jordan Walk entwickelte Bibliothek wurde im Jahr 2011 erstmals von Facebook, für den Newsfeed des Unternehmens, selber genutzt. Im Jahr 2012 wurde React dann auch für Instagram eingesetzt. Seit React 2013 zu einem Open-Source-Projekt geändert werden sollte, kam es bis zum Jahr 2017 immer wieder zu Lizenzproblemen. In diesem Jahr wurde dann die Version 16.0.0 unter der MIT-Lizenz final veröffentlicht. Alles in allem bietet diese Version des Frameworks, welche grundsätzlich als Basis für SPA's genutzt wird, viele Besonderheiten, die für eine Nutzung von React sprechen. [Ein]

4.2 Struktur

Im Folgenden werden die einzelnen Aspekte und Anforderungen beschrieben, die das Framework React mitbringt und welche dieses ausmachen.

4.2.1 User-Experience und Usability

Die User-Experience (UX) von React zeichnet sich besonders durch die großen Freiheiten aus, welche man in Bezug auf die Erstellung des Frontends erhält. Das liegt daran, dass die Logik von React nur die Darstellung der Oberfläche selber abdeckt. Bei den anderen Strukturen und Architekturen hingegen setzt React im Gegensatz zu vielen anderen Frameworks keine weiteren Eingrenzungen. Diese großen Freiheiten sind zwar einer der größten Vorteile für den Benutzer in Sachen Usability (UI), sind allerdings dadurch auch für einen schwierigen Einstieg für neue Nutzer von React verantwortlich. Um die Einsteigerfreundlichkeit für die Nutzer zu erhöhen, gibt es die 'Create React App', welche mit verschiedensten Werkzeugen eine einfachere Handhabung der Architektur und der einzelnen Komponenten ermöglicht. Schon entwickelte React-Anwendungen können schnell auf alle mögliche Gerätschaften portiert werden. Ein Beispiel dazu wären die mobilen Applikationen mit React native. Des weiteren wird dadurch die Usability verbessert, da die Anwendungen auf verschiedenen Geräten eine ähnliche Oberfläche aufweisen. [Spr20, S. 24-28]

4.2.2 Komponenten

Die Funktionsweise von Komponenten ist der Funktionsweise von JavaScript-Funktionen sehr ähnlich. Dies bedeutet, Komponenten können Eingaben, sogenannte Props empfangen und verarbeiten und geben diese dann als React-Elemente zurück. Komponenten in React lassen sich in verschiedene Arten von Komponenten einteilen. Die Funktionskomponenten sind die einfachsten Komponenten und entsprechen im Grunde einer JavaScript-Funktion. Die Funktion akzeptiert, wie oben schon beschrieben, ein Objekt mit Daten (Props) und gibt dann ein React-Element als Antwort zurück. Eine weitere Art sind Klassenkomponenten, welche für React allerdings gleich funktionieren. Durch das Rendern von Komponenten kann man benutzerdefinierte Komponenten erstellen. Sobald React eine benutzerdefinierte Komponente erkennt, werden alle JSX-Atrribute als ein Objekt übergeben, auch hier spricht man wieder von Props. React sieht viele Objekte, wie beispielweise Buttons und Screens als Komponenten an. Um diese alle sinnvoll zusammenzubringen, ist sowohl das Zusammensetzen von Komponenten, als auch das Auslagern, um große Komponenten in Kleinere aufzuteilen, möglich. Dabei spielt auch die Hierarchie der Komponenten eine wesentliche Rolle.
Ein einfaches Beispiel für Komponenten, welches durch das Rendern am Ende 'Hallo Sarah' ausgeben soll, wird im Folgenden präsentiert. [Kom]

```
function Welcome(props) {
  return <h1>Hallo {props.name}</h1>;
}

const element = <Welcome name="Sarah" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Abbildung 4.1: Komponenten rendern [Kom]

Die Funktionskomponente, die 'Hallo Sarah' ausgeben kann, kann allerdings auch als Klasse verwirklicht werden.

