React

Das umfassende Handbuch

Kapitel 2
Die ersten Schritte im Entwicklungsprozess

Auch der weiteste Weg beginnt mit einem ersten Schritt.
– Konfuzius

In diesem Kapitel lernen Sie den Entwicklungsprozess kennen: von der Initialisierung der Applikation über die Konfiguration der Entwicklungsumgebung und das Debuggen der Applikation bis hin zum Bauen der Applikation.

Der erste Schritt bei der Entwicklung einer Single-Page-Applikation ist meist mit großem Aufwand verbunden: Sie müssen Abhängigkeiten herunterladen und installieren, Strukturen schaffen, also Dateien und Verzeichnisse anlegen, und die Applikation entweder direkt vom Dateisystem starten oder über einen Webserver ausliefern. In diesem Kapitel werfen wir einen Blick auf den Lebenszyklus einer React-Applikation. Außerdem erfahren Sie, welche verschiedenen Arten es gibt, mit der Entwicklung einer solchen Applikation zu beginnen.

In der Regel greifen Sie für den Start der Entwicklung jedoch auf ein etabliertes und sehr weit verbreitetes Werkzeug mit dem Namen *Create React App* zurück. Dieses Kommandozeilenwerkzeug nimmt Ihnen die wichtigsten Arbeitsschritte ab und erzeugt Ihnen eine neue React-Applikation, mit der Sie direkt in die Entwicklung starten können.

2.1 Schnellstart

Sie wollen sofort mit der Entwicklung Ihrer Applikation beginnen und sich erst später mit den Hintergründen und den verschiedenen Alternativen beschäftigen? Dann erfahren Sie in den folgenden Abschnitten, wie Sie dies erreichen. Für Erklärungen und Details lesen Sie einfach das Kapitel bis zum Ende weiter.

2.1.1 Die Initialisierung

Für die Initialisierung einer neuen React-Applikation wechseln Sie auf die Kommandozeile Ihres Systems und geben den folgenden Befehl ein:

```
npm init react-app library
```

Listing 2.1 Initialisierung einer Applikation mit NPM

Voraussetzung für den Erfolg dieses Kommandos ist eine lokale Installation von Node.js inklusive NPM. Das `npm init`-Kommando erzeugt ein neues Verzeichnis mit dem Namen *library*, in dem sich Ihre Applikation befindet. Alle Abhängigkeiten und Strukturen sind so weit vorbereitet, dass Sie mit den Kommandos aus Listing 2.2 Ihre Applikation starten und mit der Entwicklung beginnen können.

```
cd library
npm start
```

Listing 2.2 Starten der Applikation

Weitere Informationen zu Node.js und NPM, dem *Node Package Manager*, erhalten Sie im Laufe dieses Kapitels. An dieser Stelle müssen Sie lediglich wissen, dass NPM ein Bestandteil der Node.js-Plattform ist, die Sie über *http://nodejs.org* beziehen können.

TypeScript-Unterstützung

Ich empfehle Ihnen, Ihre React-Applikation, egal wie klein sie auch sein mag, immer mit *TypeScript* zu initialisieren. Für diesen Zweck bietet *Create React App* Applikations-Templates an. Das Kommando

```
npm init react-app library --template typescript
```

Listing 2.3 Initialisierung mit TypeScript

integriert TypeScript und alle erforderlichen Werkzeuge bei der Initialisierung in die Applikation. Anschließend können Sie in das Verzeichnis wechseln und mit der Entwicklung beginnen.

2.2 Playgrounds für React

Um schnell etwas mit React auszuprobieren oder ein erstes Gefühl für die Bibliothek zu bekommen, müssen Sie nicht unbedingt etwas auf Ihrem System installieren. Es existieren zahlreiche Plattformen, die Ihnen eine Basisversion von React im Browser zur Verfügung stellen. Hier haben Sie die Möglichkeit, kleine Applikationen zu erzeugen und sie direkt im selben Fenster zu testen. Dieser Ansatz eignet sich nur für einen sehr geringen Umfang und für kleinere Experimente.

Achtung!

Sobald Sie an einer größeren Applikation arbeiten, sollten Sie den Quellcode lokal auf Ihrem System schreiben und nicht mehr auf einen solchen Playground setzen.

Im Folgenden möchte ich Ihnen mit CodePen eine dieser Plattformen vorstellen.

2.2.1 CodePen – ein Playground für die Webentwicklung

Die Plattform *CodePen* können Sie sowohl nach einer kostenlosen Anmeldung als auch anonym ohne Anmeldung nutzen. Die Adresse von CodePen lautet *https://codepen.io*. Wie auch bei anderen vergleichbaren Plattformen haben Sie drei Editoren, je einen für HTML, CSS und JavaScript. Außerdem zeigt Ihnen die Plattform das Ergebnis der Ausführung Ihres Codes in einem weiteren Abschnitt des Fensters an. Die Größe der einzelnen Sektionen können Sie per Drag-and-drop variieren, ebenso lässt sich das Layout der gesamten Plattform anpassen.

Angemeldete Benutzer*innen können ihre Experimente speichern und haben durch ein Dashboard eine Übersicht über die gespeicherten Projekte. Die Anmeldung kann entweder über ein Konto bei CodePen direkt oder aber über einen Login per Twitter, GitHub oder Facebook erfolgen.

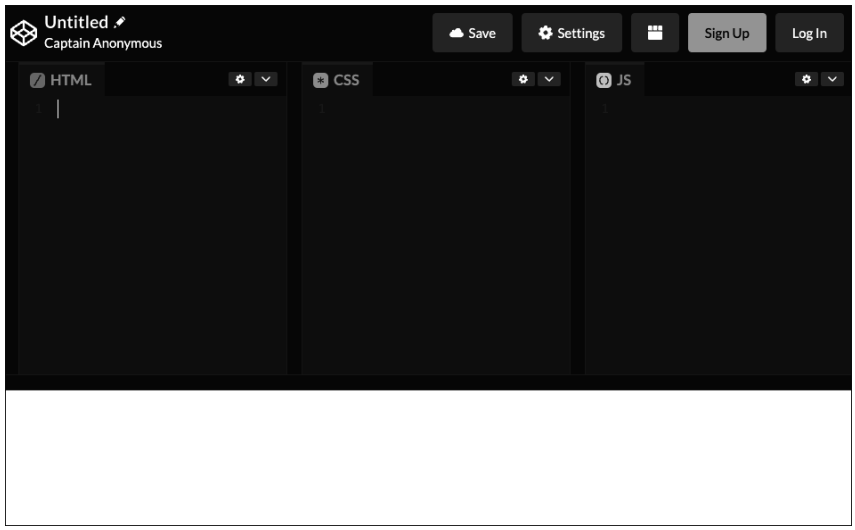


Abbildung 2.1 Ansicht von CodePen

In Abbildung 2.1 sehen Sie die Standardansicht von CodePen. Im nächsten Abschnitt erfahren Sie, wie Sie Schritt für Schritt zu einem React-Playground kommen, auf dem Sie Ihre Experimente durchführen können.

Ein React-Projekt auf CodePen

In der Standardkonfiguration bindet CodePen keinerlei Bibliotheken ein und stellt Ihnen lediglich eine Kombination von HTML, CSS und JavaScript zur Verfügung. React aktivieren Sie in dieser Umgebung, indem Sie auf das Einstellungs-Icon im JavaScript-Editor klicken und dort unter dem Punkt JS zunächst Babel als JavaScript Präprozessor auswählen und anschließend unter Add External Scripts/Pens die beiden Bibliotheken react und react-dom hinzufügen. Mit dieser Konfiguration können Sie mit der Entwicklung Ihrer React-Applikation beginnen.

Babel

Bei *Babel* handelt es sich um einen JavaScript-Compiler, der JavaScript-Quellcode entgegennimmt und wiederum JavaScript-Quellcode produziert. Der Zweck von Babel ist es, Features zu simulieren, die in bestimmten Browsern noch nicht implementiert sind. Babel an sich stellt lediglich die Compiler-Infrastruktur zur Verfügung und kann durch Plugins erweitert werden. Jedes dieser Plugins ist für einen bestimmten Aspekt verantwortlich, beispielsweise für die Unterstützung von Arrow-Funktionen. Mehrere Plugins können zu sogenannten *Presets* zusammengefasst werden. Hierbei handelt es sich um ein Komfortfeature, das es Ihnen ersparen soll, eine große Anzahl von Plugins einzubinden. So gibt es beispielsweise das *React Preset*, das einige JSX-Plugins für Sie gruppiert.

Eine React-Applikation benötigt ein HTML-Element, in das die Applikation eingefügt werden soll. Zu diesem Zweck fügen Sie im HTML-Editor ein `div`-Element mit einem `id`-Attribut und dem Wert `root` ein (siehe Listing 2.4). Der Wert dieses Attributs ist frei wählbar und kann abweichen. In diesem Fall müssen Sie die Referenz beim Bootstrappen der Applikation anpassen.

```
<div id="root"></div>
```

Listing 2.4 Container für die React-Applikation

In Listing 2.5 sehen Sie den JavaScript-Quellcode der Beispiel-Applikation. Dieser besteht aus der Definition einer Komponente und dem Rendern der React-Applikation.

```
const Greet = ({ name }) => <h1>Hallo {name}!</h1>;

const rootElement = document.getElementById('root');
const root = ReactDOM.createRoot(rootElement);
root.render(<Greet name="Leser" />);
```

Listing 2.5 Quellcode der React-Applikation

Im JavaScript-Editor erzeugen Sie zunächst eine einfache Komponente mit dem Namen `Greet`. Achten Sie darauf, dass der Name der Komponente mit einem Großbuchstaben beginnen muss. Die Komponente ist eine Arrow-Funktion, die ein JSX-Element zurückgibt. Als Argument erhält die Funktion ein Props-Objekt, über das Sie auf übergebene Werte zugreifen können. Auf den Wert `name`, den Sie über ein Destructuring-Statement extrahieren, können Sie in der JSX-Struktur mit geschweiften Klammern zugreifen. Die Details zum Aufbau einer Komponente erfahren Sie in den folgenden Kapiteln.

Nach der Definition der Komponente machen Sie sich an das eigentliche Setup der Applikation. Zunächst benötigen Sie eine Referenz auf das Container-Element, in das Sie die Applikation einfügen, also auf das `div`-Element, das Sie zuvor definiert haben. Anschließend erzeugen Sie mit der `createRoot`-Methode aus dem `ReactDOM`-Objekt und der Referenz auf das `div`-Element ein Root-Objekt. Im letzten Schritt rufen Sie die `render`-Methode des Root-Objekts auf und übergeben ihr eine weiter JSX-Struktur. Diese beinhaltet die `Greet`-Komponente als HTML-Tag mit dem Attribut `name`. Dies sorgt dafür, dass React die Komponente rendert und eine Ausgabe wie in Abbildung 2.2 erzeugt.

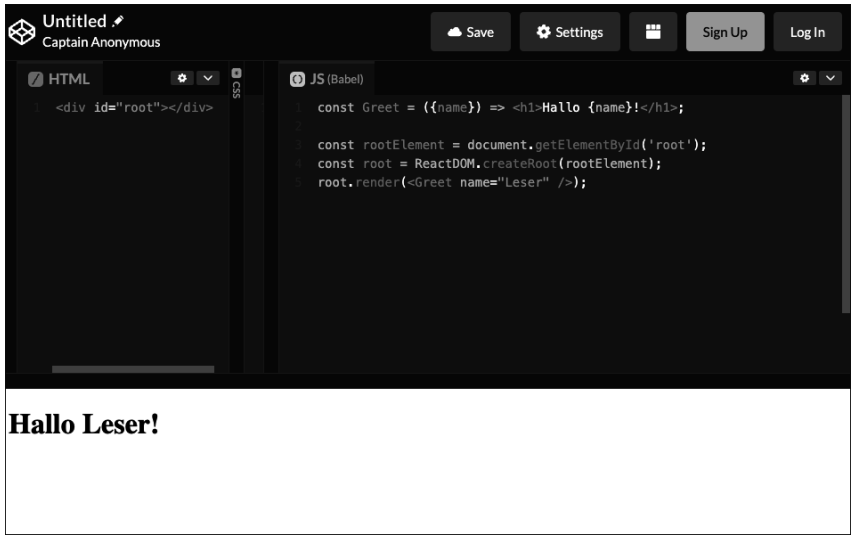


Abbildung 2.2 Ansicht der React-Applikation in CodePen

Die CodePen-Plattform ist einfach gehalten. Das hat zur Konsequenz, dass Sie im Standard-Setup keine Möglichkeit haben, Ihre Applikation in mehrere JavaScript-Dateien zu unterteilen. Andere Plattformen, beispielsweise CodeSandbox, bieten Ihnen zwar dieses Feature, aber dennoch ist die lokale Entwicklung von Applikationen deut-

lich komfortabler und flexibler. Im folgenden Abschnitt erfahren Sie, wie Sie mit der Entwicklung Ihrer React-Applikation auf Ihrem eigenen System starten können.

2.3 Lokale Entwicklung

Dem Vorteil, dass eine Plattform wie CodePen immer und überall verfügbar ist und Sie darüber Ihren Code mit anderen Entwicklern und Entwicklerinnen weltweit teilen können, steht der Nachteil gegenüber, dass das Debugging in einer solchen Umgebung nur umständlich möglich ist. Außerdem benötigen Sie eine bestehende Internetverbindung und haben auch keine Möglichkeit, Ihren Quellcode in einem lokalen Repository vorzuhalten. In den meisten Fällen werden Sie Ihre Applikation lokal auf Ihrem Rechner entwickeln. Auch hierbei gibt es wieder zahlreiche Möglichkeiten. Die einfachste Möglichkeit ist die direkte Integration der Bibliothek in eine Webseite.

2.3.1 React in eine HTML-Seite einbinden

Im Vergleich zu Bibliotheken wie Vue.js muss sich React häufig den Vorwurf gefallen lassen, dass der Einstieg in die Entwicklung vergleichsweise schwierig sei. Während bei Vue.js lediglich eine JavaScript-Datei eingebunden werden muss, müssen Sie bei React erst einen umfangreichen Installationsprozess durchlaufen. Das entspricht jedoch nur teilweise der Wahrheit.