```
54  class Welcome extends React.Component {
55    render() {
56      return (
57        <div>
58          <h1>Hallo {props.name}</h1>
59        </div>
60      );
61    }
62  }
63
64  ReactDOM.render(
65    <Welcome name="Sarah" />,
66    document.getElementById('root')
67  );
```

Abbildung 4.2: Komponentenklasse

4.2.3 Props

Die Props, welche im vorherigen Kapitel schon angesprochen wurden, kann man auch als Eigenschaften bezeichnen. Props sind also die Eigenschaften, die den Funktionen zur Verarbeitung übergeben werden. Diese können allerdings von den Funktionen und Klassen nicht mehr verändert werden, an die sie übergeben werden. Sie sind also 'read-only'. [Kom]

4.2.4 Komponenten-Hierarchie

Die Oberfläche einer React-Anwendung kann man auch in einer Komponenten-Hierarchie zerlegen. Dabei kann man sich jede einzelne Komponente als einen Baustein des Systems vorstellen, wobei man innerhalb der Hierarchie immer von den größten zu den nächst kleineren Komponenten vorgeht und so weiter. Jede Komponente sollte dabei genau eine Aufgabe erfüllen und das Ergebnis mithilfe der Props an die nächste Komponente innerhalb der Architektur weiterleiten. [Spr20, S. 37] Genauso agieren auch die beiden Komponenten *element* und *Welcome* aus Abbildung 4.1 miteinander.

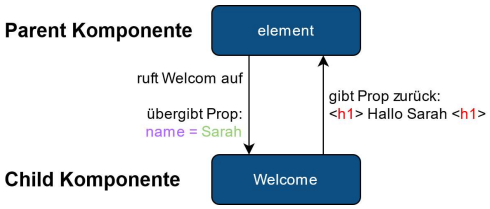

Abbildung 4.3: Komponenten-Hierarchie zu Abbildung 4.1

4.2.5 Unidirektionaler Datenfluss

Eine weitere Besonderheit von React ist der 'unidirectional data flow', im Deutschen auch unidirektionaler Datenfluss. Dieser steht in Verbindung mit der Definition der Komponenten bei React und soll den Aufbau sowie die Wechselwirkung zwischen den einzelnen Komponenten vereinfachen. Der unidirektionale Datenfluss funktioniert dementsprechend ohne komplexe Event-Händler. Stattdessen erhält eine Komponente die benötigten Daten nur über statische Eigenschaften und kann diese selber auch nicht mehr ändern. Die einzige Möglichkeit andere Komponenten trotz unidirektionalem Datenfluss noch zu beeinflussen, sind Callback-Funktionen. Diese können innerhalb der statischen Eigenschaften mitgegeben werden und mittels Callback übergeordnete Komponenten erreichen. [Ein]

4.2.6 JavaScript und XML (JSX)

Die Template-Sprache JavaScript-XML (JSX) ist an XML angelehnt und bietet eine Option für eine Syntax für React. Mittels dieser Syntax können in React JavaScript, HTML und CSS eingebunden und somit in den resultierenden Applikationen genutzt werden. Um dies zu ermöglichen, können diese Logiken durch JSX in React-Komponenten gekapselt werden. [Ein]

4.2.7 Virtual DOM (VDOM)

Der VDOM ist im Vergleich zum Standard DOM deutlich komprimierter und beschränkter, da er nur die wichtigsten Informationen verarbeitet. Dieses komprimierte, virtuelle Abbild von DOM ermöglicht eine deutlich schnellere Handhabung des Algorithmus bei einer Änderung des Programms, welche den DOM-Baum beeinflusst. Der Differenzierungsalgorithmus ist ein stark optimierter und heuristischer Algorithmus, um den VDOM möglichst schnell verändern zu können. Der VDOM, im Grunde ein JavaScript-Objekt, wird bei jeder Veränderung neu erstellt. Mithilfe eines Komponentenvergleichs wird eine Liste mit den Änderungen des DOM-Baums erzeugt. Dazu werden die zu

ändernden Knoten (weiße Kreise in Abbildung 4.4) im VDOM verändert und während der Differenzierung mit dem alten Stand des DOM-Baums abgeglichen. Beim Rendern wird diese Version überschrieben.[DOMb]

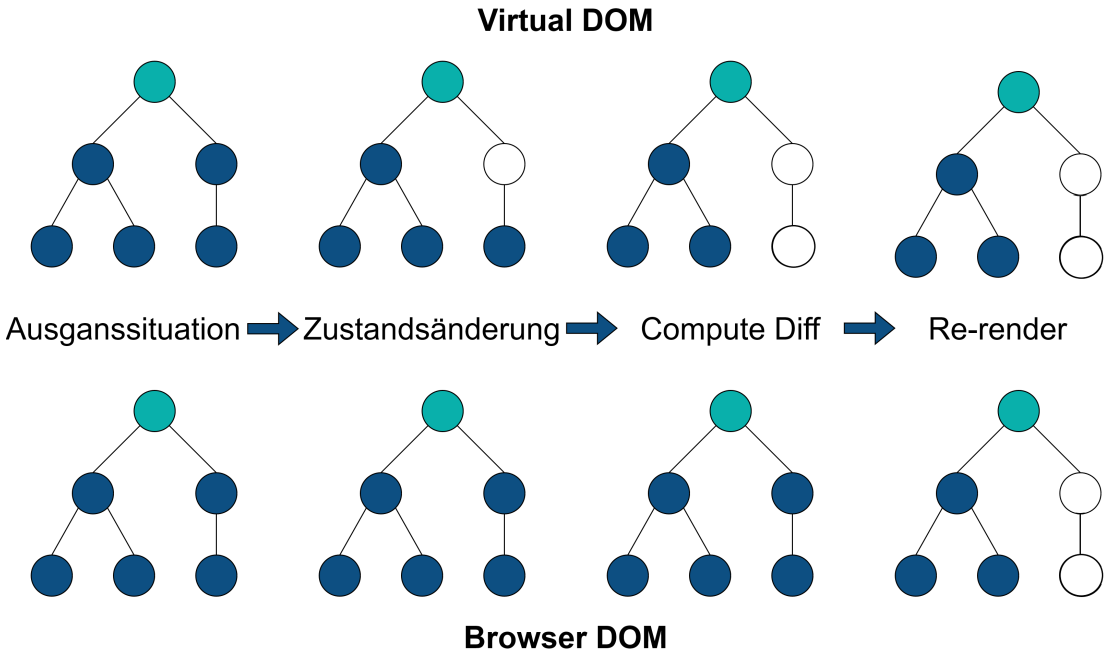

Abbildung 4.4: Virtual DOM vs. Browser DOM

4.2.8 Lebenszyklus von Komponenten

Der Lebenszyklus von Komponenten ist ein wichtiges Thema bei React. Dabei kommt es darauf an, ab wann und wie lange auf die einzelnen, erstellten Komponenten zugegriffen werden kann und wie diese Zugriff auf den DOM-Baum erhalten, um diesen manipulieren zu können. Zuallererst muss eine Komponente initialisiert werden, um auf diese zugreifen zu können. Dafür gibt es 3 wichtige Methoden:

- *componentWillMount()*
- *render()*
- *componentDidMount()*

Die *componentWillMount()* kann die letzten Veränderungen an der Komponente vornehmen, bevor diese zum DOM hinzugefügt wird.
Des weiteren gibt es die Methode *render()*, welche für jede Aktualisierung bzw. Veränderung des Zustands der Komponente aufgerufen wird und entweder den aktualisierten Stand oder null (bei keiner Änderung) zurückgibt.

Die Methode *componentDidMount()* sorgt dafür, dass die Komponente den Zugriff auf den DOM-Baum erhält.
Häufig verwendet wird so etwas beispielsweise in Kombination mit Timern oder Event-Listenern. Um eine Komponente am Ende wieder vom DOM-Baum abzukoppeln, beziehungsweise zu entfernen, wird die Methode *componentWillUnmount* verwendet. Diese kann unter anderem auch genutzt werden, um vorhandene Prozesse, wie beispielsweise einen zuvor gestarteten Timer, zu stoppen. Somit endet dann auch der Lebenszyklus einer Komponente. [Lif] Das Beispiel des Timers wird in der folgenden Abbildung 4.5 noch einmal aufgegriffen.

```
54  class Clock extends React.Component {
55    constructor(props) {
56      super(props);
57      this.state = {date: new Date()};
58    }
59
60    componentDidMount() {
61      this.timerID = setInterval(
62        () => this.tick(),
63        1000
64      );
65    }
66
67    componentWillUnmount() {
68      clearInterval(this.timerID);
69    }
70
71    tick() {    this.setState({      date: new Date()    });  }
72    render() {
73      return (
74        <div>
75          <h1>Hello, world!</h1>
76          <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
77        </div>
78      );
79    }
80  }
81
82  ReactDOM.render(
83    <Clock />,
84    document.getElementById('root')
85  );
```

Abbildung 4.5: Beispiel für einen Timer[Tim]

4.3 State - Zustandsverwaltung

4.3.1 Lokaler/globaler Zustand

Bei der Entwicklung einer Anwendung stellt sich immer wieder die Frage, wo und wie werden die Zustandsdaten oder der Zustand (State) gehalten. Dies ist auch bei der Ent-

wicklung mit React ein wichtiges Thema. Beim lokalen State handelt es sich um einen Zustand, der die Komponente direkt vorhält und diese ausschließlich intern benutzt. Bei dem globalen State hingegen steht der Zustand für mehrere Komponenten bereit, wobei auch die ganze Applikation möglich wäre. Der Zustand in React lässt sich unter anderem anhand der klassischerweise verwendeten Funktionen, beziehungsweise Hooks unterscheiden. Während beim Lokal-State die useState-Hook im Vordergrund steht, wird für den Global-State eine useReducer-Hook verwendet. Die Zustände lassen sich allerdings nicht nur an den verschieden Funktionen, sondern auch durch unterschiedliche Interaktionsarten differenzieren. Die Art der Interaktion, beziehungsweise die Wahl der richtigen Technologie, wird darin unterschieden, wie häufig gewisse Komponenten, beispielsweise Eingabefelder, aktualisiert werden müssen. Dabei sollten häufig zu aktualisierende Felder global gehalten werden, wohingegen Anwendungen, die zum Beispiel einen einmaligen Login benötigen, eher lokal angelegt werden sollten. Alle diese Unterscheidungen sind essenziell wichtig für eine optimale Performance.[Har]

4.3.2 useState-Hook

Im Grunde ist eine Hook eine spezielle Funktion, welche sich in React-Funktionen (Komponenten) einhaken kann. Mithilfe der *useState-Hook* wird der klassische Lokal-State erzeugt. Dabei wird der Zustand direkt verändert. Diese Hook ersetzt somit in gewisser Weise den *this-Zeiger*, der innerhalb von Funktionskomponenten nicht existiert. Innerhalb einer Komponente kann man also anstatt einer Zuweisung, eine *useState-Hook* aufrufen, welche dann eine Zustandsvariable erstellen kann. [Har]

4.3.3 useReducer-Hook

Eine Reducer-Funktion ist eine seiteneffektfreie und reguläre JavaScript-Funktion. Die *useReducer-Hook* wird implementiert, um die Logik zur Zustandsbearbeitung in ihr auszulagern. Das Ganze funktioniert, indem die Funktion den alten Zustand bekommt und diesen der Action übergibt, um auf Basis dieser beiden Komponenten einen neuen Zustand zu erzeugen. Der neue Zustand wird danach an die genutzte Komponente übergeben, wodurch diese neu gerendert wird. Dabei haben die Reducer-Funktion, sowie die zugehörigen Actions, keinerlei Abhängigkeiten zu React. Diese Actions können aufgrund der Unabhängigkeit somit frei definiert werden. Weitere Komponenten der Reducer-Funktion sind eine beinhaltete type-Property zur Identifikation und teilweise auch ein *Payload*, der beispielsweise eine Fehlermeldung enthalten kann, die dann an entsprechender Stelle ausgegeben werden kann. Verallgemeinert ist der *Payload* also eine Möglichkeit, der Funktion entsprechende Daten mitzugeben. Der Unterschied zu *useState* besteht in der Aktion (Action), da diese mittels der *useReducer-Hook* nur ausgelöst wird, anstatt den Zustand wie bei *useState* direkt zu verändern. [Har]

Kapitel 5

Redux als zentrale Zustandsverwaltung einer SPA

5.1 Einleitung

Redux ist eine JavaScript-Bibliothek, welche theoretisch in Kombination mit jedem JavaScript-Framework verwendet werden kann. Im Normalfall wird Redux aber gemeinsam mit dem Framework React genutzt, um unter anderem grafische Oberflächen zu erstellen. Redux ist ähnlich wie React von Facebook inspiriert. Genauer gesagt ist es an das Flux-Konzept[Flu] angelehnt, dass standardmäßig dafür sorgen soll, den State einer Applikation einfach und nachvollziehbar zu verwalten. Wichtig für die Nutzung von Redux ist vor allem, zu prüfen, ob eine Applikation wirklich komplex genug ist, um einen State-Container sinnvoll einzubauen. Falls eine Anwendung, das gilt besonders für kleinere und simpel gehaltene Applikationen, darauf nicht unbedingt angewiesen ist, wird ein weiteres Framework diese eher komplexer gestalten, anstatt einen Nutzen zu bringen.[Spr]

5.2 Struktur

Alles in allem stützt sich die Struktur von Redux auf drei Grundprinzipien. Diese Prinzipien sind der Store, die Actions und die Reducer, welche im Folgenden noch expliziter beschrieben werden. [Spr]

5.2.1 Store

Den Store, einen der drei Redux-Prinzipien, gibt es genau einmal in einer Redux-Applikation. Dieser beinhaltet alle wichtigen Zustandsinformationen der Anwendung. Mithilfe dieses Aufbaus kann sowohl das Füllen der Applikation mit den benötigten Daten als auch das Debugging vereinfacht werden, da dies durch den einen Store in nur einer Aktion erledigt werden kann. [Spr]

5.2.2 Action

Ein weiteres Prinzip sind die Actions. Diese sind wichtig für alle Veränderungen, die am Store vorgenommen werden sollen. Eine Action, mit den zu veränderten Daten wird an

den Reducer dispatched. Dieser liest den Zustand des Stores aus und erzeugt anhand der Action-Daten einen neuen Zustand. Danach wird neu gerendert.[Spr]

5.2.3 Reducer

Das letzte Prinzip sind die Reducer, welche die aufgerufenen Actions verarbeiten. Somit bestimmen die Reducer, wie die gesendeten Actions den State verändern. Diese Reducer funktionieren, indem sie eine Eingabe bekommen und daraus eine seiteneffektfreie Ausgabe erstellen. Somit bedienen sich die Reducer am Konzept der Pure Functions und können daher gut und einfach getestet werden. [Spr] Das Zusammenspiel der wichtigsten Komponenten wird in der folgenden Architektur-Abbildung von Redux noch einmal zusammengefasst.