Um React in einer kleinen lokalen Anwendung zu verwenden, gehen Sie ähnlich vor wie bei der Konfiguration eines Playgrounds, beispielsweise CodePen. Zunächst erzeugen Sie eine HTML-Datei mit dem Namen *index.html*:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Hello React!</title>

    <script
      src="https://unpkg.com/react@18/umd/react.development.js"
      crossorigin
    ></script>
    <script
      src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
      crossorigin
    ></script>
    <script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
```

```
    <script src="index.jsx" type="text/babel"></script>
</head>
<body>
    <div id="root"></div>
</body>
</html>
```

Listing 2.6 Einstiegsdatei mit direkter React-Einbindung (»index.html«)

In Listing 2.6 sehen Sie die Struktur dieser Einstiegsdatei. Zunächst müssen Sie dafür sorgen, dass die Bibliotheken `react` und `react-dom` geladen werden. Dies erreichen Sie über das CDN (Content Delivery Network) *unpkg.com*. Damit Sie die komfortablere JSX-Syntax verwenden können, um Ihre Komponenten zu erzeugen, benötigen Sie außerdem Babel, das Sie ebenfalls von *unpkg.com* beziehen können. Mit diesem Setup können Sie schließlich die *index.jsx*-Datei einbinden, die Ihre eigene Applikation enthält. Wichtig hierbei ist, dass Sie das `type`-Attribut mit dem Wert `text/babel` angeben. Damit sorgt Babel dafür, dass die Datei übersetzt wird, sodass der Browser den JSX-Code interpretieren kann. Im `body` Ihrer HTML-Datei definieren Sie den Container, in den Ihre Applikation eingehängt wird. Die Dateiendung *.jsx* signalisiert, dass es sich bei dieser Datei nicht um eine gewöhnliche JavaScript-Datei handelt, sondern dass JSX verwendet wird und deshalb ein Werkzeug wie Babel zur Übersetzung benötigt wird.

Der Inhalt der *index.jsx*-Datei entspricht dem Quellcode, den Sie im ersten Beispiel in CodePen als JavaScript ausgeführt haben. Listing 2.7 fasst diesen noch mal für Sie zusammen:

```
const Greet = ({ name }) => <h1>Hallo {name}!</h1>;

const rootElement = document.getElementById('root');
const root = ReactDOM.createRoot(rootElement);
root.render(<Greet name="Leser" />);
```

Listing 2.7 JavaScript-Quellcode der lokalen React-Applikation

Wenn Sie die *index.html*-Datei lokal in Ihren Browser laden, sehen Sie lediglich eine weiße Seite. Bei einem Blick in die JavaScript-Konsole Ihres Browsers sehen Sie die Fehlermeldung, dass die Datei *index.jsx* nicht geladen werden konnte. Das liegt daran, dass Babel versucht, diese Datei über einen asynchronen Request zu laden, und sol-

che Anfragen aus Sicherheitsgründen nicht über das *file://*-, sondern nur über das *http://*-Protokoll ausgeführt werden können.

Am einfachsten lässt sich dieses Problem beheben, indem Sie Ihre Applikation mit einem lokalen Webserver testen.

Einen lokalen Webserver nutzen

Einige Browserfeatures erfordern es, dass ein HTML-Dokument von einem Webserver und nicht von der lokalen Festplatte eingelesen wird. Um dies zu vereinfachen, existieren zahlreiche Projekte, die Ihnen einen leichtgewichtigen Webserver zur Verfügung stellen. Eines der bekanntesten Projekte trägt den Namen `http-server` und ist als NPM-Paket verfügbar.

Zur Verwendung des `http-server`-Webservers stehen Ihnen mehrere Möglichkeiten zur Verfügung: Sie können ihn entweder lokal, global oder *on demand* installieren.

Lokale Installation

Für die lokale Installation wechseln Sie auf die Konsole Ihres Systems, navigieren in das Verzeichnis, in dem sich Ihre React-Applikation befindet, und führen die folgenden Kommandos aus:

```
npm init -y
npm install http-server
```

Das erste Kommando erzeugt eine *package.json*-Datei für Ihr Projekt, die dessen Beschreibung und Konfiguration wie beispielsweise auch installierte Abhängigkeiten enthält. Der zweite Befehl installiert den `http-server` im aktuellen Verzeichnis im Unterverzeichnis *node_modules*. Daraufhin können Sie den `http-server` mit dem Kommando

```
node_modules/.bin/http-server .
```

starten. Der Server liefert dann den Inhalt des aktuellen Verzeichnisses auf allen verfügbaren Netzwerkschnittstellen Ihres Systems aus, sodass Sie über *http://localhost:8080* auf Ihre Applikation zugreifen können.

Globale Installation

Die globale Installation erreichen Sie auf der Kommandozeile mit dem Befehl

```
npm install -g http-server
```

Dies sorgt dafür, dass das Paket in ein Verzeichnis installiert wird, das sich im Suchpfad des Systems befindet, sodass Sie den `http-server` auf der Konsole mit dem Kommando

```
http-server .
```

ausführen können. Auch hier liefert das Programm wieder den Inhalt des aktuellen Verzeichnisses auf allen Schnittstellen auf dem TCP-Port 8080 aus.

On-Demand-Ausführung

Die dritte und letzte Variante besteht aus der Verwendung des `npx`-Kommandos, das seit Version 5.2 Bestandteil des NPM ist. Mit

```
npx http-server .
```

wird zunächst nach einer lokalen Version des Pakets gesucht; wird es nicht gefunden, wird es heruntergeladen und direkt ausgeführt. Auch hier wird wieder der Inhalt des aktuellen Verzeichnisses verfügbar gemacht. Nach Beendigung der Ausführung wird das eben heruntergeladene Paket wieder verworfen.

Welche Möglichkeit für welchen Zweck?

Generell sollten Sie globale Installationen von Paketen vermeiden, da dies eine implizite Abhängigkeit darstellt und Sie sich auf eine Version für Ihr gesamtes System festlegen. Die lokale Installation eines Pakets lohnt sich immer dann, wenn Sie die Funktionalität öfter als nur einmal benötigen; die On-Demand-Installation deckt den Fall ab, dass Sie eine Funktionalität einmal oder nur wenige Male benötigen.

Wenn Sie nun den `http-server` wie beschrieben ausführen, erreichen Sie Ihre Applikation über die URL *http://localhost:8080*, wo sie fehlerfrei ausgeführt werden sollte und Sie eine Ansicht wie in Abbildung 2.3 erhalten sollten.

Nach diesen beiden Ausflügen in die Ausführung von React widmen wir uns in den folgenden Abschnitten der Variante, die am häufigsten verwendet wird, um mit einer React-Applikation zu arbeiten: dem Projekt *Create React App*.

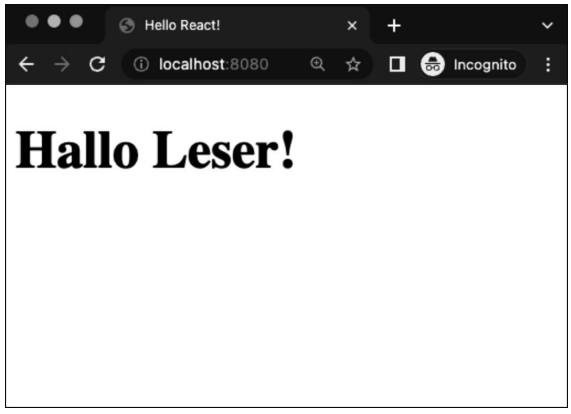


Abbildung 2.3 Lokale Ausführung der React-Applikation

2.4 Der Einstieg in die Entwicklung mit React

Nahezu alle großen JavaScript-Frameworks verfügen über Kommandozeilenwerkzeuge, die Ihnen Routinearbeiten abnehmen, die im Zuge der Entwicklung anfallen.

Die Aufgabe, die im Entwicklungsprozess am meisten Zeit beansprucht, ist das Einrichten der Umgebung. Das liegt vor allem daran, dass zahlreiche Werkzeuge für den Entwicklungs- und Buildprozess zusammengefügt werden müssen.

Typischerweise spielen bei der Entwicklung einer Webapplikation Werkzeuge wie Paketmanager, Bundler wie *Webpack*, *Babel* als Transpiler und noch zahlreiche weitere zusammen. Ein Kommandozeilenwerkzeug wie *Create React App* übernimmt die Koordination dieser Werkzeuge und erstellt eine Basisstruktur für Ihre Applikation.

2.4.1 Anforderungen

Für die Implementierung einer React-Applikation benötigen Sie einige Werkzeuge auf Ihrem Rechner.

Node.js

So sollten Sie über eine aktuelle Version von Node.js verfügen. Diese erhalten Sie entweder direkt von *https://nodejs.org/* als Installer-Paket oder über den Paketmanager Ihres Betriebssystems.

Node.js

React ist zwar eine Frontend-Bibliothek, die unabhängig von Node.js verwendet werden kann. Im Entwicklungsprozess spielt die serverseitige JavaScript-Plattform dennoch eine wichtige Rolle. Viele Werkzeuge, die Sie während der Implementierung nutzen, basieren auf Node.js. Node.js basiert auf der V8-Engine, die auch im Chrome-Browser zum Einsatz kommt. Die Plattform verfügt über eine Reihe von Kernmodulen, die Ihnen beim Entwickeln den Zugriff auf die Ressourcen des Betriebssystems (wie Netzwerk oder Festplatte) erlauben.

Neben der eigentlichen JavaScript-Plattform enthält das Node.js-Paket außerdem *NPM*, einen Paketmanager, mit dem sich die Abhängigkeiten Ihrer Applikation verwalten lassen.

Im Verlauf dieses Kapitels lernen Sie mit *Yarn* eine weitestgehend API-kompatible Alternative zu NPM kennen.

Auf der Kommandozeile können Sie mit den Befehlen

```
$ node -v
V19.2.0
$ npm -v
8.19.3
```

prüfen, ob die beiden Werkzeuge korrekt installiert sind.

Editor

Neben Node.js sollten Sie einen Editor auf Ihrem System installieren, mit dem Sie den Quellcode Ihrer Applikation schreiben. Ich empfehle Ihnen, entweder den kostenlosen Editor *Visual Studio Code* von Microsoft oder die kostenpflichtige Entwicklungsumgebung *WebStorm* von JetBrains zu nutzen. Grundsätzlich können Sie jedoch jeden beliebigen Editor verwenden. Sie sollten allerdings darauf achten, dass dieser über Komfortfeatures wie beispielsweise Syntax-Highlighting für JavaScript, TypeScript und im besten Fall auch für React und JSX verfügt und Ihnen *Code Completion* bietet, also die korrekte Vervollständigung von begonnenen Variablen- und Methodennamen.

Browser

React kann in verschiedenen Umgebungen ausgeführt werden. Normalerweise werden React-Anwendungen jedoch im Browser ausgeführt. Also sollten Sie für die Entwicklung zumindest über eine aktuelle Version eines der vier Hauptbrowser verfügen – also Chrome, Firefox, Edge oder Safari. Die Beispiele in diesem Buch, sofern sie spezifische Browserfeatures betreffen, beziehen sich auf Chrome, weil dieser Browser so weitverbreitet ist. Funktionalitäten wie die Entwicklerwerkzeuge finden Sie auch in den anderen Browsern in ähnlicher Form und in einem ähnlichen Umfang wieder.

React unterstützt alle modernen Browser. Mit der Version 18 von React fällt die Unterstützung für den Internet Explorer weg. Microsoft stellte die Unterstützung für diesen Browser ohnehin am 15.06.2022 ein. Falls Sie dennoch ältere Browserversionen oder gar den Internet Explorer unterstützen wollen, müssen Sie verschiedene *Polyfills* installieren. Diese sind im Paket `react-app-polyfill` zusammengefasst und lassen sich über einen Paketmanager installieren. Anschließend müssen Sie die für Ihren Browser passende Version lediglich noch einbinden. Listing 2.8 demonstriert dies am Beispiel des Internet Explorers 11:

```
import 'react-app-polyfill/ie11';
```

Listing 2.8 Polyfills für den Internet Explorer 11 laden

Polyfill

Als *Polyfill* bezeichnet man ein Stück JavaScript-Quellcode, mit dessen Hilfe ein im Browser nicht vorhandenes Feature emuliert wird. Polyfills sind in der Regel deutlich leistungsschwächer als die eigentlichen Browser-Features. Hier geht es jedoch hauptsächlich darum, keine Benutzer*innen auszuschließen, und weniger um Performance.

2.4.2 Installation von Create React App

Die bisher vorgestellten Varianten, um mit der Entwicklung mit React zu starten, haben den Nachteil, dass sie zwar einen schnellen Start in die Entwicklung erlauben, der Entwicklungskomfort jedoch auf der Strecke bleibt. Dieses Problem löst das Projekt *Create React App*. Dabei handelt es sich um ein Kommandozeilenwerkzeug, das von Facebook entwickelt wird. Die Website des Projekts finden Sie unter *https://create-react-app.dev/*. Im Gegensatz zu anderen CLI-Werkzeugen (wie beispielsweise dem Angular-CLI), die Sie über den gesamten Lebenszyklus einer Applikation begleiten, unterstützt Create React App Sie lediglich beim Erstellungsprozess der Applikation.

Create React App nimmt Ihnen nahezu alle Arbeiten ab, die bei der Initialisierung einer Applikation anfallen. Das Werkzeug erzeugt die Basisstruktur der Applikation, lädt alle benötigten Abhängigkeiten herunter und bereitet alles so weit vor, dass Sie direkt mit der Entwicklung beginnen können. Die erzeugte Applikation nutzt in der Community etablierte Pakete für Standardaufgaben, beispielsweise *Webpack* als Bundler und *Babel* als Transpiler. Die Konfiguration der eingesetzten Werkzeuge wird jedoch standardmäßig von Create React App übernommen und vor Ihnen versteckt, sodass Sie hiermit nicht in Berührung kommen. In den meisten Fällen reicht die Standardkonfiguration auch aus. Sollte dies für Ihre Applikation nicht der Fall sein, haben Sie die Möglichkeit, sich die Konfiguration exportieren zu lassen und sie entsprechend anzupassen. Wie das genau funktioniert, erfahren Sie in Abschnitt 2.4.4, »React Scripts«.

Ein weiteres Komfortfeature, das *Webpack* Ihnen bietet, ist der *Webpack-Dev-Server*, ein lokaler Webserver, über den Sie Ihre Applikation direkt testen können. Während der Entwicklung werden bei jeder Änderung am Quellcode die betroffenen Dateien neu verarbeitet und in die laufende Applikation eingefügt. Die Infrastruktur sorgt dafür, dass der Browser automatisch neu geladen wird, sodass die Änderungen sofort wirksam werden. Für den Produktivbetrieb sollten Sie den Dev-Server keinesfalls verwenden, da ein Reload der Applikation das Zurücksetzen des States der Applikation im Browser zur Folge hat.

Das vorrangige Ziel von Create React App ist, Ihnen einen schnellen Start in die Entwicklung zu ermöglichen und dabei die Komplexität möglichst gering zu halten. Das macht sich zum einen durch die bereits erwähnte Standardkonfiguration bemerkbar und zum anderen durch den Umgang mit Abhängigkeiten: Werfen Sie einen Blick in die Liste der direkt installierten Abhängigkeiten in der *package.json*-Datei einer frischen React-Applikation, die mit Create React App erzeugt wurde, dann finden Sie insgesamt drei installierte Bibliotheken: `react`, `react-dom` und `react-scripts`. Alle weiteren erforderlichen Abhängigkeiten sind wiederum Abhängigkeiten dieser drei Bibliotheken, was auch den Updateprozess einer Applikation erheblich vereinfacht.

Doch nun genug der einleitenden Worte, beginnen wir mit der Entwicklung unserer Applikation! Für die Installation von Create React App stehen Ihnen mehrere Möglichkeiten zur Verfügung, je nachdem, für welchen Paketmanager und welche Strategie Sie sich entscheiden.

Initialisierung mit »npm init« und »npx«

Für JavaScript existiert zwar eine ganze Reihe von Paketmanagern, NPM ist jedoch mit Abstand der am meisten verwendete. Die einfachste Variante, Ihre React-Applikation zu initialisieren, besteht aus der Verwendung des `npm init`-Kommandos. Mit