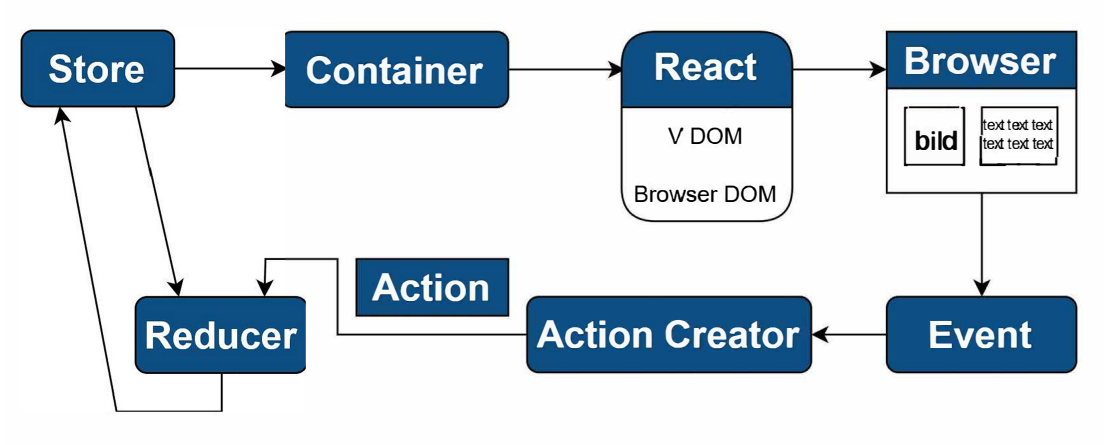

Abbildung 5.1: Typischer Ablauf von Redux und React in Bezug auf SPA's

5.2.4 Synchrone und asynchrone Kommunikation

Bei der synchronen Kommunikation soll alles gleichzeitig abgewickelt werden. Um die Daten vor dem Senden oder Empfangen zu synchronisieren, warten die einzelnen Prozesse darauf, bis die Kommunikation abgeschlossen ist und die Daten dann gemeinsam synchron übergeben werden. [Syn] Redux selber ist vollkommen synchron, allerdings kann mithilfe von asynchronen Middlewares auch eine asynchrone Kommunikation hergestellt werden. Um dies zu erreichen, muss eine Middleware auf den Redux-Store angewendet werden und dabei mit einem asynchronen *Thunk* interagieren. Durch den Zugriff des *Thunks* kann der *Dispatch*, anstatt auf die synchrone Anwendung, auf den asynchronen Rückruf (*Thunk*) reagieren.[Red]

Die asynchrone Kommunikation hingegen funktioniert komplett ohne das Blockieren von Prozessen. Bei dieser Kommunikation geschieht das Senden und Empfangen von Daten daher zeitlich versetzt, wie das beispielsweise auch beim klassischen E-Mail- oder SMS-Verkehr der Fall ist. [Asy]

5.2.5 Middleware

Die Middlewares bei Redux ermöglichen es, Nebeneffekte auszuführen, ohne dabei den State und dessen Aktualisierungsprozess zu beeinflussen. Im Grunde funktioniert das Zusammenspiel von Actions und Reducern, um den State zu aktualisieren, ohne Probleme. Allerdings kann es immer mal zu unerwünschten Nebenwirkungen kommen, falls man beispielsweise alle gesendeten Actions protokollieren will oder auch mit einer weiteren, externen API kommunizieren möchte. Um zu vermeiden, dass solche Zusatzfunktionen den geregelten Ablauf stark beeinflussen, beziehungsweise unterbrechen, gibt es das Middleware-Pattern in Redux. Die Middleware lässt sich allgemein zwischen dem Senden der Action und der Zustandserstellung vom Reducer einordnen. An dieser Stelle empfängt sie die Action und kann somit entsprechend reagieren. Dabei kann man eine Middleware unter anderem selber schreiben und einbauen, was meist gemacht wird, wenn man Actions protokollieren will. Des Weiteren gibt es auch viele vorhandene Middlewares, wie z. B. *Thunk* oder *Saga*, die man nutzen kann, um den eigentlichen synchronen Ablauf asynchron zu gestalten. [Gui]

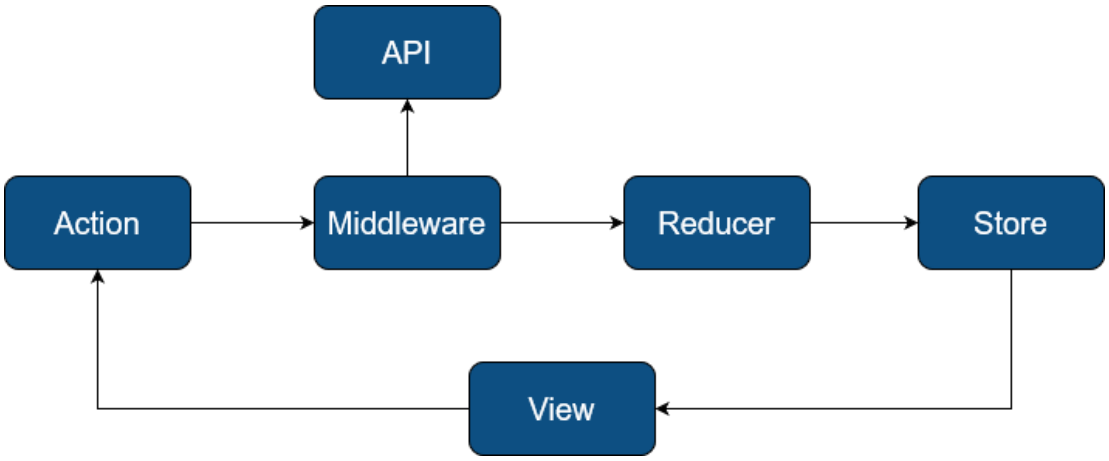

Abbildung 5.2: Struktur Redux und React mit Middleware

Kapitel 6

SPA-Beispiel mit React und Redux

6.1 Entwicklungsumgebung

Die genutzte Entwicklungsumgebung, in der die Applikation getestet wurde, war Visual Studio Code. Die Website wurde dann über Chrome beziehungsweise Microsoft Edge ausgeführt. Die genaueren Schritte werden im Folgenden aufgezählt:

1. IDE: Visual Studio Code der Version 1.63
2. Quellcode-Verwaltung - Git
3. Installer / Paketverwalter - NPM
4. Web-Entwicklungsserver - NodeJS
5. REST-API - JSON-Server
6. Debugging - Web-Entwicklungstools von Google Browser Chrome
7. Degugging - Redux devTools

6.2 Beispiel Anwendung

6.2.1 Beschreibung

Die verwendete Beispielapplikation basiert auf der von mir benutzten Grundliteratur von Marc Garreau und Will Faurot. Diese haben ein Buch zu React und Redux mit dem Titel 'Redux in Action' [MG18,] verfasst. Dieses befasst sich sowohl mit allen Grundlagen und Anwendungsformen der Frameworks als auch mit einer Verbindung zu der Redux-Bibliothek. Im Zuge dessen, um auch das gesamte System von React und Redux kennen zu lernen und zu analysieren, gibt es eine Beispielapplikation, welche an diese Frameworks angelehnt ist. Diese Applikation stellt eine klassischen To-Do-Liste dar, welche mithilfe von React und Redux eine SPA kreiert. Diese To-Do-Liste lässt sich in 3 Stufen unterteilen und aktualisieren. Außerdem lassen sich je nach Optimierungsgrad der Entwicklung die einzelnen Elemente der Liste mithilfe von Buttons sowohl hinzufügen und löschen, als auch bereits vorhandene To-Do's wieder bearbeiten. Um am Ende auch einen zeitlichen Nutzen der Aktivitäten festzustellen, gibt es einen Timer, der die Zeit

einer aktuell praktizierten Tätigkeit misst und anzeigt. Der aktuelle Zustand der Anwendung, also alle Aufgaben und die zugehörigen Bearbeitungszeiten und deren Zustände in den Tabellen, werden in einer JSON-Datenbank gespeichert. Der Zugriff auf die JSON-Datenbank erfolgt über die REST-API.

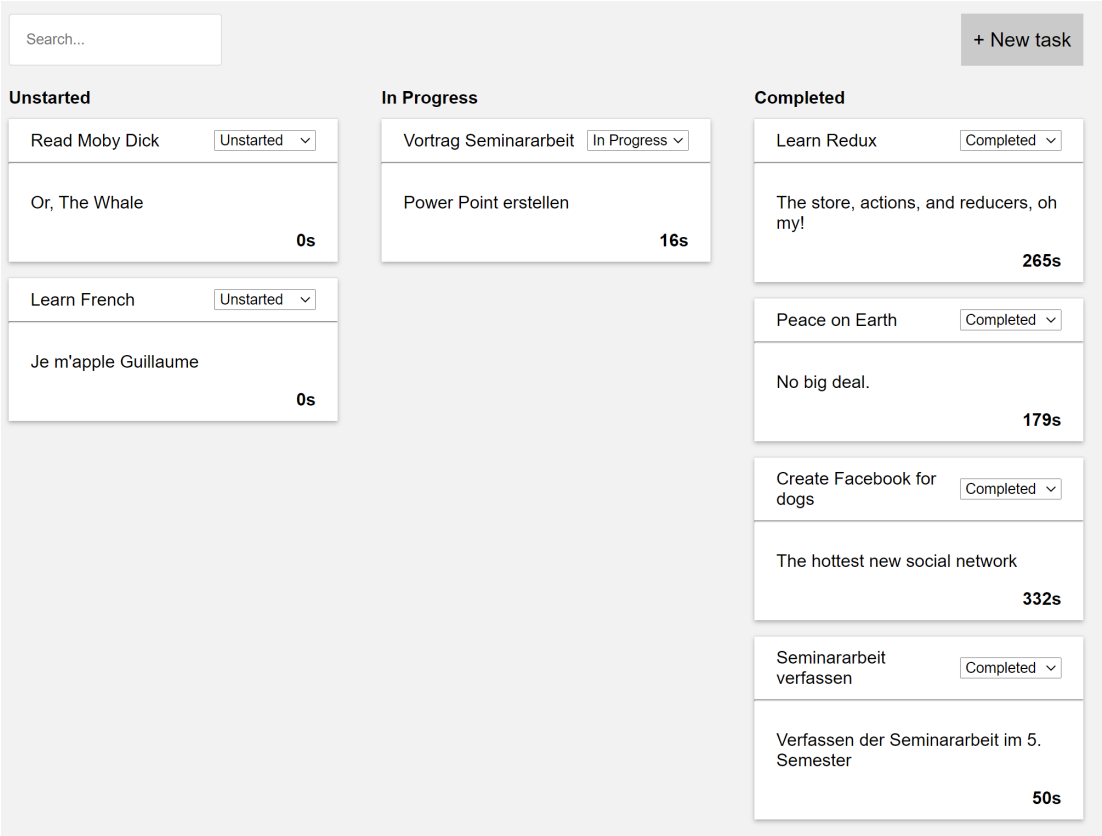

Abbildung 6.1: Beispielapplikation

6.2.2 Installation

Die Installation dieser Beispielapplikation wird mithilfe eines GitHub-Repository ermöglicht, in dem alle gegebenen Kapitel als Download erhältlich sind. [Gar] Die genaueren Schritte werden im Folgenden aufgezählt: [Ins]

1. Der Quellcode wurde von GitHub mit Hilfe von Git geholt: git clone [Gar]
2. Die benötigten Pakete wurden mit dem Paketverwalter / Installer zusammen gestellt: npm install / npm audit fix
3. Der JSON-Server für die REST-API gestartet: json-server –watch db.json –port 3001

4. Die Applikation gestartet mit ”npm start”

5. Die Applikation wurde im Google Browser Chrome ausgeführt unter der URL: http://localhost:3000

Bei der Analyse in den folgenden Kapiteln wurde der Code-Stand des neunten Kapitels verwendet.

6.3 Implementierung

Die Implementierung des Programms zum Stand des neunten Kapitels wird in diesem Abschnitt im Detail analysiert. Der Schwerpunkt wird auf den Umgang der Beispielapplikation in Bezug auf die wichtigsten Stützen der Struktur von React und Redux gelegt.

6.3.1 Store

Mithilfe von Redux wird ein Zustands-Container erstellt, der Redux Store. Dieser kann initialisiert werden, nachdem die Actions einer Applikation definiert und die Reducer erstellt wurden. Sobald Redux dann mittels *'npm install -P Redux'* dem Projekt hinzugefügt und die zulässigen Methoden in der index-Datei importiert wurden, kann der Store eingegliedert werden. Optional kann der Store, wie auch in der Beispielapplikation, eine Middleware enthalten.

```
6   import { createStore, applyMiddleware } from 'redux';
21  const sagaMiddleware = createSagaMiddleware();
22
23  const store = createStore(
24    rootReducer,
25    composeWithDevTools(applyMiddleware(thunk, sagaMiddleware))
26  );
```

Abbildung 6.2: Anlegen des Store-Containers mit der Verbindung zur Middleware

Im Grunde ist Store nur ein Objekt mit ein paar Methoden und verwaltet den globalen Applikationszustand (State). Um dies zu gewährleisten, nutzt der Store die Methoden *getState()* und *dispatch()*. Dabei sorgt die Methode *getState()* für das Lesen der Daten im State und mithilfe von *dispatch()* werden die Daten im State dann verändert. In der Beispielapplikation wird der State unter anderem über die Dispatch-Funktion verändert, wenn Aufgaben (Tasks) der To-Do-Liste hinzugefügt oder bereits vorhandene Aufgaben bearbeitet werden.

```
24  export function createTask({ title, description, status = 'Unstarted' }) {
25    return dispatch => {
26      dispatch(createTaskRequested());
27      return api.createTask({ title, description, status }).then(resp => {
28        dispatch(createTaskSucceeded(resp.data));
29      });
30    };
31  }
```

Abbildung 6.3: Dispatch bei der Methode zur Erstellung neuer Aufgaben

Durch React kann die Komponente *Provider* in die Applikation eingebunden werden, diese sorgt dafür, der Applikation den Store zur Verfügung zu stellen. Um dies zu gewährleisten, wird die *Provider-Komponente* auf der obersten Komponentenebene integriert.

```
30  ReactDOM.render(
31    <Provider store={store}>
32      <App />
33    </Provider>,
34    document.getElementById('root')
35  );
```

Abbildung 6.4: Einbinden der Provider-Komponente auf der höchsten Ebene

Damit die Komponenten der Anwendung auf den Store zugreifen können, muss eine Verbindung zwischen den Eigenschaften der Komponenten und dem State geschaffen werden. Dies geschieht innerhalb der Beispielapplikation mittels der Methode *mapStateToProps()*. Um die Komponente App mit der Funktion zu verbinden, wird diese mithilfe von *export default connect(mapStateToProps(App))* an diese Funktion gebunden. Nachdem der Zustand des Stores sowohl bei der Initialisierung als auch bei jeder Veränderung mithilfe von *render()* aktualisiert wird, werden die Daten aus dem aktuellen Store als Props an die Komponente übergeben. Dies geschieht in Zeile 47 von Abbildung 6.5, indem eine Verbindung zwischen den Werten aus dem Store und den Props *tasks*, *isLoading* und *error* zugeordnet werden. In Fall von *tasks* werden die Daten aus dem Store sogar anhand eines Selektors, *getGroupedAndFilteredTask*, spezifizierter ausgewählt. Die Daten aus dem Store können an dieser Stelle zwar sowohl ohne als auch mit Selektoren übergeben werden, allerdings sind Selektoren praktischer. Dies ist gegeben, weil bei einer Strukturänderung des Stores nur die Selector-Funktionen verändert werden, anstatt dass der komplette Code angepasst werden muss.

```
43  function mapStateToProps(state) {
44    //Erstellen der beiden Props und binden an den Store
45    const { isLoading, error } = state.tasks;
46    //
47    return { tasks: getGroupedAndFilteredTasks(state), isLoading, error };
48  }
49  //Verbindet Funktion zur Komponente App
50  export default connect(mapStateToProps)(App);
```

Abbildung 6.5: Methode mapStateToProps()

6.3.2 Actions und Action Creators

Actions bieten die einzige Möglichkeit den State verändern zu können. Dies geschieht, wie in Abbildung 6.4 bereits gezeigt, mithilfe der Methode *dispatch()*. Eine Action ist grundsätzlich nur ein simples Objekt, welches beschreibt, was sich ändern soll. Der Name wird dann innerhalb der Action als Type angegeben. Um eine Action individuell auszubauen, gibt es die Action Creator. Das sind Funktionen, welche ein Action-Objekt zurückgeben. Im Beispiel in der Abbildung 6.4 wird mit *dispatch()* zuerst der Action *createTaskRequested()* aufgerufen und danach mit *api.createTask()* eine neue Task anlegt. Die Action *CREATE_TASK_SUCCEEDED* quittiert, dass eine neue Aufgabe der To-Do-Liste erfolgreich hinzugefügt wurde.

```
15  export function createTaskSucceeded(task) {
16    return {
17      type: 'CREATE_TASK_SUCCEEDED',
18      payload: {
19        task,
20      },
21    };
22  }
```

Abbildung 6.6: Action Creator createTaskSuceeded

Actions können optional auch einen *Payload* enthalten. Dies ist ein Objekt, welches Daten zur Verfügung stellt, falls diese für die Action benötigt werden. Im Fall von Abbildung 6.7 gibt es einen *Payload*, dieser enthält die schon zuvor mitgegebenen Daten (siehe Abbildung 6.4) zu der neu erstellten Aufgabe, welche die Informationen Titel, Beschreibung und den aktuellen Status enthalten. In der Applikation wird das Ganze über einen Button 'newTask+' angelegt, wobei der *dispatch()* und die damit verbundene Action erst beim Betätigen des Save-Buttons in Aktion treten.

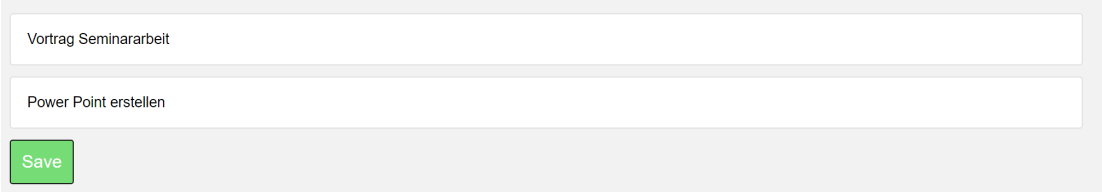

Abbildung 6.7: Task erstellen in der Anwendung

6.3.3 Reducer

Nachdem über *dispatch()* eine Action an den Store geschickt wurde, kommen die Reducer dazu. Diese verarbeiten die Action und entscheiden, wie sich der State ändern soll. Die Reducer sorgen dafür, dass bereits vorhandene State Objekte nicht direkt verändert werden, sondern mithilfe der Action ein neuer State erstellt wird, bevor dieser zurückgegeben wird. Der State muss in der index.js von den Reducern erst einmal mittels *initialState* definiert werden, damit die unterschiedlichen Werte des States vom Reducer im weiteren Verlauf auf einer Kopie korrekt aktualisiert werden können. Für die Aufgaben der To-Do-Liste kommt es nicht nur auf die Aufgaben selber an, sondern auch ob es zu einen Fehler (*error*) kommt, beziehungsweise ob die Applikation gerade im Nachlade-Prozess ist oder einen neuen Term sucht.

```
4  const initialState = {
5    tasks: [],
6    isLoading: false,
7    error: null,
8    searchTerm: '',
9  };
```

Abbildung 6.8: Erstellen des initialState in der index.js von Reducer

Dies geschieht meistens, genauso wie in der Beispielanwendung auch, über eine Switch-Anweisung. Switch differenziert dabei nach dem jeweiligen Action-Type, wobei die Actions dabei in die jeweiligen case-Unterscheidungen aufgeteilt sind. Mithilfe des JavaScript Spread Operators (drei Punkte vor dem Begriff 'state') wird eine Kopie von diesem erstellt und zusätzlich wird in der Applikation noch der jeweilige Loading-Status sowie optional ein *Payload* der State-Kopie mitgegeben.

```
export default function tasks(state = initialState, action) {
  switch (action.type) {
    case 'FETCH_TASKS_STARTED': {
      return {
        ...state,
        isLoading: true,
      };
    }
    case 'FETCH_TASKS_SUCCEEDED': {
      return {
        ...state,
        tasks: action.payload.tasks,
        isLoading: false,
      };
    }
    case 'FETCH_TASKS_FAILED': {
      return {
        ...state,
        isLoading: false,
        error: action.payload.error,
      };
    }
    case 'CREATE_TASK_SUCCEEDED': {
      return {
        ...state,
        tasks: state.tasks.concat(action.payload.task),
      };
    }
```

Abbildung 6.9: Ausschnitt einer Switch-Unterscheidung beim Holen der Tasks aus der JSON-Datenbank

6.3.4 Middleware Thunk

Im Normalfall verlaufen alle Abhandlungen zur Verarbeitung von Daten mithilfe von Reducern und Action Creatorn synchron. Das Konzept der Middlewares, welche zwischen den Actions und den Reducern als Mittelstück agieren, ermöglichen dann allerdings auch die Asynchronität. Die Middleware Thunk wird von Redux zur Verfügung gestellt und muss direkt bei der Initialisierung des Stores mitgegeben werden. Dafür muss zusätzlich noch die Bibliothek *'redux-thunk'* importiert werden.

Im Grunde ist *Thunk* nichts anderes als eine kleine Funktion. Diese wird unter an-

derem auch in der Beispielapplikation bei der Funktion *createTask()* verwendet, siehe Abbildung 6.4. Die Besonderheit, die *Thunk* in diesem Fall ausmacht, besteht darin, dass durch den Reducer mit *dispatch()* nicht nur eine Action zurückgegeben wird. In dem Beispiel erkennt man dies auch in den Zeilen 27 und 28 in Abbildung 6.4, in denen zusätzlich zu den mitgegeben Werten für die Task auch noch eine weitere Methode mit *dispatch()* verarbeitet wird. Diese wird durch *then* eingeleitet und das ist auch die Stelle, an der die Middleware *Thunk* eingreift.

6.3.5 Middleware Saga

Eine weitere Redux Middleware, welche in der Beispielapplikation verwendet wird, ist die *Saga* Middleware. Diese muss zunächst standardmäßig importiert werden. Zusätzlich muss noch einen weiterer Import getätigt werden, um Redux mitzuteilen, welche *Saga/s* genutzt werden. In der index.js der Hauptdatei aus der Beispielanwendung wird auf die *rootSaga* zurückgegriffen.

```
import createSagaMiddleware from 'redux-saga';
import rootSaga from './sagas';
```

Abbildung 6.10: Importieren der *rootSaga*

Damit *createSagaMiddleware()* als factory-Funktion zum Erstellen einer *Saga* Middleware genutzt werden kann, muss diese wie folgt definiert werden.

```
21   const sagaMiddleware = createSagaMiddleware();
```

Abbildung 6.11: Definition von *createSagaMiddleware()*

Die Saga Middleware dient dazu, Objekte von Sagas zu empfangen. Sagas werden als Generatorfunktionen implementiert, was an der Deklaration *'function*'* erkennbar ist. Der Vorteil dieser Generatorfunktionen liegt darin, dass diese auch während einem Funktionsaufruf verlassen und später wieder aufgerufen werden können. Dies geschieht durch *'yield'*. Zurückgegeben wird ein Generator-Objekt, welches im Grunde einen Initiator darstellt. In der Beispielapplikation werden alle Generatorfunktionen in sagas.js verwaltet. Eine Funktion, die dort unter anderem verwaltet wird, ist die *handleProgressTimer-Funktion*. Diese sorgt dafür, die Zeit solange zu aktualisieren, bis der Timer gestoppt wird.
Dafür wird *yield* mit einem *call-Ausdruck* genutzt, um in einem bestimmten Abstand den Status abzufragen und dann mithilfe eines weiteren *yield* in Verbindung mit *put()* die Zeitanzeige zu aktualisieren. Anhand des sagaspezifischen *yield* kann eine ständige Kommunikation mit dem Timer der Anwendung ermöglicht werden.

```
26  export function* handleProgressTimer({ type, payload }) {
27    if (type === 'TIMER_STARTED') {
28      while (true) {
29        yield call(delay, 1000);
30        yield put({
31          type: 'TIMER_INCREMENT',
32          payload: { taskId: payload.taskId },
33        });
34      }
35    }
36  }
```

Abbildung 6.12: Saga handleProgressTimer in sagas.js

6.4 Debugging mit Redux DevTools

Die Redux DevTools sind eine Debugging-Plattform und ermöglichen es, Redux-Applikationen in Echtzeit zu korrigieren und zu bearbeiten (Life-Editing). Die Entwicklertools können dem Nutzer auf unterschiedliche Weisen angezeigt werden. Dabei kann man diese unter anderem als Komponente der Applikation ansehen, aber auch in einem getrennten Pop-Up Fenster. Eine weitere Option, die auch beim Debugging der Anwendung genutzt wurde, besteht darin, diese innerhalb der Developer Tools des Browsers zu nutzen. Dafür installiert man zuerst die Redux DevTools über den Chrome Browser und fügt diese im Anschluss dem Store der zu debuggenden Applikation hinzu, um dieser die Tools zugänglich zu machen.

```
7  import { composeWithDevTools } from 'redux-devtools-extension';
```

Abbildung 6.13: Importieren der devTools

Mithilfe dieser DevTools können sowohl Zustände überprüft als auch Actions abgebrochen oder sogar zur Laufzeit verändert werden. Nach der Installation der Redux DevTools für Chrome wird ein eigenes Redux-Panel den DevTools hinzugefügt und kann über diese aufgerufen werden.

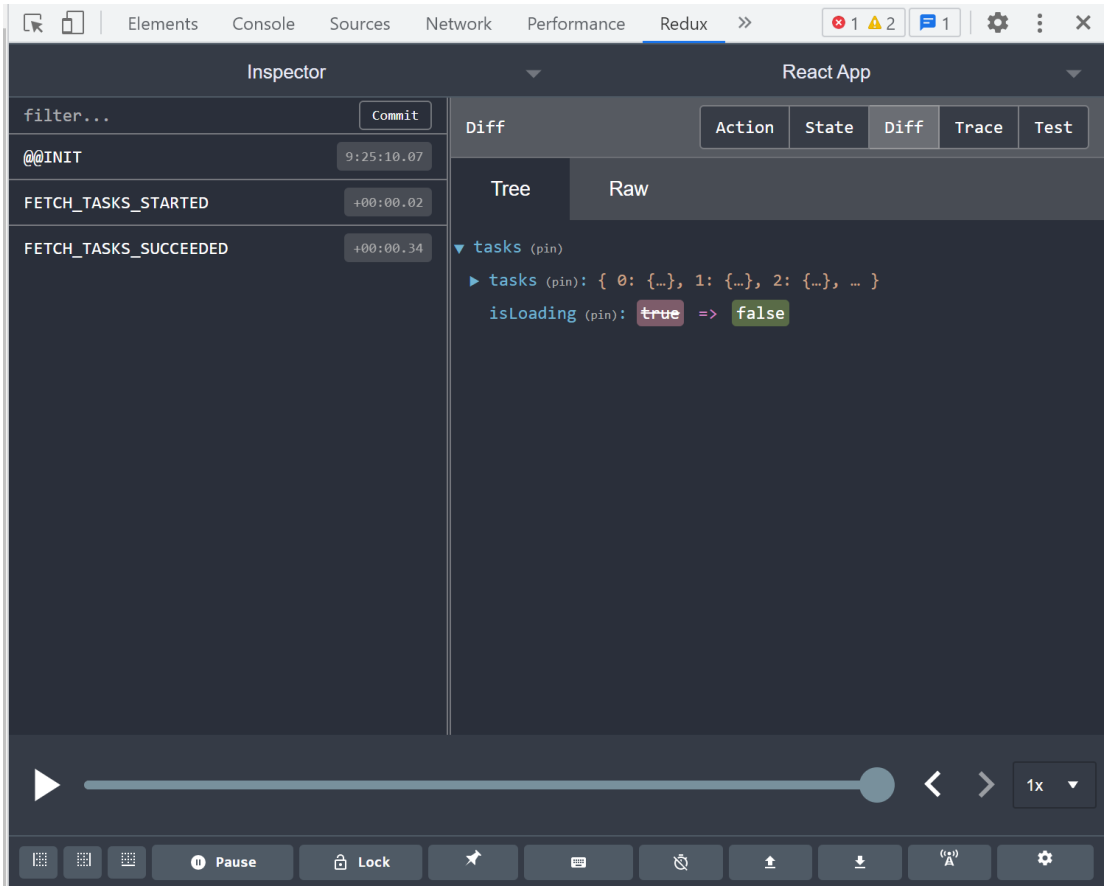

Abbildung 6.14: Redux-Panel der Redux DevTools

Kapitel 7

Ergebnisse

7.1 Fazit

Nach der ausführlichen Recherche und der Analyse der im vorherigen Kapitel behandelten Beispielapplikation lässt sich schlussfolgern, dass die Anwendung der Frameworks React und Redux in Kombination viele Vorteile bietet, um eine SPA zu entwickeln, welche auch auf unterschiedlichen Geräten eine sinnvolle Anwendung findet. Dies ist einer der essenziellen Aspekte, den die Frameworks mitbringen, um eine Applikation, wie beispielsweise für Julib eXtended, zu entwickeln, welche auch auf unterschiedlichen Geräten im Home-Office nutzbar ist. Einen weiteren Pluspunkt bringt dabei die geringe Bandbreitennutzung aufgrund der angewendeten Dynamik mit sich, da dadurch auch die Nutzung in Wohngegenden mit schlechterer Internetleitung wahrscheinlicher gewährleistet ist. Aber nicht nur das schnelle und dynamische Nachladen der Applikation ist bei der Analyse positiv aufgefallen. Denn auch die meiner Meinung nach sehr guten Debugging-Methoden der Redux DevTools vereinfachen dem Entwickler die Handhabung mit Fehlern, die bei der Entwicklung auftreten können. Besonders stechen dabei das eigene, gut strukturierte Redux-Panel und das Echtzeit-Debugging heraus.
Des Weiteren sprechen besonders auch die Performance Aspekte für eine Nutzung. Diese werden durch den unidirektionalen Datenfluss zwischen View, Actions und dem State erzeugt, welcher nur durch die Möglichkeit asynchroner Middleware beeinflusst werden kann. Zusätzlich verstärkt wird die Performance aber auch durch die generelle Komponenten-Hierarchie, welche eine feste Struktur vorgibt. Das dadurch kreierte State-Management ermöglicht vor allem die Entwicklung komplexer Benutzeroberflächen.
Alles in allem lässt sich sagen, dass die Kombination beider Frameworks allen Anforderungen entspricht, damit diese in Form einer Applikation von Julib extended als SPA im Rahmen der Bachelorarbeit entwickelt werden kann, um diese von meinem Institut der ZB allen Nutzern zur Verfügung zu stellen.

7.2 Ausblick

Innerhalb der Beispielanwendung konnte ich die Grundkonzepte der Frameworks React und Redux gut kennenlernen und vertiefen. Allerdings gibt es auch noch Punkte mit denen sich in dieser Anwendung nicht befasst wurde, welche aber weitere Vorteile bringen können, um die Usability einer SPA weiter zu verbessern. Auch in Bezug auf eine mögliche eigene Applikation in der Bachelorarbeit wäre ein Webspeicher eine sinnvolle

Erweiterung. Dieser Speicher kann den letzten Zustand einer Applikation speichern und somit auch eine eingeschränkte Offline-Nutzung einer Applikation gewähren, indem die bereits vorhandene Daten aus dem Webspeicher abgebildet werden. Eine weitere interessante Option, die mit der Framework Kombination als Grundlage möglich wäre, wäre die Erweiterung von einer SPA in eine Progressive-Web-App (PWA). Eine PWA bietet viele Möglichkeiten, die sonst nur nativen Apps vorbehalten sind. Unter anderem ist zum Beispiel die Installation einer Anwendung möglich, anstatt diese nur im Web aufrufen zu können. Dennoch besitzt sie auch Vorteile gegenüber normalen Apps, da man diese unter anderem viel einfacher veröffentlichen kann, indem man sie nicht in einem App-Store mit langwierigen Veröffentlichungsprozessen publizieren muss. Außerdem führt diese Unabhängigkeit der Stores auch zu einer Reduktion der Kosten einer Veröffentlichung.

Literaturverzeichnis

[AJA] *Vom einfachen AJAX-Request zum komplexen Objektaustausch mit JSON mittels jQuery.* https://xuad.net/artikel/vom-einfachen-ajax-request-zum-komplexen-objektaustausch-mit-json-mittels-jquery. – Stand: 16.05.2013

[Asy] *Asynchrone Kommunikation.* Wikipedia. https://de.wikipedia.org/wiki/Asynchrone_Kommunikation. – Stand: 28.04.2021

[Aug] Augsten, Dipl.-Ing. (FH) Stefan Luber / S.: *Definition „JS (Skriptsprache)“Was ist JavaScript?* https://www.dev-insider.de/was-ist-javascript-a-586580/. – Stand: 24.03.2017

[DOMa] *Document Object Model (DOM): Definition, Aufbau und Beispiel.* https://www.ionos.de/digitalguide/websites/web-entwicklung/was-ist-das-document-object-model-dom/. – Stand: 20.03.2020

[DOMb] *Was ist eigentlich ein (virtuelles) DOM?* https://t3n.de/news/eigentlich-virtuelles-dom-858160/. – Stand: 22.09.2017

[Ein] *React.* Wikipedia. https://de.wikipedia.org/wiki/React. – Stand: 06.12.2021

[Flu] *ReactJS - Flux Concept.* https://www.tutorialspoint.com/reactjs/reactjs_flux_concept.htm. – Stand: 03.01.2022

[Fra] *Framework.* https://techterms.com/definition/framework. – Stand: 07.03.2013

[Fun] *Funktionale Programmierung für JavaScript-Entwickler - Map.* https://ichi.pro/de/funktionale-programmierung-fur-javascript-entwickler-map-59542361388454. – Stand: 31.12.2021

[Gar] Garreau, Marc: *wfro/parsnip.* https://github.com/wfro/parsnip. – Stand: 23.12.2017

[Gui] Guida, Mike: *An Overview of Redux Middleware for React Applications.* https://www.metaltoad.com/blog/overview-redux-middleware-react-applications. – Stand: 02.08.2019

[Har] Hartmann, Nils: *The Art of State: Zustandsmanagement in React-Anwendungen, Teil 1.* https://www.heise.de/ratgeber/The-Art-of-State-Zustandsmanagement-in-React-Anwendungen-4934595.html. – Stand: 30.10.2020

[Hey] Heydenreich, Hendrike: *Wie Ajax die Praxis verändert.* Computerwoche. https://www.computerwoche.de/a/wie-ajax-die-praxis-veraendert,1215633. – Stand: 07.08.2006

[HTM] *Hypertext Markup Language (HTML).* https://whatis.techtarget.com/de/definition/Hypertext-Markup-Language-HTML. – Stand: 30.04.2021

[Ins] *Appendix. Installation.* https://livebook.manning.com/book/redux-in-action/appendix/21. – Stand: 13.12.2021

[JSO] *Javascript JSON.* https://www.mediaevent.de/javascript/json.html. – Stand: 31.10.2021

[Jul] *JuLib eXtended.* https://julib.fz-juelich.de/vufind/. – Stand: 13.12.2021

[Kom] *Komponenten und Props.* https://de.reactjs.org/docs/components-and-props.html. – Stand: 13.12.2021

[Lif] *ReactLebenszyklus der Reaktionskomponente.* https://learntutorials.net/de/reactjs/topic/2750/lebenszyklus-der-reaktionskomponente. – Stand: 28.12.2021

[MG18] Marc Garreau, Will F.: *Redux in Action.* Manning Publications Co., 2018

[MPA] *Was ist eigentlich eine Single-Page-Webanwendung?* https://t3n.de/news/single-page-webanwendung-1023623/. – Stand: 25.04.2018

[Qua] Quanit, Mohammad: *Single Page Application vs. Multi-page Application.* https://dev.to/mquanit/single-page-application-vs-multi-page-application-p2m. – Stand: 20.12.2019

[Red] *reduxAsynchroner Datenfluss.* https://learntutorials.net/de/redux/topic/3474/asynchroner-datenfluss. – Stand: 13.12.2021

[SPA] *Was ist eigentlich eine Single-Page-Webanwendung?* https://t3n.de/news/single-page-webanwendung-1023623/. – Stand: 25.04.2018

[Spi19] Spichale, Kai: *API-Design : Praxishandbuch für Java- und Webservice-Entwickler.* Heidelberg : dpunkt.verlag, 2019

[Spr] Springer, Sebastian: *Redux im Überblick.* https://entwickler.de/react/redux-im-uberblick. – Stand: Anfang 2018

[Spr20] Springer, Sebastian: *React Das umfassende Handbuch*. Rheinwerk Computing, 2020

[Syn] *Synchrone Kommunikation*. Wikipedia. `https://de.wikipedia.org/wiki/Synchrone_Kommunikation`. – Stand: 27.10.2018

[Tab] *Kampf um die Web-Apps*. `https://www.affde.com/de/spa-vs-mpa.html`. – Stand: 05.10.2021

[Tim] *State and Lifecycle*. `https://reactjs.org/docs/state-and-lifecycle.html`. – Stand: 30.12.2021

[Ver] *Using HTTP Methods for RESTful Services*. `https://www.restapitutorial.com/lessons/httpmethods.html`. – Stand: 13.12.2021

[Wik] *Hypertext Markup Language*. Wikipedia. `https://de.wikipedia.org/wiki/Hypertext_Markup_Language`. – Stand: 18.11.2021