```
npm init react-app library
```

Listing 2.9 Initialisierung einer React-Applikation mit »npm init«

erzeugen Sie ebenfalls ein lokales Verzeichnis *library*, in dem Create React App die Struktur für Ihre Applikation aufbaut. Ist Create React App nicht auf Ihrem System installiert, lädt NPM es temporär herunter, führt es aus und verwirft es anschließend wieder. Normalerweise verwenden Sie das Kommando `npm init`, um in einem interaktiven Prozess eine neue *package.json*-Datei zu erzeugen. In unserem Fall übergeben Sie zusätzlich den Initialisierer `react-app`. NPM verwendet dann das Werkzeug `npx`, um das Paket mit dem Namen `create-` und der Bezeichnung des Initialisierers herunterzuladen und es auszuführen. Alternativ können Sie `npx` auch direkt nutzen. In diesem Fall lautet der zugehörige Befehl:

```
npx create-react-app library
```

Listing 2.10 Erzeugung einer neuen React-Applikation mit »npx«

Die Wirkung beider Kommandos ist äquivalent und Sie können im Anschluss direkt mit der Arbeit an Ihrer Applikation beginnen.

Der Node Package Manager (NPM)

Der Name dieses Werkzeugs ist etwas irreführend. NPM ist ein Paketmanager für JavaScript, den Sie sowohl für die serverseitige Entwicklung mit Node.js als auch zur Verwaltung von Abhängigkeiten im Frontend verwenden können. NPM wird als unabhängiges Open-Source-Projekt entwickelt und ist Bestandteil der meisten Node.js-Installationen. Auf der Kommandozeile ist NPM unter dem Kommando `npm` systemweit verfügbar.

Neben dem `npm`-Befehl gibt es weitere wichtige Bestandteile von NPM in Ihrem Projekt: Die *package.json*-Datei, die sich normalerweise im Wurzelverzeichnis Ihrer Applikation befindet, enthält die Beschreibung Ihrer Applikation. Im *node_modules*-Ver-

zeichnis werden die installierten Pakete gespeichert. Dieses Verzeichnis wird normalerweise nicht mit der Applikation versioniert. Stattdessen führen Sie, wenn Sie eine bestehende Applikation aus dem Versionskontrollsystem neu aufbauen wollen, das Kommando `npm install` aus, um alle Abhängigkeiten, die in der *package.json*-Datei aufgeführt sind, zu installieren.

Neben der *package.json*-Datei existiert außerdem die *package-lock.json*-Datei. In dieser Datei sind sämtliche installierten Abhängigkeiten Ihrer Applikation sowie deren Abhängigkeiten aufgelistet. Mit dieser Datei wird sichergestellt, dass jede Installation Ihrer Applikation der anderen gleicht. In der *package-lock.json* finden Sie die Paketnamen, ihren Speicherort in der NPM-Registry und einen Integritäts-Hash, der vor Manipulationen des Pakets schützt. Sie sollten sowohl die *package.json* als auch die *package-lock.json* Ihres Projekts versionieren.

Sie können mit NPM Pakete installieren, aktualisieren und auch wieder von Ihrem System entfernen. Die folgende Liste stellt Ihnen die wichtigsten Kommandos vor:

- `npm install <Paketname>`: Lädt das angegebene Paket aus dem NPM-Respository herunter und installiert es. Das Paket wird automatisch als Abhängigkeit in die *package.json*-Datei Ihres Projekts eingetragen. Mit der Option `–save-dev` können Sie angeben, dass es sich um eine Abhängigkeit handelt, die nur während der Entwicklung benötigt wird.
- `npm update <Paketname>`: Aktualisiert das angegebene Paket im Rahmen der erlaubten Versionsspanne, die in der *package.json*-Datei angegeben ist.
- `npm remove <Paketname>`: Entfernt das Paket.
- `npm list`: Listet die aktuell installierten Pakete auf.
- `npm init`: Erzeugt eine *package.json*-Datei.

Hinweis

Im weiteren Verlauf dieses Buchs nutze ich NPM als Paketmanager. Sämtliche Beispiele können Sie jedoch mit geringen Anpassungen auch mit *Yarn* oder einem beliebigen anderen Paketmanager wie beispielsweise *pnpm* nachvollziehen.

Verzichten Sie auf die globale Installation über NPM

Eine früher sehr häufig anzutreffende Variante für die Verwendung von Create React App war die globale Installation des Werkzeugs mit der Option `-g`, die Sie im folgenden Kommando sehen:

```
npm install -g create-react-app
```

Listing 2.11 Create React App mit NPM global installieren

Die globale Installation sorgt dafür, dass Ihnen das Werkzeug systemweit auf der Kommandozeile zur Verfügung steht. Der Nachteil ist, dass Sie dann systemweit nur eine festgelegte Version von Create React App haben und Sie sich selbst um die Aktualisierung des Werkzeugs kümmern müssen. Da Sie Create React App nur einmal für eine Applikation benötigen – und zwar bei der Initialisierung –, ist das Risiko, dass das Werkzeug veraltet, sehr hoch. Mittlerweile verbietet Create React App die globale Installation sogar: Versuchen Sie, das Werkzeug in diesem Modus zu verwenden, erhalten Sie die Fehlermeldung »We no longer support global installation of Create React App«. In diesem Fall bleibt Ihnen nichts anderes übrig, als die globale Installation zu deinstallieren und eine der zuvor beschriebenen Varianten zu nutzen.

Neben NPM können Sie zur Initialisierung Ihrer Anwendung mit Yarn auf eine vollwertige Alternative zurückgreifen.

Yarn

Yarn ist ein Paketmanager für JavaScript von Facebook. Das Werkzeug ist Open Source und kostenlos nutzbar.

Installiert wird Yarn je nach Betriebssystem entweder über ein Installer-Paket oder über den Paketmanager Ihres Systems. Weitere Informationen zur Installation finden Sie unter *https://yarnpkg.com/getting-started/install*.

Yarn ist weitestgehend API-kompatibel mit NPM und verfügt über einen annähernd gleichen Funktionsumfang. An dieser Stelle werden Sie sich berechtigterweise fragen, warum Sie sich neben dem etablierten NPM noch mit einem weiteren Paketmanager beschäftigen sollten. Der Grund liegt in der Entwicklungsgeschichte von Yarn. Werfen Sie einen Blick auf die Website von Yarn unter *https://yarnpkg.com*, stechen Ihnen die drei Schlagwörter »fast«, »reliable« und »secure« ins Auge. Das sind die drei wichtigsten Bereiche, in denen die früheren Versionen von NPM eher schwach aufgestellt waren. Um diese Probleme zu adressieren, schufen einige Entwickler von Facebook den Paketmanager Yarn.

- **Fast:** Yarn verfügt über einen Caching-Mechanismus, der dafür sorgt, dass einmal installierte Pakete nicht noch einmal heruntergeladen werden müssen, sondern direkt aus dem lokalen Cache ausgeliefert werden können. Außerdem ist Yarn in der Lage, Downloads von Paketen zu parallelisieren und so die Zeit der Installation weiter zu verkürzen.
- **Reliable:** Mit der *yarn.lock*-Datei sorgt Yarn dafür, dass sich mehrere Installationen Ihrer Applikation auch auf unterschiedlichen Systemen gleichen. In älteren Versionen von NPM wurden lediglich die Versionen der direkt installierten Pakete festgehalten, nicht aber die ihrer Abhängigkeiten, was teilweise zu schwerwiegenden Problemen führte. In der *yarn.lock*-Datei sind sämtliche zu installierenden Pakete sowie deren komplette Abhängigkeitsbäume festgelegt und mit Integritäts-Hashes versehen.

- **Secure:** Die Integritäts-Hashes in der *yarn.lock*-Datei stellen sicher, dass die Pakete nicht nachträglich manipuliert werden können. Wird auch nur ein kleiner Teil des Pakets angepasst, stimmt der Hashwert nicht mehr überein und die Installation schlägt fehl.

Die Aussage, dass Konkurrenz das Geschäft belebt, gilt auch für den Markt der Paketmanager. Nach dem Erscheinen von Yarn haben die Entwickler von NPM schnell nachgezogen und die größten Schwachstellen in NPM behoben, sodass beide Werkzeuge sich mittlerweile auf Augenhöhe begegnen.

Der große Vorteil für Sie ist, dass NPM und Yarn die gleiche Infrastruktur nutzen. Ist ein Paket unter NPM verfügbar, können Sie es auch mit Yarn installieren und umgekehrt. Außerdem nutzen beide Paketmanager das *node_modules*-Verzeichnis zum Speichern der Abhängigkeiten und die *package.json*-Datei, um die wichtigsten Konfigurationen Ihrer Applikation festzuhalten. Lediglich in einigen Kommandos unterscheiden sich Yarn und NPM. Installieren Sie unter NPM Ihre Pakete mit `npm install react`, erreichen Sie dies unter Yarn mit dem Kommando `yarn add react`.

Initialisierung mit »yarn create«

Yarn bietet Ihnen eine Alternative zu NPM. Ähnlich wie bei NPM installieren Sie auch mit Yarn Create React App nicht auf Ihrem System. Mit dem folgenden Kommando nutzen Sie eine temporäre Variante des Werkzeugs, um Ihre Applikation zu initialisieren. Danach wird die Installation von Create React App wieder verworfen.

```
yarn create react-app library
```

Listing 2.12 Initialisierung einer React-Applikation mit Yarn

Egal für welche Variante der Installation Sie sich entschieden haben, nach der Ausführung des jeweiligen Kommandos verfügen Sie über eine voll funktionsfähige React-Applikation, mit der Sie arbeiten können. Bevor wir im nächsten Schritt in die Struktur der Applikation einsteigen, folgen nun noch einige weiterführende Informationen zu Create React App.

Kommandozeilenoptionen von Create React App

Im Normalfall reicht es aus, wenn Sie Create React App nur mit dem Verzeichnisnamen der zu erstellenden Applikation aufrufen. Darüber hinaus bietet das Werkzeug einige hilfreiche Optionen, die ich Ihnen im Folgenden kurz vorstellen möchte:

- `-h`, `--help`: Diese Option zeigt Ihnen eine Liste der verfügbaren Optionen an.
- `-V`, `--version`: Mithilfe dieser Option lassen Sie sich die Version von Create React App ausgeben.

- `--verbose`: Diese Option aktiviert eine umfangreichere Ausgabe. Diese kommt vor allem bei der Fehlersuche zum Einsatz.
- `--info`: Auch diese Option wird häufig beim Debugging eingesetzt. Es werden Umgebungsinformationen ausgegeben, beispielsweise die Betriebssystem- und Browserversionen.
- `--use-npm`: Ist Yarn auf dem System installiert, auf dem Sie Create React App ausführen, wird es standardmäßig als Paketmanager verwendet. Mit der Option `–use-npm` können Sie die Verwendung von NPM erzwingen.
- `--use-pnp`: Plug-and-play ist ein relativ neues Feature von Yarn. Mit ihm ist es möglich, die Installation von Abhängigkeiten einer React-Applikation erheblich zu beschleunigen, da die Pakete nicht ins *node_modules*-Verzeichnis kopiert werden. Mit dieser Option aktivieren Sie das Plug-and-play-Feature für Ihre React-Applikation.
- `--scripts-version`: Das Paket `react-scripts` ist das Herzstück einer Applikation, die Sie mit Create React App erzeugen. Mit dieser Option können Sie die Version der `react-scripts` angeben. Dabei haben Sie nicht nur die Auswahl zwischen verschiedenen Varianten von konkreten Versionsnummern und alternativen Paketen, die über NPM verfügbar sind, sondern können auch lokale Verzeichnisse oder Archive einsetzen.
- `--template`: Mit der Option `--template` können Sie ein benutzerdefiniertes Template für Ihre Applikation angeben. Hier haben Sie mehrere Optionen, beispielsweise ein offizielles *Create React App*-Template, ein lokales Verzeichnis oder eine Archivdatei. Beispiele für Templates, die Create React App Ihnen zur Verfügung stellt, sind `typescript` oder `cra-template-pwa`.

Vor allem die neueren Optionen wie `–use-pnp` oder `–template` zeigen, in welche Richtung sich Create React App weiterentwickelt: von einem reinen Hilfsmittel zur Initialisierung einer Applikation hin zu einem flexiblen Werkzeug, das Ihnen eine moderne Entwicklung und vielfältige Wahlmöglichkeiten bietet.

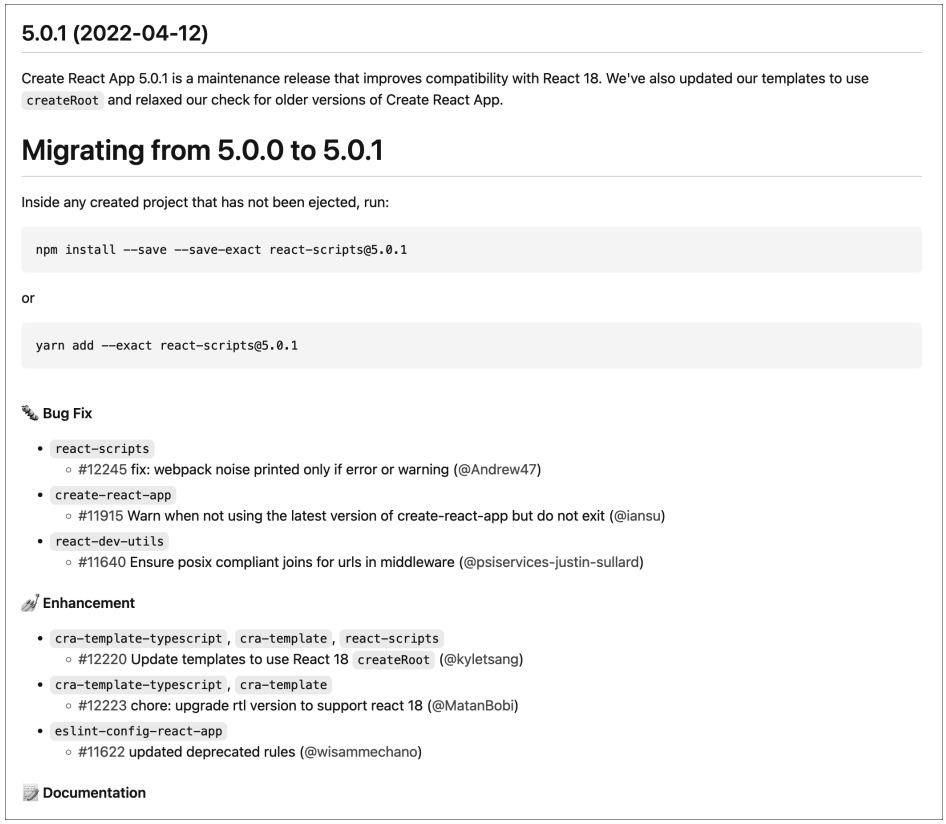
Aktualisierung einer bestehenden Applikation

Entwickeln Sie über längere Zeit an einer Applikation, kommen Sie früher oder später in die Situation, dass eine neue Version des `react-scripts`-Pakets veröffentlicht wird und Sie Ihre Applikation aktualisieren sollten, um von Verbesserungen profitieren zu können. Ob eine solche Aktualisierung ansteht, finden Sie heraus, indem Sie das Kommando `npm outdated` im Verzeichnis Ihrer Applikation ausführen.

Um beispielsweise von Version 5.0.0 auf Version 5.0.1 zu aktualisieren, verwenden Sie den folgenden Befehl auf der Kommandozeile im Verzeichnis Ihrer Applikation:

```
npm install react-scripts@5.0.1
```

Listing 2.13 Aktualisieren des »react-scripts«-Pakets



5.0.1 (2022-04-12)

Create React App 5.0.1 is a maintenance release that improves compatibility with React 18. We've also updated our templates to use `createRoot` and relaxed our check for older versions of Create React App.

Migrating from 5.0.0 to 5.0.1

Inside any created project that has not been ejected, run:

```
npm install --save --save-exact react-scripts@5.0.1
```

or

```
yarn add --exact react-scripts@5.0.1
```

🐛 Bug Fix

- `react-scripts`
  - #12245 fix: webpack noise printed only if error or warning (@Andrew47)
- `create-react-app`
  - #11915 Warn when not using the latest version of create-react-app but do not exit (@iansu)
- `react-dev-utils`
  - #11640 Ensure posix compliant joins for urls in middleware (@psiservices-justin-sullard)

🚀 Enhancement

- `cra-template-typescript`, `cra-template`, `react-scripts`
  - #12220 Update templates to use React 18 `createRoot` (@kyletsang)
- `cra-template-typescript`, `cra-template`
  - #12223 chore: upgrade rtl version to support react 18 (@MatanBobi)
- `eslint-config-react-app`
  - #11622 updated deprecated rules (@wisammechano)

📝 Documentation

Abbildung 2.4 Auszug aus dem Changelog von Create React App

Ein solches Update sollten Sie jedoch nicht leichtfertig durchführen – gerade bei Major-Releases, also einer Aktualisierung der Hauptversionsnummer, kann es zu Breaking Changes kommen, die die Funktionsfähigkeit Ihrer Applikation einschränken können. Um dies zu vermeiden, sollten Sie vor jeder Aktualisierung zunächst das Changelog unter *https://github.com/facebook/create-react-app/blob/master/CHANGELOG.md* prüfen (siehe Abbildung 2.4). Hier finden Sie neben verschiedenen Kategorien wie *Bug Fix* oder *Enhancement* auch Informationen zur Migration auf die neue Version und eventuelle Breaking Changes, die Sie beachten müssen.

Nachdem Sie das Update durchgeführt haben, müssen Sie Ihre Applikation neu starten, damit die Änderungen wirksam werden.

Die Bibliotheken `react` und `react-dom` müssen Sie unabhängig von `react-scripts` separat aktualisieren. Auch hier hilft Ihnen die regelmäßige Ausführung von `npm out-`

`dated`, um über neue Releases informiert zu werden. Das Entwicklerteam von React pflegt ebenfalls ein Changelog mit den Änderungen, die ein neues Release der Bibliothek mit sich bringt. Das Changelog finden Sie unter *https://github.com/facebook/react/blob/master/CHANGELOG.md*.

Automatische Aktualisierung des Quellcodes mit »react-codemod«

Bei einigen Aktualisierungen von React werden Änderungen am bestehenden Code der Applikation erforderlich. Damit Sie nun nicht jede Stelle in Ihrer Applikation von Hand anpassen müssen, existiert das Projekt `react-codemod`. Die Kombination aus *JSCodeshift* und den *Codemod*-Scripts erlaubt es Ihnen, bestimmte Änderungen an Ihrem Quellcode automatisiert durchführen zu lassen. Was zunächst etwas gefährlich klingt, wird bei Facebook im großen Stil an produktivem Code regelmäßig durchgeführt.

Bei Änderungen an zentralen Schnittstellen ist die Anpassung des Quellcodes von Hand keine Option und auch die Manipulation über reguläre Ausdrücke gelangt schnell an ihre Grenzen. Daher implementierten die Entwickler von Facebook das Werkzeug *JSCodeshift*. Es verwendet *recast*, um einen *Abstract Syntax Tree* (AST) in einen veränderten AST zu transformieren. Diese Veränderungen können beispielsweise die erwähnten Änderungen der Schnittstellen sein. Dabei wird der Coding-Stil so gut wie möglich beibehalten.

2.4.3 Alternativen zu Create React App

Der Einsatz von Create React App ist nur eine Variante, wie Sie die Entwicklung einer React-Applikation starten können. Es gibt zahlreiche weitere Möglichkeiten, ohne dass Sie gleich alle Strukturen selbst aufsetzen müssen. Populäre Alternativen sind beispielsweise:

- **Vite**: Vite ist ein Buildtool, also eine Alternative zu Webpack. Es ist modular aufgebaut und bietet Ihnen Templates für verschiedene Setups, ähnlich wie es bei Create React App der Fall ist. Die Templates, die im Zusammenhang mit React relevant sind, sind `react` und `react-ts`. Das `react`-Template baut eine React-Applikation auf Basis von JavaScript auf. Das `react-ts`-Template integriert darüber hinaus noch TypeScript in den Entwicklungsprozess. Mit dem Kommando `npm create vite library -- --template react-ts` erzeugen Sie eine neue React-Applikation mit dem `react-ts`-Template. Weitere Informationen zum Thema »Setup von React-Applikationen« finden Sie unter *https://vitejs.dev/guide/*.
- **Parcel**: Parcel reiht sich ebenfalls in die Reihe der Web-Buildtools ein. Der Featureumfang ist ähnlich wie bei Webpack und Vite. Nur beim Setup ist etwas mehr Arbeit notwendig als mit den eleganten Template-Lösungen von Create React App und Vite. Die Dokumentation von Parcel beinhaltet eine Schritt-für-Schritt-Anlei-

tung für den Aufbau einer React-Applikation, diese finden Sie unter *https://parceljs.org/recipes/react/*.

- **Snowpack**: Snowpack nutzen Sie ähnlich wie die anderen Werkzeuge für den Buildprozess von Webapplikationen. Snowpack ist im Vergleich zu seinen Konkurrenten Parcel und Webpack sehr leichtgewichtig gehalten und kommt nahezu ohne Konfiguration aus. Wie schon Vite bietet es auch ein Template-System, in dem Sie mit nur einem Kommando eine lauffähige React-Applikation erhalten. Auf der Kommandozeile können Sie das Kommando `npx create-snowpack-app react-snowpack –template @snowpack/app-template-minimal` nutzen, um Ihre Applikation aufzusetzen. Weitere Informationen finden Sie in der Dokumentation unter *www.snowpack.dev/tutorials/react*.

Darüber hinaus gibt es Frameworks, die auf React basieren, wie *RedwoodJS* oder *Next.js*. Diese Frameworks enthalten eigene Kommandozeilen-Werkzeuge, die Ihnen bei der Initialisierung und beim Aufbau Ihrer Applikation helfen.

2.4.4 React Scripts

Neben den Abhängigkeiten, die Sie für die Entwicklung benötigen, stellt Ihnen das Paket `react-scripts` ein weiteres Hilfsmittel zur Verfügung: die *React Scripts*. Dabei handelt es sich um ein ausführbares Programm, mit dem Sie Ihre Applikation starten oder für den Produktivbetrieb bauen können. Die verschiedenen Scripts stehen Ihnen als Einträge in der *package.json*-Datei unter der `scripts`-Sektion zur Verfügung. Insgesamt gibt es vier verschiedene Scripts mit unterschiedlichen Bedeutungen:

- `start`: Das Start-Script startet den *Webpack-Dev-Server* und führt Ihre Applikation aus. In der Standardkonfiguration erreichen Sie Ihre Applikation unter der URL *http://localhost:3000*. Ist der Port bereits von einer anderen Applikation belegt, erhalten Sie eine entsprechende Fehlermeldung auf der Konsole. Mit der `PORT`-Umgebungsvariablen können Sie einen alternativen Port auswählen.

  In Listing 2.14 sehen Sie, wie Sie statt Port `3000` alternativ Port `8181` nutzen können:

  ```
  PORT=8181 npm start
  ```

  Listing 2.14 Die Applikation auf einem alternativen Port ausführen

  In einer Applikation, die Sie mit Create React App erzeugen, können Sie beispielsweise das ECMAScript-Modulsystem mit den Schlüsselwörtern `import` und `export` verwenden. Damit das in allen Browsern funktioniert, kommt eine Kombination von Babel und Webpack zum Einsatz. Ein positiver Seiteneffekt der verwendeten Werkzeuge ist, dass die Applikation bei Änderungen automatisch neu geladen wird.

Es lohnt sich auch, ab und zu einen Blick auf die Konsole zu werfen, da hier auch Fehler- und Warnmeldungen ausgegeben werden, falls es im Buildprozess zu Problemen kommt. Generell empfiehlt sich hier, eine Zero-Warning-Policy zu verfolgen, also eine Konsolenausgabe ohne Warnungen.

- `build`: Bei der Entwicklung sollten Sie Ihre Applikation in möglichst viele kleine Komponenten unterteilen. Jede dieser Komponenten hat genau einen Zweck und liegt in einer separaten Datei. Das `build`-Script dient dazu, die vielen Komponenten- und Hilfsdateien zu einem Applikationsbundle zusammenzupacken und dieses so zu optimieren, dass es möglichst wenig Speicherplatz benötigt, was sich positiv auf die benötigte Bandbreite auswirkt, wenn die Applikation zum Client ausgeliefert wird. Führen Sie dieses Script aus, finden Sie das Ergebnis im *build*-Verzeichnis Ihrer Applikation.
- `test`: Die initiale Applikation ist bereits für Unittests vorbereitet. Alle erforderlichen Bibliotheken sind installiert, und die Konfiguration ist vorbereitet. Für die `App`-Komponente, die Wurzelkomponente, die Create React App für Sie standardmäßig erzeugt, ist auch bereits ein einfacher Test vorbereitet. Diesen können Sie mithilfe des `test`-Scripts ausführen.
- `eject`: Create React App konfiguriert die Applikation so, dass Sie nicht an die Konfiguration kommen. Es gibt allerdings Situationen, in denen Sie die Webpack-Konfiguration bearbeiten möchten. Gerade, wenn Sie zusätzliche Webpack-Plugins einbinden, müssen Sie diese konfigurieren. Das `eject`-Script extrahiert die Webpack-Konfiguration aus Ihrer Applikation, sodass Sie sie selbst modifizieren können. Mit dem `eject`-Script müssen Sie jedoch vorsichtig sein. Führen Sie es einmal aus, wird die Konfiguration exportiert. Dieser Prozess ist jedoch nicht mehr umkehrbar. Haben Sie also einmal die Konfiguration »ejectet«, können Sie sie nicht wieder zurückbringen.

Die vier Scripts sind in der *package.json*-Datei in der `scripts`-Sektion hinterlegt und können über den Paketmanager ausgeführt werden. Um Ihre Applikation also im Entwicklungsmodus zu starten, führen Sie das folgende Kommando im Wurzelverzeichnis Ihrer Applikation aus:

```
npm start
```

Listing 2.15 Starten der Applikation im Entwicklungsmodus

Das Script bereitet die Applikation für den Einsatz vor und startet den Dev-Server. Als Ausgabe erhalten Sie eine entsprechende Erfolgsmeldung (siehe Abbildung 2.5).

Die Ausführung von `npm start` sorgt außerdem dafür, dass Ihre Applikation im Standardbrowser Ihres Systems geöffnet wird. Als Ergebnis erhalten Sie im Browser die Ansicht aus Abbildung 2.6.

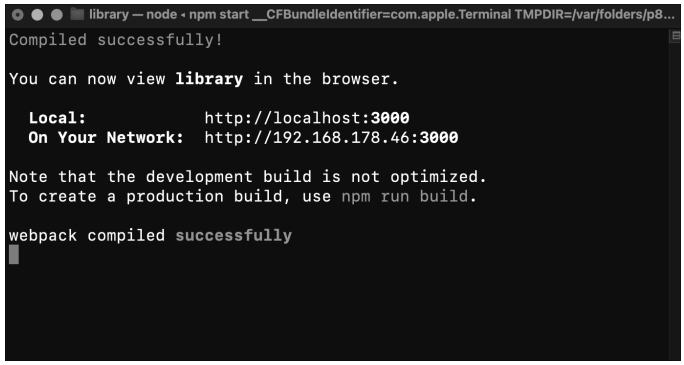


Abbildung 2.5 Konsolenausgabe von »npm start«

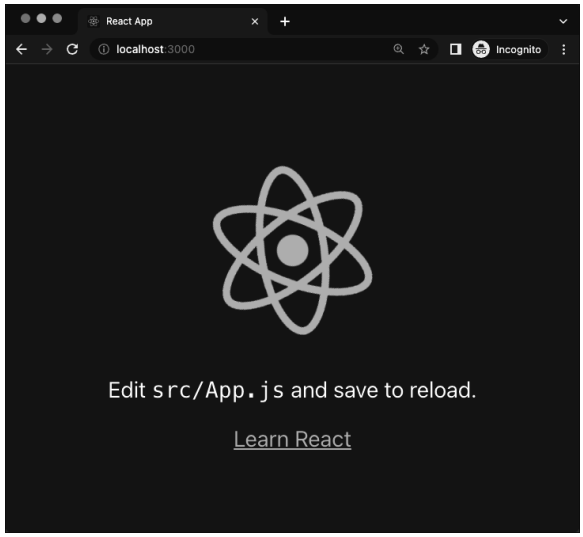


Abbildung 2.6 Die initiale React-Applikation

Die React-Scripts weisen noch einige praktische Erweiterungen mehr auf, die ich Ihnen nicht vorenthalten möchte.

Achtung

NPM weist beim Umgang mit den Scripts eine Besonderheit auf. Es gibt die sogenannten Standardscripts wie `start` oder `test`, die Sie direkt mit `npm start` beziehungsweise `npm test` ausführen können. Neben diesen können Sie beliebige weitere Scripts definieren. Zum Ausführen dieser benutzerdefinierten Scripts müssen Sie das `run`-Kommando verwenden. Bei `build` und `eject` handelt es sich um solche benutzerdefinierten Scripts. Für einen Build Ihrer Applikation führen Sie also auf der Kommandozeile den Befehl `npm run build` aus.

2.4.5 Serverkommunikation im Entwicklungsbetrieb

Bei Single-Page-Applikationen ist die Trennung zwischen Frontend und Backend in der Regel sehr strikt. Auf der einen Seite steht die React-Applikation im Browser und auf der anderen Seite eine Serverschnittstelle, die mit einer beliebigen Programmiersprache umgesetzt sein kann. Während der Entwicklung nutzen Sie den Webpack-Dev-Server, um Ihre Frontend-Applikation auszuliefern. Außerdem können Sie Komfortfeatures wie das automatische Neuladen bei Änderungen am Code nutzen. Das serverseitige Backend kümmert sich in einer solchen Applikation um die Persistierung von Daten und weitere Aspekte wie beispielsweise die Authentifizierung von Benutzern und Benutzerinnen. Der Dev-Server liefert das Backend nicht aus. Es läuft als separater Serverprozess. Das bedeutet, dass das Frontend und das Backend über zwei verschiedene URLs erreichbar sind. Sicherheitsmechanismen im Browser verhindern, dass Sie ohne Weiteres zwischen verschiedenen Servern kommunizieren können. In Abbildung 2.7 sehen Sie die Fehlermeldung, die Sie bei dem Versuch erhalten, auf ein entferntes Backend (*http://google.de* in diesem Fall) zuzugreifen.

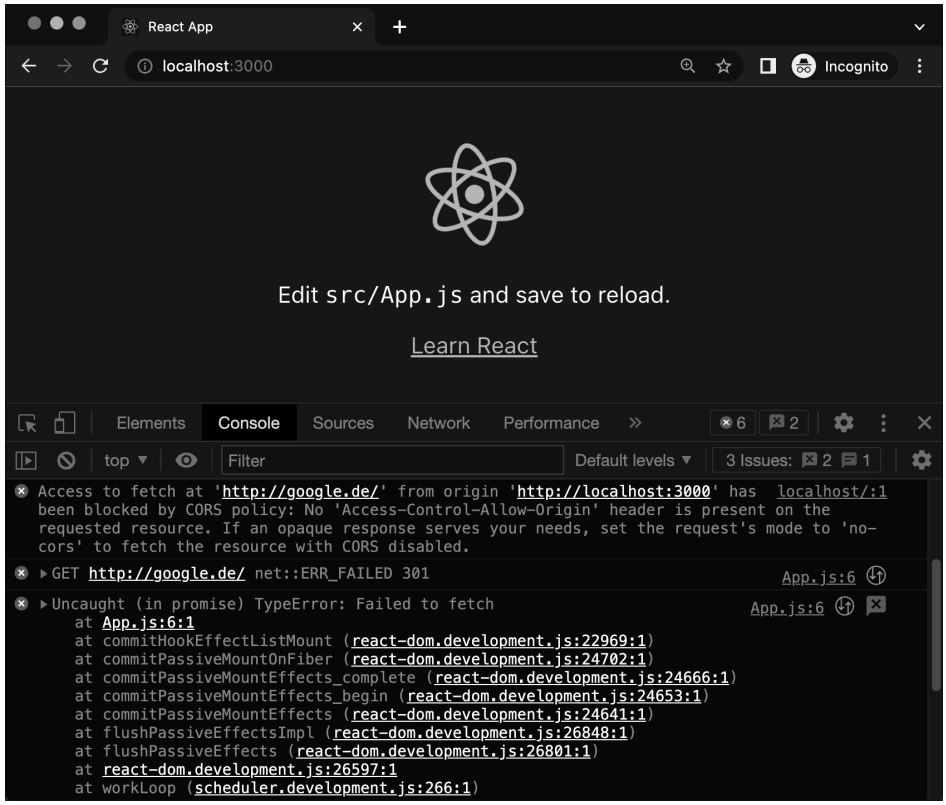


Abbildung 2.7 Fehlermeldung beim Zugriff auf eine Backend-Schnittstelle

Um derartige Fehlermeldungen zu vermeiden, verfügt der Webpack-Dev-Server über eine Proxy-Erweiterung. Diese sorgt dafür, dass sämtliche Anfragen vom Frontend an den Dev-Server gesendet werden und dieser sie dann an die entsprechenden Systeme weiterleitet. Alles, was Sie hierfür tun müssen, ist, folgenden Eintrag in Ihre *package.json*-Datei einzufügen:

```
"proxy": "http://google.de"
```

Listing 2.16 Proxykonfiguration in der »package.json«-Datei

Nach dieser Anpassung müssen Sie einmal Ihre Applikation neu starten, damit die Änderungen wirksam werden. Sobald Sie nun eine Anfrage ans Backend starten, leitet der Dev-Server sie entsprechend weiter. Damit der Proxy funktionieren kann, müssen die Anfragen allerdings an denselben Host gerichtet werden, der auch die Applikation ausliefert. Im Produktivbetrieb ist dies in der Regel auch so gegeben, sodass es keine weiteren Probleme geben sollte. Mehr zum Thema Serverkommunikation erfahren Sie in Kapitel 3, »Die Grundlagen von React«.

2.4.6 Verschlüsselte Kommunikation während der Entwicklung

Im Normalfall wird Ihre Applikation vom Webpack-Dev-Server über HTTP ausgeliefert, also nicht verschlüsselt. In den meisten Fällen stellt das auch kein Problem dar, da die Browserschnittstellen, die HTTPS erfordern, eine Ausnahme bei der Auslieferung von *localhost* machen. Relevant wird die Verwendung von HTTPS vor allem, wenn Sie das bereits vorgestellte Proxy-Feature verwenden und an ein HTTPS-Backend weiterleiten möchten. In diesem Fall setzen Sie die HTTPS-Umgebungsvariable, bevor Sie den Dev-Server starten. Der Server nutzt dann ein selbst signiertes Zertifikat, um die Verbindung zu verschlüsseln. Wie das auf einem Unix-System funktioniert, sehen Sie in Listing 2.17:

```
HTTPS=true npm start
```

Listing 2.17 Starten der Applikation mit HTTPS-Unterstützung

Beachten Sie, dass Sie dann Ihre Applikation über die URL *https://localhost:3000* erreichen, also das Protokoll anpassen müssen. Nachdem Sie Ihre Applikation nun gestartet und einen ersten Eindruck von der Entwicklungsumgebung bekommen haben, stelle ich Ihnen im nächsten Schritt den Aufbau der Applikation vor.

2.5 Die Struktur der Applikation

Create React App bereitet Ihnen die Applikation so weit vor, dass Sie sofort mit der Entwicklung beginnen können. Damit Sie sich in der erzeugten Struktur besser zurechtfinden, erhalten Sie in diesem Abschnitt eine kurze Erklärung zu den verschiedenen Dateien und Verzeichnissen.

Im Basisverzeichnis Ihrer Applikation liegt eine Reihe von Dateien. Sie enthalten globale Einstellungen und Konfigurationen für die Applikation:

- *.gitignore*: Diese Datei listet die Dateien und Verzeichnisse auf, die nicht ins Git-Repository eingecheckt werden sollen. Hierunter fällt beispielsweise das *node_modules*-Verzeichnis.
- *package.json*: In der *package.json* finden Sie die Konfiguration der Applikation, beispielsweise die Start-Scripts und die installierten Abhängigkeiten.
- *README.md*: Die Readme-Datei enthält normalerweise die Dokumentation der Applikation. In der initialen Version dieser Datei finden Sie die Beschreibung der verschiedenen Start-Scripts und einen Verweis zur Onlinedokumentation. In Ihrer Applikation sollten Sie versuchen, in dieser Datei alles zu dokumentieren, was Entwickler*innen benötigen, um an der Entwicklung der Applikation zu arbeiten. Dazu gehören das Setup der Entwicklungsumgebung, der Build- und Releaseprozess und Besonderheiten der Applikation. Das Ziel sollte sein, dass neue Kolleg*innen nur die Readme durchlesen müssen und danach produktiv mitarbeiten können.
- *package-lock.json*: In der *package-lock.json*-Datei werden die exakten Versionen und die Integritäts-Hashes der installierten Abhängigkeiten und deren Unterabhängigkeiten festgehalten. Dies stellt sicher, dass die Installationen Ihrer Applikation auf allen Systemen zu jedem Zeitpunkt gleich sind. Außerdem stellt der Paketmanager durch die Prüfsummen der Pakete sicher, dass keine manipulierten Pakete eingeschleust werden können.

Neben den Dateien befinden sich im Wurzelverzeichnis der Applikation auch einige Verzeichnisse:

- *node_modules*: In diesem Verzeichnis werden alle installierten NPM-Pakete in einer flachen Hierarchie vorgehalten. Das *node_modules*-Verzeichnis ist dem Paketmanager vorbehalten, sodass Sie die Inhalte nicht von Hand manipulieren sollten, damit diese Änderungen nicht an andere Entwickler verteilt werden und so verloren gehen.
- *public*: Das *public*-Verzeichnis enthält die Einstiegsdatei in Ihre Applikation: die *index.html*. Diese wird bei einer Anfrage eines Clients ausgeliefert und sorgt dafür, dass ein Grundgerüst zur Verfügung steht. In diesem Verzeichnis können Sie statische *Assets* wie HTML, CSS, JavaScript und Mediendateien ablegen. In den meis-

ten Fällen ist es jedoch empfehlenswert, diese Daten dynamisch über das Modulsystem einzubinden, da Webpack so die Möglichkeit hat, die Inhalte zu optimieren.

Die Platzierung von Assets im *public*-Verzeichnis ist empfehlenswert, wenn die Namen der Ressourcen durch den Buildprozess nicht verändert werden dürfen, wie es beispielsweise mit der *manifest.json*-Datei der Fall ist, da der Browser die Datei mit exakt diesem Namen erwartet. Außerdem kann es sinnvoll sein, Dateien hier abzulegen, die nicht Bestandteil Ihres Applikationspakets sein sollen.

Direkt aus der *index.html*-Datei können Sie nur auf Inhalte des *public*-Verzeichnisses referenzieren. Um auf Inhalte des *public*-Verzeichnisses zu verweisen, nutzen Sie die `PUBLIC_URL`-Variable. Innerhalb der *index.html*-Datei wird diese von Prozentzeichen eingefasst, wie Sie in Listing 2.18 am Beispiel des Favicons sehen:

```
<link rel="icon" href="%PUBLIC_URL%/favicon.ico">
```

Listing 2.18 Einbinden des Favicons in die »index.html«-Datei

Um aus Ihrer Applikation heraus (beispielsweise aus einer Komponente) auf die Inhalte dieses Verzeichnisses zu verweisen, nutzen Sie ebenfalls die `PUBLIC_URL`-Variable. Dieser stellen Sie beim Zugriff die Objektstruktur `process.env` voran, wie Sie in Listing 2.19 sehen:

```
<img src={process.env.PUBLIC_URL + /cat.jpeg'} />
```

Listing 2.19 Verweis auf Inhalte des »public«-Verzeichnisses aus der Applikation

- *src*: Im *src*-Verzeichnis werden Sie sich die meiste Zeit während der Entwicklung aufhalten. Hier speichern Sie die Dateien ab, die die Komponenten und sämtliche Hilfskonstrukte für Ihre Applikation enthalten.

  Create React App gibt hier eine leichtgewichtige Struktur vor, die aus einer Reihe von Dateien besteht. Die *index.js*-Datei bildet den Einstieg in die Datei und rendert die Wurzelkomponente, die sich wiederum in der Datei *App.js* befindet. *App.css* enthält Styledefinitionen, die in der *App.js* geladen werden. Mit *App.test.js* haben Sie auch bereits einen ersten Unittest für Ihre Applikation, den Sie mit dem Kommando `npm test` ausführen können. In der *index.css*-Datei finden Sie allgemeine Styledefinitionen, beispielsweise für das `body`-Element Ihrer Applikation. Die Datei *logo.svg* stellt schließlich ein statisches Asset dar, auf das aus dem JavaScript-Code Ihrer Applikation verwiesen wird.

2.6 Fehlersuche in einer React-Applikation

Ein in der Entwicklung häufig unterschätztes Hilfsmittel ist der Webbrowser. Ihn können Sie nicht nur zur Anzeige Ihrer Applikation verwenden, sondern auch aktiv

in den Entwicklungsprozess einbinden. Moderne Webbrowser verfügen über eine Reihe hilfreicher Werkzeuge zur Analyse und Fehlersuche in einer Applikation. Allen voran ist der Debugger zu nennen. Mit der Tastenkombination Cmd+Alt+i auf dem Mac beziehungsweise Strg+Alt+i oder F12 auf einem Windows- oder Linux-System öffnen Sie die Entwicklerwerkzeuge des Browsers. In Abbildung 2.8 sehen Sie die Ansicht eines Browsers, der an einem Breakpoint in der Applikation angehalten hat.

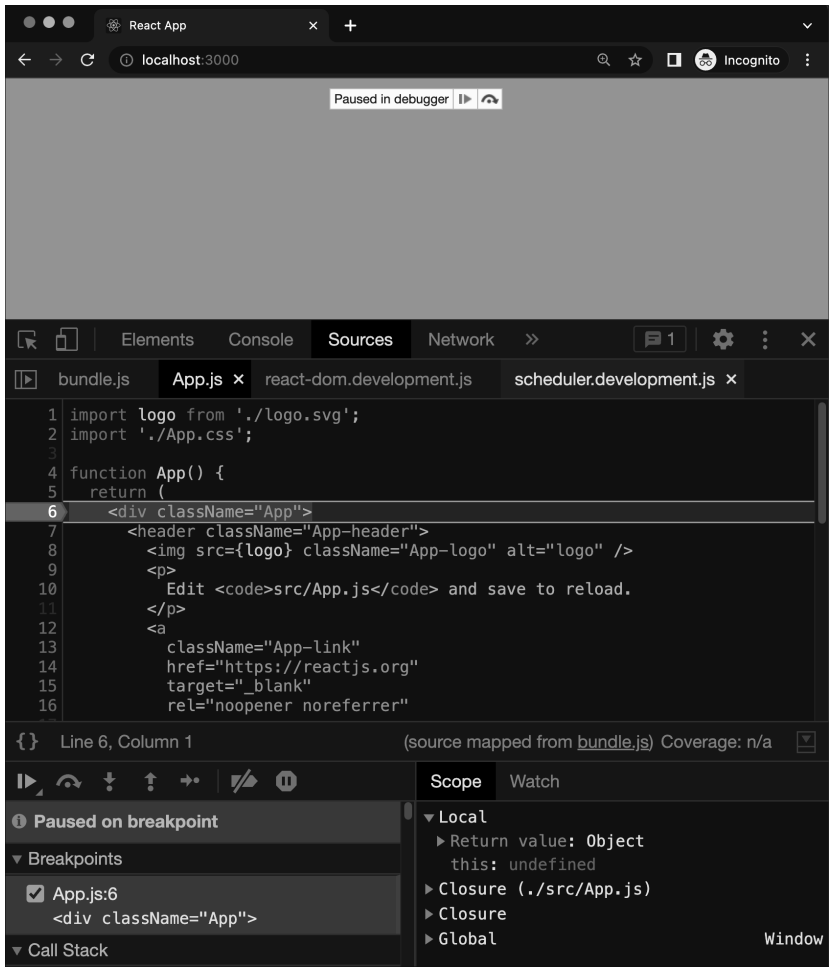


Abbildung 2.8 Angehaltener Debugger in einer React-Applikation

Um Ihre Applikation zu debuggen, öffnen Sie die Entwicklerwerkzeuge, wechseln auf den SOURCES-Tab und setzen einen Breakpoint, indem Sie auf die Zeilennummer klicken. Ein Rechtsklick auf die Zeilennummer bietet Ihnen außerdem die Möglichkeit, bedingte Breakpoints zu setzen, bei denen der Browser nur anhält, wenn eine zuvor definierte Bedingung erfüllt ist. Eine weitere Möglichkeit, einen Breakpoint zu

setzen, ist die Verwendung des `debugger`-Statements. Hierfür schreiben Sie im Quellcode Ihrer Applikation an der gewünschten Stelle das Schlüsselwort `debugger`. Sind die Entwicklerwerkzeuge Ihres Browsers geöffnet, wird die Ausführung an dieser Stelle angehalten.

Über die Symbolleiste im rechten oberen Teil der Entwicklerwerkzeuge können Sie im Debugger navigieren. Die Symbole bedeuten im Einzelnen:

- **Resume**: Pausiert der Debugger an einem Breakpoint, kann der Ablauf des Scripts mit dieser Schaltfläche fortgesetzt werden.
- **Step over**: Überspringt den nächsten Funktionsaufruf.
- **Step into**: Springt in die aufzurufende Funktion hinein. Diese Aktion fügt einen Eintrag zum Callstack hinzu.
- **Step out**: Springt aus der aktuellen Funktion heraus. Mit dieser Aktion wird der aktuelle Eintrag vom Callstack entfernt.
- **Step**: Diese Aktion springt in die nächste Zeile.
- **Deactivate Breakpoints**: Deaktiviert alle aktuell gesetzten Breakpoints, sodass die Ausführung ohne Unterbrechung fortgesetzt wird.
- **Pause on Exception**: Mit dieser Schaltfläche sorgen Sie dafür, dass die Ausführung pausiert wird, sobald eine Exception geworfen wird.

Ist die Ausführung Ihrer Applikation an einem Breakpoint pausiert, erhalten Sie zusätzliche Informationen über den aktuellen Zustand Ihrer Applikation. So können Sie sich die aktuell gültigen Variablenbelegungen oder den Callstack ansehen oder Watch-Expressions definieren. Das sind Ausdrücke, die bei jedem Schritt im Debugger erneut ausgewertet werden. Das Ergebnis wird Ihnen in der entsprechenden Sektion des Debuggers angezeigt.

Neben der Verwendung des Browser-Debuggers gibt es außerdem die Möglichkeit, Ihre Applikation direkt in Ihrer Entwicklungsumgebung zu debuggen. Das hat den Vorteil, dass Sie direkt in Ihrem Quellcode arbeiten und dort auch Breakpoints setzen können. Wie Sie den Debugger für Ihre Entwicklungsumgebung einrichten, entnehmen Sie bitte der jeweiligen Dokumentation. Im Fall von Visual Studio Code ist ein Debugger integriert. Die zugehörige Dokumentation finden Sie unter *https://code.visualstudio.com/docs/editor/debugging*. Auch die Dokumentation von WebStorm enthält hierzu einen eigenen Abschnitt unter *www.jetbrains.com/help/webstorm/debugging-javascript-in-chrome.html*.

2.6.1 Arbeiten mit den React Developer Tools

Der Debugger ist allerdings nicht das einzige Hilfsmittel, auf das Sie bei der Entwicklung einer React-Applikation zugreifen können. Sowohl für Firefox als auch für

Chrome existiert eine Browsererweiterung, die Ihnen die Struktur und die dynamischen Daten Ihrer Applikation anzeigen kann. Für Chrome finden Sie die *React Developer Tools* im Chrome Web Store unter der URL *https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi*. Für Firefox ist das Werkzeug unter *https://addons.mozilla.org/en-US/firefox/addon/react-devtools/* verfügbar.

Nach der Installation finden Sie in den Entwicklerwerkzeugen Ihres Browsers zwei Tabs mit den Bezeichnungen Profiler und Components. Mit dem Profiler können Sie Leistungsdaten zur Laufzeit Ihrer Applikation aufnehmen und anschließend auswerten. Im Components-Tab sehen Sie die aktuelle Komponentenstruktur Ihrer Applikation und können die einzelnen Komponenten inspizieren sowie deren Props und State überprüfen. Wie die React Dev Tools im Browser aussehen, zeigt Abbildung 2.9.

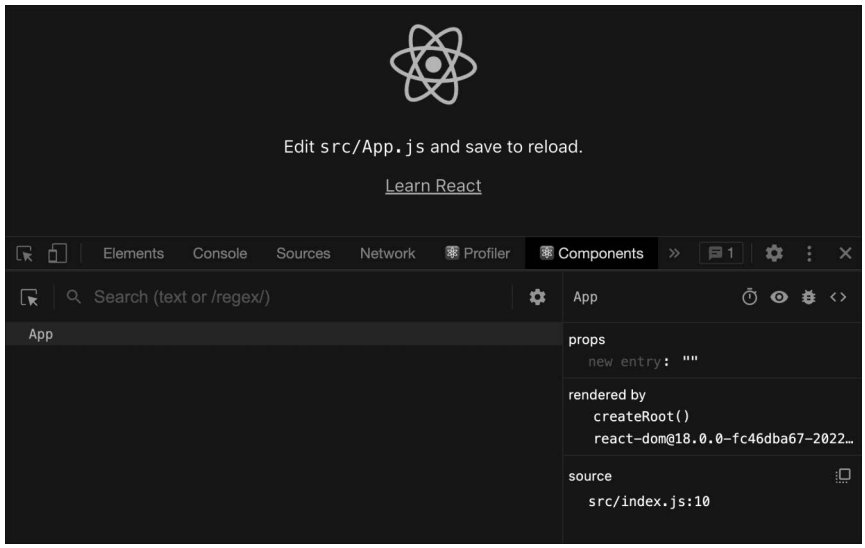


Abbildung 2.9 Die »React Dev Tools« in Chrome

Damit kommen wir zum letzten Schritt im Entwicklungsprozess: dem Bauen der Applikation für den Produktivbetrieb.

2.7 Die Applikation bauen

Im Entwicklungsbetrieb wird der Quellcode zwar durch den Webpack-Dev-Server übersetzt, sodass der Browser ihn problemlos ausführen kann. Der Server optimiert den Code jedoch nicht. Um Ihre Applikation für den Produktivbetrieb vorzubereiten, führen Sie den Befehl `npm run build` aus.

Als Ergebnis wird ein neues Verzeichnis mit dem Namen *build* erstellt. In diesem Verzeichnis befinden sich sämtliche Ressourcen, die Sie benötigen, um die Applikation an Ihre Benutzer*innen auszuliefern. Den Inhalt dieses Verzeichnisses können Sie in das Document-Root-Verzeichnis eines Webservers kopieren und so Ihre Applikation deployen. Dieses Verhalten können Sie anpassen, indem Sie in Ihre *package.json*-Datei einen Eintrag mit dem Schlüssel `homepage` und der URL Ihrer Applikation als Wert einfügen.

Um zu testen, ob der Build funktioniert hat, können Sie auch einen lokalen Webserver installieren und Ihre Applikation über ihn ausliefern. Eine der einfachsten Lösungen hierfür bietet das NPM-Paket `http-server`. Dieses führen Sie mit dem Kommando `npx http-server build` im Wurzelverzeichnis Ihrer Applikation aus.

2.8 Zusammenfassung

Dieses Kapitel diente dazu, Ihnen den Entwicklungsprozess mit React näherzubringen und Ihnen die Werkzeuge vorzustellen, die Ihnen hierbei zur Verfügung stehen.

- Sie können React sowohl in bestehende Webseiten einbinden als auch eine React-Applikation von Grund auf neu implementieren.
- Mit Playgrounds wie *CodePen* können Sie einfache Experimente mit React durchführen, indem Sie die Bibliothek direkt einbinden. Diese Variante eignet sich allerdings nicht für umfangreiche Projekte.
- React lässt sich sehr leichtgewichtig durch direkte Einbindung der Bibliotheksdateien in eine statische HTML-Seite integrieren. Hierbei verzichten Sie jedoch auf den Komfort einer vorgefertigten Struktur und konfigurierter Werkzeuge.
- *Create React App* ist das Werkzeug der Wahl, wenn es um das Setup größerer Anwendungen geht. Auf der generierten Struktur können Sie auch umfangreiche Applikationen umsetzen.
- Das Paket `react-scripts`, das mit Create React App installiert wird, bietet Ihnen vier Scripts, die Ihnen das Starten, Testen, Bauen sowie das Exportieren der Webpack-Konfiguration ermöglichen.
- Das Verhalten von Create React App sowie von `react-scripts` können Sie über Kommandozeilenoptionen sowie Umgebungsvariablen beeinflussen.
- Alle modernen Browser verfügen über leistungsstarke Entwicklerwerkzeuge, die Ihnen bei der Fehlersuche helfen. Außerdem existieren mit den *React Developer Tools* Erweiterungen, die Ihnen bei der Analyse einer React-Applikation helfen.

Im nächsten Kapitel lernen Sie die Grundbausteine von React kennen und beginnen mit der Entwicklung Ihrer Applikation.

Kapitel 10
Formulare in React

Wenn du mit mir konversieren möchtest, definiere zuerst deine Termini.
– Voltaire

Bisher hat sich die Interaktion von Benutzern und Benutzerinnen mit Ihrer Applikation auf einfache Klick-Events beschränkt, die sich auf bestimmte Elemente beziehen. In den meisten Fällen sind diese eingeschränkten Möglichkeiten jedoch nicht ausreichend. Gerade wenn es um die Erzeugung oder die Modifikation von Datensätzen geht, müssen Sie auf vollwertige Formulare zurückgreifen. In einer React-Applikation können Sie sämtliche gültigen HTML-Formularelemente integrieren und sie für Ihre Zwecke nutzen.

Im Zuge dieses Kapitels lernen Sie mit den *Uncontrolled* und *Controlled Components* verschiedene Arten der Formularbehandlung kennen. Außerdem erfahren Sie am konkreten Beispiel von *React Hook Form*, wie Sie externe Bibliotheken zur Arbeit mit Formularen einsetzen können.

10.1 Uncontrolled Components

Schnellstart

Eine Uncontrolled Component ist nicht mit dem State einer Komponente synchronisiert. Sie greifen über eine *Ref* auf den Wert eines solchen Formularelements zu:

```
import React, { useRef } from 'react';
import './App.css';

const App: React.FC = () => {
  const inputRef = useRef<HTMLInputElement>(null);
  function handleClick() {
    console.log(inputRef.current!.value);
  }
  return (
    <div>
```

```
      <input type="text" ref={inputRef} />
      <button onClick={handleClick}>ok</button>
    </div>
  );
};

export default App;
```

Listing 10.1 Beispiel für eine Uncontrolled Component (»src/App.tsx«)

Für die Arbeit mit einer Uncontrolled Component erzeugen Sie eine Ref und verknüpfen diese mit der `ref`-Prop des Formularelements. Anschließend können Sie auf den Wert des Formularelements über die `current.value`-Eigenschaft der Ref zugreifen.

Die einfachste Art, mit Formularen umzugehen, sind *Uncontrolled Components*. Ihr Name rührt von der Tatsache her, dass React keine Kontrolle über diese Formularelemente ausübt, sondern sie lediglich über Referenzen anspricht.

In der Regel sollten Sie bei der Implementierung Ihrer Applikation auf Controlled Components setzen, da diese dafür sorgen, dass die Variablen in Ihrer Komponente und die Werte der Eingabeelemente synchronisiert werden. Sie können dann jederzeit auf die Variablen zugreifen und erhalten immer den aktuellen Wert. Anders ist die Situation bei den Uncontrolled Components: Bei ihnen greifen Sie – wie in alten jQuery-Applikationen – über eine Referenz auf den Wert zu.

10.1.1 Der Umgang mit Referenzen in React

React verfolgt bei der Implementierung der Oberfläche einer Applikation einen deklarativen Ansatz. Sie beschreiben also mehr, was Sie erreichen möchten, und weniger, wie Sie es erreichen möchten. Das ist auch ein Grund, warum es als Antipattern angesehen wird, wenn Sie innerhalb Ihrer Applikation Elemente explizit selektieren und Operationen darauf anwenden. Aber genau dieses Muster ist die Grundlage für Uncontrolled Components. Mit den *Referenzen* in React (oder kurz *Refs*) haben Sie eine Möglichkeit, ohne `querySelector` auf die DOM-Elemente Ihrer Applikation zuzugreifen.

Den Kern der Refs in React bildete in der Vergangenheit die Methode `createRef`. Sie wurde mit der Einführung der Hooks-API de facto durch den *Ref-Hook* abgelöst. Die `useRef`-Funktion erzeugt eine ungebundene Referenz, der Sie ein konkretes Element zuweisen können. Dies geschieht über das `ref`-Attribut eines Elements. Zur Demonstration des Umgangs mit Refs sowie mit Uncontrolled Components in React implementieren wir im Folgenden ein einfaches Anmeldeformular. Da sich die Beispiele bisher immer in einem übersichtlichen Umfang bewegen, reicht es aus, die Dateien

direkt im *src*-Verzeichnis der Applikation abzulegen. Setzen Sie eine umfangreichere Applikation um, stoßen Sie mit diesem Ansatz allerdings an Grenzen, da die Übersicht bei einer großen Anzahl von Dateien schnell verloren geht.

Organisation des Quellcodes

Je nachdem, wie umfangreich Ihre Applikation ist, bieten sich verschiedene Formen der Strukturierung des Quellcodes an. Das bedeutet jedoch nicht, dass Sie zu Beginn der Arbeit bereits wissen müssen, wie umfangreich Ihr Quellcode werden wird. Sie können zunächst mit einem leichtgewichtigen Ansatz starten und später zusätzliche Strukturen umsetzen. Mit dieser Vorgehensweise entwickelt sich die Struktur zusammen mit dem Funktionsumfang weiter.

Leichtgewichtiger Ansatz

Der aktuelle Stand der Beispielapplikation stellt die leichtgewichtigste Variante der Strukturierung dar. Hierbei befinden sich sämtliche Dateien im *src*-Verzeichnis der Applikation. Die Unterscheidung der verschiedenen Dateitypen geschieht in diesem Fall anhand der Dateinamen. Hierbei sollten Sie darauf achten, dass auf den ersten Blick ersichtlich wird, worum es sich bei einer Datei handelt. Ihre Komponenten können Sie beispielsweise mit der Endung *.component.tsx* versehen, um deutlich zu machen, dass die jeweilige Datei eine Komponente enthält.

Gruppierung nach der Art der Elemente

Ein Ansatz, der etwas mehr Struktur bietet, sich aber auch nur bedingt für große Applikationen eignet, ist die Gruppierung der Dateien anhand ihres Typs. Hier erzeugen Sie Verzeichnisse, die die unterschiedlichen Typen repräsentieren. So könnten sich für die Beispielapplikation die Verzeichnisse *components*, *hooks*, *models* und *util* ergeben. Das *components*-Verzeichnis enthält die einzelnen Komponentendateien. Bei einer großen Anzahl von Komponenten können Sie hier wieder Unterverzeichnisse pro Komponente erzeugen, die dann die Komponente selbst, den zugehörigen Test und das Stylesheet enthalten. Das *hooks*-Verzeichnis beinhaltet die Hooks der Applikation. Im *models*-Verzeichnis liegen die Datenklassen, und das *utils*-Verzeichnis enthält schließlich Dateien mit Hilfsfunktionen.

Fachliche Gruppierung

Bei der fachlichen Gruppierung bildet die Verzeichnisstruktur die einzelnen fachlichen Module der Applikation ab. Haben Sie in Ihrer Applikation beispielsweise ein Modul, das sich um die Benutzerverwaltung kümmert, bietet sich hierfür ein Verzeichnis mit dem Namen *user* an. Alle Komponenten, die mit der Anmeldung der Benutzer*innen zu tun haben, können Sie in einem Verzeichnis mit dem Namen *Login* organisieren.

Der Vorteil der fachlichen Gruppierung ist, dass diese Variante auch für umfangreiche Applikationen verwendet werden kann und dafür sorgt, dass die Verzeichnisstruktur auch bei einer großen Anzahl von Komponenten noch aufgeräumt bleibt.

In Listing 10.2 sehen Sie die Implementierung des Formulars als Funktionskomponente:

```
import React, { useRef } from 'react';

const Login: React.FC = () => {
  const usernameRef = useRef<HTMLInputElement>(null);
  const passwordRef = useRef<HTMLInputElement>(null);

  return (
    <form>
      <div>
        <label>
          Benutzername:
          <input type="text" ref={usernameRef} />
        </label>
      </div>
      <div>
        <label>
          Passwort:
          <input type="password" ref={passwordRef} />
        </label>
      </div>
      <button type="submit">anmelden</button>
    </form>
  );
};

export default Login;
```

Listing 10.2 Basisformular mit Uncontrolled Components (»src/ Login.tsx«)

Die Basis des Anmeldeformulars bildet die JSX-Struktur in der Komponentenfunktion. Hierbei handelt es sich um ein einfaches Formular mit einem Textfeld für den Benutzernamen und einem Passwortfeld. Beide Elemente werden mithilfe eines Labels mit einer Beschriftung versehen. Zum Absenden des Formulars verwenden Sie einen `submit`-Button. Der einzige React-spezifische Teil dieses Formulars sind die `ref`-Attribute der beiden Eingabefelder. Mit ihrer Hilfe können Sie von den Methoden der Komponente aus auf die Formularfelder zugreifen. Beide Referenzen erzeugen Sie mithilfe der `useRef`-Funktion von React.

Nutzen Sie, wie im Beispiel, TypeScript, ist `useRef` als eine generische Funktion umgesetzt, bei der Sie zusätzlich angeben, auf welche Art von Element die Referenz verweist. Im Fall des Beispiels ist dies der Typ `HTMLInputElement`. Mit diesem Stand kön-

nen Sie die `Login`-Komponente bereits testweise in die `App`-Komponente integrieren. In Listing 10.3 sehen Sie den Quellcode der `App`-Komponente:

```
import React from 'react';
import './App.css';
import Login from './Login';

const App: React.FC = () => {
  return <Login />;
};

export default App;
```

Listing 10.3 Einbindung der »Login«-Komponente (»src/App.tsx«)

Autofocus eines Eingabefelds mit Refs

Mit Refs lassen sich Features umsetzen, beispielsweise ein Autofocus eines Feldes. Zwar gibt es für diesen Zweck das `autofocus`-Attribut eines HTML-Elements. Dieses ist jedoch durch den dynamischen Charakter von React nicht immer zuverlässig. Eine bessere Alternative ist hier die Verwendung von Refs. Zu diesem Zweck definieren Sie einen Effect-Hook, der nur beim Mounten der Komponente aktiv wird, und führen dort die `focus`-Methode des `username`-Elements aus. Diese steht ihnen innerhalb der `current`-Eigenschaft des `Ref`-Objekts zur Verfügung. Listing 10.4 enthält die Anpassung an der `Login`-Komponente:

```
import React, { useEffect, useRef } from 'react';

const Login: React.FC = () => {
  const usernameRef = useRef<HTMLInputElement>(null);
  const passwordRef = useRef<HTMLInputElement>(null);

  useEffect(() => {
    usernameRef.current!.focus();
  }, []);

  return (
    <form>
      <div>
        <label>
          Benutzername:
          <input type="text" ref={usernameRef} />
        </label>
      </div>
```

```
      <div>
        <label>
          Passwort:
          <input type="password" ref={passwordRef} />
        </label>
      </div>
      <button type="submit">anmelden</button>
    </form>
  );
};

export default Login;
```

Listing 10.4 Autofocus mit Refs (»src/login/Login.tsx«)

Mit dem ! nach der `current`-Eigenschaft teilen Sie dem TypeScript-Compiler mit, dass die `current`-Eigenschaft auf jeden Fall ein Objekt ist, obwohl die Möglichkeit besteht, dass es theoretisch `null` sein könnte. Wenn Sie Ihre Applikation mit diesen Änderungen im Browser öffnen, ist das Eingabefeld für den Benutzernamen standardmäßig aktiviert und Sie können dort direkt mit der Eingabe beginnen.

Um das `Login`-Formular zu komplettieren, müssen Sie noch eine Routine implementieren, die das Formular absendet.

Formular absenden

Bisher haben Sie lediglich mit Klick-Events gearbeitet. React bietet im Zusammenhang mit Formularen noch einige weitere Event-Handler. Im Fall des Login-Formulars müssen Sie beispielsweise auf das `submit`-Event reagieren, das Sie entweder durch Betätigen der ↵-Taste oder durch einen Klick auf den `submit`-Button auslösen können. Innerhalb der `handleSubmit`-Methode implementieren Sie die Logik, um entsprechend auf das `submit`-Event zu reagieren:

```
import React, { FormEvent, useEffect, useRef } from 'react';

type Props = {
  onLogin: (username: string, password: string) => void;
};

const Login: React.FC<Props> = ({ onLogin }) => {
  const usernameRef = useRef<HTMLInputElement>(null);
  const passwordRef = useRef<HTMLInputElement>(null);

  useEffect(() => {
```

```
    usernameRef.current!.focus();
  }, []);

  function handleSubmit(event: FormEvent<HTMLFormElement>) {
    event.preventDefault();
    onLogin(usernameRef.current!.value, passwordRef.current!.value);
  }

  return (
    <form onSubmit={handleSubmit}>
      <div>
        <label>
          Benutzername:
          <input type="text" ref={usernameRef} />
        </label>
      </div>
      <div>
        <label>
          Passwort:
          <input type="password" ref={passwordRef} />
        </label>
      </div>
      <button type="submit">anmelden</button>
    </form>
  );
};

export default Login;
```

Listing 10.5 Behandlung des »submit«-Events in der »Login«-Komponente (»src/Login.tsx«)

Wie Sie in Listing 10.5 sehen, wird der `handleSubmit`-Funktion automatisch die Objektrepräsentation des `submit`-Events übergeben. Dieses Objekt ist vom Typ `FormEvent`. Das Standardverhalten eines HTML-Formulars ist, dass beim Abschicken des Formulars die Seite neu geladen wird. Dies können Sie mit der `preventDefault`-Methode des Event-Objekts verhindern. Auf die einzelnen Werte der Formularfelder greifen Sie über die beiden `Ref`-Objekte zu. Das Ausrufezeichen nach `current` besagt, dass dieser Wert nicht `null` ist, sodass es zu keinem TypeScript-Fehler kommt. Weil eine Komponente für nur einen bestimmten Aspekt verantwortlich ist, kümmert sich die `Login`-Komponente um die Darstellung des Anmeldeformulars und das zugehörige Event-Handling, nicht aber um das Senden der Anmeldedaten zum Server. Diese Aufgabe übernimmt die Funktion `onLogin`, die über die Props in die Komponente hineingereicht wird.

Wenn Sie jetzt noch in der `App`-Komponente die `onLogin`-Prop korrekt übergeben, können Sie Ihre Komponente auch schon testen. In einer minimalen Implementierung reicht hier eine Callback-Funktion aus, die den Benutzernamen und das Passwort akzeptiert und beide Informationen auf der Konsole ausgibt. Womit wir auch schon beim nächsten Thema sind. An dieser Stelle bietet es sich an, dass Sie auch einen Unittest für Ihre Komponente verfassen. In diesem Test prüfen Sie, ob nach der Eingabe des Benutzernamens und des Passworts das Absenden des Formulars funktioniert und die `onLogin`-Funktion korrekt aufgerufen wird. Für diesen Test legen Sie eine neue Datei *Login.spec.tsx* im *src*-Verzeichnis an. Der Test kann dann so wie in Listing 10.6 aussehen:

```
import { fireEvent, render, screen } from '@testing-library/react';
import Login from './Login';

describe('Login', () => {
  it('should call onLogin correctly after submitting the form', () => {
    const onLogin = jest.fn();
    render(<Login onLogin={onLogin} />);

    const username = screen.getByTestId('username');
    const password = screen.getByTestId('password');
    const submit = screen.getByTestId('submit');

    fireEvent.change(username, { target: { value: 'testuser' } });
    fireEvent.change(password, { target: { value: 'testpassword' } });
    fireEvent.click(submit);

    expect(onLogin).toHaveBeenCalledWith('testuser', 'testpassword');
  });
});
```

Listing 10.6 Test der »Login«-Komponente (»src/Login.spec.tsx«)

Damit Ihr Test ordnungsgemäß ausgeführt werden kann, müssen Sie in der `Login`-Komponente bei den beiden Input-Elementen und dem Submit-Button die `data-testid`-Attribute einfügen. In Listing 10.7 sehen Sie den angepassten JSX-Code der `Login`-Komponente:

```
<form onSubmit={handleSubmit}>
  <div>
    <label>
      Benutzername:
      <input type="text" ref={usernameRef} data-testid="username" />
    </label>
```

```
  </div>
  <div>
    <label>
      Passwort:
      <input type="password" ref={passwordRef} data-testid="password" />
    </label>
  </div>
  <button type="submit" data-testid="submit">
    anmelden
  </button>
</form>
```

Listing 10.7 Anpassungen an der JSX-Struktur der »Login«-Komponente (»src/Login.tsx«)

Im Test erzeugen Sie eine Spy-Funktion für die `onLogin`-Prop, mit der Sie am Ende des Tests prüfen können, ob das Formular ordnungsgemäß funktioniert. Dann rendern Sie das Formular und übergeben dabei die Spy-Funktion über die `onLogin`-Prop. Im nächsten Schritt nutzen Sie die `getByTestId`-Methode, um Zugriff auf die einzelnen Elemente zu erhalten, setzen mit `fireEvent.change` die Werte für den Benutzernamen und das Passwort und lassen den Test schließlich auf den Submit-Button klicken. Danach prüfen Sie mit dem `toHaveBeenCalledWith`-Matcher von Jest, ob die Spy-Funktion mit der richtigen Benutzernamen-Passwort-Kombination aufgerufen wurde.

Damit führt Jest zwar den Test erfolgreich aus, Ihre aktuelle Implementierung deckt jedoch nur den Erfolgsfall ab. Tritt ein Fehler auf, erfahren die Benutzer*innen nicht direkt etwas davon.

Fehlerbehandlung in Uncontrolled Components

Bei solchen Formularimplementierungen müssen Sie jedoch nicht auf eine Fehlerbehandlung verzichten. Diese kann, je nach Anwendungsfall, schon während der Eingabe, vor dem Absenden des Formulars oder erst serverseitig erfolgen. Für welche Variante Sie sich entscheiden, hängt jeweils von den Anforderungen Ihrer Applikation ab. Je früher Sie die Eingabe prüfen, desto schneller erhält Ihr Benutzer eine Rückmeldung. In einigen Fällen ist die clientseitige Prüfung jedoch auch nicht möglich, da der Client nicht über die erforderlichen Informationen verfügt; dann muss die Prüfung serverseitig erfolgen, und die Rückmeldung verzögert sich etwas.

Bei der ersten Variante implementieren Sie einen Event-Handler für das `change`-Event eines Formularfeldes und überprüfen bei jeder Änderung, ob es sich um einen gültigen Wert handelt. Ist dies nicht der Fall, können Sie den Benutzern und Benutzerinnen ein direktes Feedback geben. Für eine solche Umsetzung eignen sich jedoch Controlled Components erheblich besser, da diese direkt mit der Komponentenstruktur verbunden sind. Doch dazu später mehr.

Führen Sie die Überprüfung erst beim Absenden des Formulars durch, erfolgt das Feedback an die Benutzer und Benutzerinnen nicht mehr unmittelbar bei der Eingabe, aber noch vor einer potenziell zeitintensiven Serveranfrage. In diesem Fall setzen Sie die Prüfung im `submit`-Handler um, und abhängig vom Ergebnis zeigen Sie entweder eine Fehlermeldung an oder übermitteln die Daten an den Server.

Bei der dritten Variante erfolgt die Überprüfung serverseitig. Dies ist bei einer Anmeldung erforderlich, da die Clientseite nicht weiß, ob es sich bei den angegebenen Informationen um gültige Anmeldeinformationen handelt.

Achtung!

Zusätzlich zur clientseitigen Validierung sollten Sie Benutzereingaben in jedem Fall auch serverseitig prüfen, da eine clientseitige Überprüfung von potenziellen Angreifer*innen umgangen werden kann.

Im Beispiel prüfen Sie nun, ob sowohl ein Benutzername als auch ein Passwort angegeben wurde. Außerdem müssen Sie mit einem Fehlschlag der Anmeldung umgehen können. Zur Visualisierung von Fehlern setzen Sie in der `Login`-Komponente einen `error`-State um. Dieser weist im Erfolgsfall eine leere Zeichenkette auf; erst wenn ein Fehler auftritt, wird die entsprechende Fehlermeldung gesetzt. Zusätzlich dazu kann die Elternkomponente eine `error`-Prop setzen, um anzuzeigen, dass die Anmeldung fehlgeschlagen ist. Die angepasste Version der `Login`-Komponente sehen Sie in Listing 10.8:

```
import React,{ FormEvent, useEffect, useRef, useState } from 'react';

type Props = {
  onLogin: (username: string, password: string) => void;
  loginError?: string;
};

const Login: React.FC<Props> = ({ onLogin, loginError }) => {
  const [validationError, setValidationError] = useState<string>('');

  const usernameRef = useRef<HTMLInputElement>(null);
  const passwordRef = useRef<HTMLInputElement>(null);

  useEffect(() => {
    usernameRef.current!.focus();
  }, []);
```

```
  function handleSubmit(event: FormEvent<HTMLFormElement>) {
    event.preventDefault();
    let error = 'Bitte geben Sie einen Benutzernamen ↩
        und ein Passwort ein.';
    const username = usernameRef.current!.value;
    const password = passwordRef.current!.value;
    if (username && password) {
      error = '';
      onLogin(username, password);
    }
    setValidationError(error);
  }

  return (
    <form onSubmit={handleSubmit}>
      {loginError && <div data-testid="loginError">{loginError}</div>}
      {validationError && (
        <div data-testid="validationError">{validationError}</div>
      )}
      <div>
        <label>
          Benutzername:
          <input type="text" ref={usernameRef} data-testid="username" />
        </label>
      </div>
      <div>
        <label>
          Passwort:
          <input type="password" ref={passwordRef} data-testid="password" />
        </label>
      </div>
      <button type="submit" data-testid="submit">
        anmelden
      </button>
    </form>
  );
};

export default Login;
```

Listing 10.8 Anzeige von Fehlermeldungen (»src/Login.tsx«)

Für die Arbeit mit den Fehlern erweitern Sie zunächst den Props-Typ um einen optionalen `loginError`. Dann definieren Sie einen lokalen `validationError`-State innerhalb der Komponente. Die Formularvalidierung erfolgt wie angekündigt innerhalb der `handleSubmit`-Funktion. Hier definieren Sie eine Fehlermeldung und prüfen, ob ein Benutzername und ein Passwort vorliegen. Ist dies der Fall, setzen Sie die Fehlermeldung auf eine leere Zeichenkette zurück und rufen die `onLogin`-Funktion auf. Im letzten Schritt setzen Sie den `validationError`-State auf den aktuell gültigen Wert, also entweder auf eine leere Zeichenkette im Erfolgsfall oder die Fehlermeldung, wenn die Validierung fehlgeschlagen ist. Beide Fehlerfälle führen in der Komponente zur Ausgabe einer entsprechenden Meldung. Damit Sie beide Fälle mit Unittests absichern können, weisen Sie den Containern jeweils eine `data-testid`-Eigenschaft zu.

In Ihrem bestehenden Test der `Login`-Komponente können Sie zunächst sicherstellen, dass im Erfolgsfall keine der beiden Fehlermeldungen angezeigt wird. Anschließend kümmern Sie sich um die Überprüfung der Darstellung des Anmeldefehlers. Dazu rendern Sie die Komponente im Test mit der `loginError`-Prop und erwarten, dass die Fehlermeldung angezeigt wird. Der Test für den Validierungsfehler ist etwas aufwendiger. Für diesen Test füllen Sie lediglich das Eingabefeld für den Benutzernamen aus und lassen das Passwort leer. Nach dem Absenden des Formulars prüfen Sie, ob die Validierungsfehlermeldung angezeigt wird. Den Quellcode der drei Tests für die `Login`-Komponente finden Sie in Listing 10.9:

```
import { fireEvent, render, screen } from '@testing-library/react';
import Login from './Login';

describe('Login', () => {
  it('should call onLogin correctly after submitting the form', () => {
    const onLogin = jest.fn();
    render(<Login onLogin={onLogin} />);

    const username = screen.getByTestId('username');
    const password = screen.getByTestId('password');
    const submit = screen.getByTestId('submit');

    fireEvent.change(username, { target: { value: 'testuser' } });
    fireEvent.change(password, { target: { value: 'testpassword' } });
    fireEvent.click(submit);

    expect(onLogin).toHaveBeenCalledWith('testuser', 'testpassword');
    expect(screen.queryByTestId('loginError')).not.toBeInTheDocument();
    expect(screen.queryByTestId('validationError')).not.toBeInTheDocument();
  });
```

```
  it('should display an external login error', () => {
    render(<Login onLogin={jest.fn()} loginError= ↩
        "Anmeldung fehlgeschlagen" />);

    const loginError = screen.getByTestId('loginError');
    expect(loginError).toBeInTheDocument();
    expect(loginError).toHaveTextContent('Anmeldung fehlgeschlagen');
  });

  it('should display an error if the validation fails', () => {
    const onLogin = jest.fn();
    render(<Login onLogin={onLogin} />);

    const username = screen.getByTestId('username');
    const password = screen.getByTestId('password');
    const submit = screen.getByTestId('submit');

    fireEvent.change(username, { target: { value: 'testuser' } });
    fireEvent.click(submit);

    const validationError = screen.queryByTestId('validationError');

    expect(onLogin).not.toHaveBeenCalled();
    expect(screen.queryByTestId('loginError')).not.toBeInTheDocument();
    expect(validationError).toBeInTheDocument();
    expect(validationError).toHaveTextContent(
      'Bitte geben Sie einen Benutzernamen und ein Passwort ein.'
    );
  });
});
```

Listing 10.9 Tests für die Fehlerfälle in der »Login«-Komponente (»src/Login.spec.tsx«)

Bei dieser Version des Anmeldeprozesses handelt es sich lediglich um einen Zwischenschritt; in Kapitel 14, in dem es um zentrales State-Management in Ihrer Applikation geht, werden wir den Anmeldeprozess noch tiefer in die Applikation integrieren.

In der Zwischenzeit sollten Sie sich jedoch noch um das Aussehen des Anmeldeformulars kümmern, damit die Benutzer und Benutzerinnen Ihrer Applikation nicht von der Ansicht abgeschreckt werden. Dazu fügen Sie in die JSX-Struktur des Formulars noch einige `className`-Props ein, wie Sie in Listing 10.10 sehen können:

```
<form onSubmit={handleSubmit} className="Login">
  {loginError && (
    <div data-testid="loginError" className="error">
      {loginError}
    </div>
  )}
  {validationError && (
    <div data-testid="validationError" className="error">
      {validationError}
    </div>
  )}
  <div>
    <label>
      Benutzername:
      <input type="text" ref={usernameRef} data-testid="username" />
    </label>
  </div>
  <div>
    <label>
      Passwort:
      <input type="password" ref={passwordRef} data-testid="password" />
    </label>
  </div>
  <button type="submit" data-testid="submit">
    anmelden
  </button>
</form>
```

Listing 10.10 Anpassungen der Struktur der »Login«-Komponente (»src/Login.tsx«)

Danach können Sie eine Datei mit dem Namen *Login.scss* im *src*-Verzeichnis anlegen, die die Styles für die Komponente in Form eines SCSS-Stylesheets enthält. Damit das Styling funktioniert, müssen Sie sicherstellen, dass das `sass`-Paket in Ihrer Applikation installiert ist. Den Quellcode des Stylesheets finden Sie in Listing 10.11:

```
.Login {
  width: 400px;
  height: 200px;
  margin: 50px auto;
  border: 1px solid black;
  box-shadow: 0 0 5px black;
  display: flex;
  flex-direction: column;
  justify-content: space-around;
```

```
  align-items: center;
  padding-left: 20px;
  label {
    width: 400px;
    position: relative;
    display: inline-block;
  }
  input {
    position: absolute;
    left: 120px;
  }
  .error {
    color: red;
  }
}
```

Listing 10.11 Styling des Login-Formulars (»src/login/Login.scss«)

Dieses Stylesheet binden Sie mit der Anweisung `import './Login.scss'` in die `Login`-Komponente ein. Mit diesen Änderungen sollte die initiale Ansicht Ihrer Applikation so wie in Abbildung 10.1 aussehen.

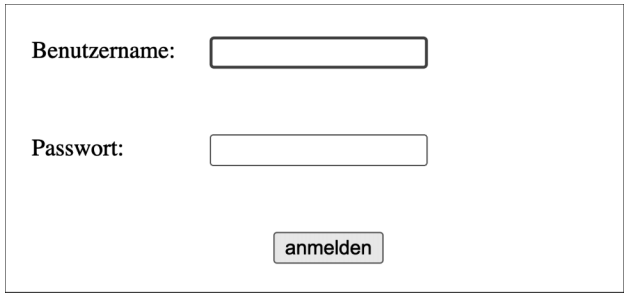


Abbildung 10.1 Anmeldeformular

Neben den bereits erwähnten Uncontrolled Components gibt es Controlled Components, um Formulare zu implementieren.

10.2 Controlled Components

Schnellstart

Eine *Controlled Component* ist fest mit dem State einer Komponente verbunden. Eine Änderung am Formular-Element bedeutet gleichzeitig eine Änderung des States.

Gleiches gilt für die Gegenrichtung. Dazu verbinden Sie den State mit der `value`-Prop des Formularelements und den `onChange`-Handler mit dem Setter des States:

```
import React, { useState } from 'react';
import './App.css';

const App: React.FC = () => {
  const [name, setName] = useState('');

  return (
    <input
      type="text"
      value={name}
      onChange={(event) => setName(event.target.value)}
    />
  );
}

export default App;
```

Listing 10.12 Implementierung einer Controlled Component (»src/App.tsx«)

Mit *Controlled Components* kommen Sie in React-Applikationen deutlich häufiger in Berührung als mit der Uncontrolled-Variante. Der Grund ist vor allem, dass Controlled Components deutlich komfortabler in der Benutzung sind. Damit Sie ein Gefühl für den Umgang mit dieser Art von Formular-Implementierung bekommen, setzen wir in den folgenden Abschnitten ein Formular zur Verwaltung von Büchern um, über das Sie neue Datensätze anlegen, aber auch bestehende modifizieren können. Der Gedanke hinter den Controlled Components ist, dass ein Formularelement seinen eigenen State verwaltet und Sie diesen mit dem State einer Komponente synchronisieren. Im ersten Schritt erzeugen Sie im *src*-Verzeichnis eine neue Datei mit dem Namen *Form.tsx*. Den Kern der Formularimplementierung bildet die `Form`-Komponente, die Sie als Funktionskomponente umsetzen:

```
import React, {
  ChangeEvent,
  FormEvent,
  ReactElement,
  useEffect,
  useState,
} from 'react';
import { Book, InputBook } from './Book';
```

```
type Props = {
  onSave: (book: InputBook) => void;
  book?: Book;
};

const initialBook: InputBook = {
  title: '',
  author: '',
  isbn: '',
};

const Form: React.FC<Props> = ({ onSave, book: inputBook }) => {
  const [book, setBook] = useState<InputBook>(initialBook);

  useEffect(() => {
    if (inputBook) {
      setBook(inputBook);
    }
  }, [inputBook]);

  function handleChange(event: ChangeEvent<HTMLInputElement>): void {
    setBook((prevBook) => {
      return { ...prevBook, [event.target.name]: event.target.value };
    });
  }

  function handleSubmit(event: FormEvent<HTMLFormElement>): void {
    event.preventDefault();
    onSave(book);
  }

  return (
    <form className="Form" onSubmit={handleSubmit}>
      <div>
        <label htmlFor="title">Titel:</label>
        <input
          type="text"
          id="title"
          name="title"
          value={book.title}
          onChange={handleChange}
          data-testid="title"
        />
```

```
      </div>
      <div>
        <label htmlFor="author">Autor:</label>
        <input
          type="text"
          id="author"
          name="author"
          value={book.author}
          onChange={handleChange}
          data-testid="author"
        />
      </div>
      <div>
        <label htmlFor="isbn">ISBN:</label>
        <input
          type="text"
          id="isbn"
          name="isbn"
          value={book.isbn}
          onChange={handleChange}
          data-testid="isbn"
        />
      </div>
      <div>
        <button type="submit" data-testid="submit">
          speichern
        </button>
      </div>
    </form>
  );
};

export default Form;
```

Listing 10.13 Aufbau der Formularkomponente (»src/Form.tsx«)

Damit Ihr Formular sowohl mit neuen als auch mit bestehenden Daten umgehen kann, müssen Sie zusätzlich zum `Book`-Typ einen weiteren Typ definieren. Dieser Typ trägt den Namen `InputBook` und hat eine optionale `id`- sowie eine optionale `rating`-Eigenschaft. In TypeScript erreichen Sie dies, indem Sie den `Omit`-Typ wie in Listing 10.14 in der Datei *Book.ts* verwenden:

```
export type Book = {
  id: number;
  title: string;
  author: string;
  isbn: string;
  rating: number;
};

export type InputBook = Omit<Book, 'id' | 'rating'> & {
  id?: number;
  rating?: number;
};
```

Listing 10.14 Definition des »InputBook«-Typs (»src/Book.ts«)

Mit dem Omit-Typ nutzen Sie den Book-Typ und entfernen die beiden Eigenschaften `id` und `rating`. Mit dem `&`-Operator fügen Sie den entstandenen Typ mit einem weiteren Typ zusammen, der beide Eigenschaften optional macht.

Die `Form`-Komponente akzeptiert als Prop eine `onSave`-Funktion, die wiederum ein `InputBook`-Objekt erhält. Diese Funktion können Sie von der Elternkomponente an die Form-Komponente übergeben, um den Datensatz zu speichern. Die zweite Prop ist ein `Book`-Objekt. Dabei handelt es sich um einen bestehenden Datensatz, den Sie mit der `Form`-Komponente ändern können. Diese Prop ist optional, Sie müssen sie also nicht zwingend übergeben.

Die beiden Funktionen des Formulars, also das Anlegen neuer und das Modifizieren bestehender Datensätze, führen jedoch zu einem Problem: Welche Information liegt im State der Komponente? Der Fall eines neuen Datensatzes ist verhältnismäßig einfach umzusetzen: Sie definieren einen initialen Wert für den State, in dem die drei verpflichtenden Eigenschaften des `InputBook`-Typs leere Zeichenketten als Werte aufweisen, und übergeben dieses Objekt beim Aufruf an die `useState`-Funktion.

Interessanter ist die Verarbeitung eines bestehenden Datensatzes, den Sie mittels Prop an die Komponente übergeben. Hier kommt ein Effect-Hook ins Spiel. Wenn Sie die Prop, die Sie im Beispiel im Destructuring-Statement in `inputBook` umbenennen, als Abhängigkeit des Effect-Hooks definieren, können Sie auf die Änderungen dieser Prop reagieren. React führt diesen Effect-Hook sowohl initial als auch bei jeder Änderung der Prop aus. Sie müssen dann nur noch prüfen, ob `inputBook` existiert und einen Wert aufweist, und können diesen Wert dann direkt in den State schreiben.

Die JSX-Struktur der `Form`-Komponente besteht aus einem `form`-Element, drei `input`-Elementen für die Eingabe des Titels, des Autors und der ISBN, jeweils mit zugehörigem `label`-Element, und schließlich einem Submit-Button zum Absenden des Formulars. Die Besonderheit dieses Formulars besteht darin, dass jedes der `input`-Ele-

mente direkt mit dem State der Komponente verbunden ist. Die `value`-Eigenschaft des Elements enthält den Wert der zugehörigen State-Eigenschaft; im Fall des Titels ist das beispielsweise `book.title`. Das führt dazu, dass das Element stets den Wert des States anzeigt.

Wichtig ist außerdem der `change`-Handler: Lassen Sie diesen weg, können Sie den Wert des Formularelements nicht mehr ändern. Der `change`-Handler ist für alle drei Felder die gleiche Funktion, sodass Sie ihn in Form der `handleChange`-Funktion auslagern können. Die Funktion erhält die Repräsentation des Change-Events als Argument. Löst der Browser ein Change-Event aus, verändern Sie den State und nutzen den Spread-Operator, um ein neues Objekt mit den Eigenschaften des bisherigen States zu erstellen. Außerdem setzen Sie eine dynamische Eigenschaft. Der Name dieser Eigenschaft stammt aus dem `name`-Attribut des geänderten `input`-Elements, und der Wert ist der veränderte Wert, auf den Sie über die `value`-Eigenschaft zugreifen können. Diese Vorgehensweise überschreibt den Wert des ursprünglichen State-Objekts und führt so zur Aktualisierung des States und damit auch des Formulars.

Der letzte Schritt besteht schließlich aus der Umsetzung des `submit`-Handlers für das Formular. Auch diese Funktion schreiben Sie nicht direkt in die JSX-Struktur, um sie nicht unnötig aufzublähen. Die `handleSubmit`-Funktion sorgt mit einem Aufruf der `preventDefault`-Methode dafür, dass der Browser nicht versucht, das Formular direkt zum Server zu senden und dabei die Seite neu zu laden.

Damit können Sie die Form-Komponente in Ihre `App`-Komponente einbinden und das Ergebnis testen. In Listing 10.15 sehen Sie, wie Sie auch das Bearbeiten einer existierenden Komponente simulieren können:

```
import React from 'react';
import './App.css';
import Form from './Form';

const book = {
  id: 1,
  title: 'JavaScript - das umfassende Handbuch',
  author: 'Philip Ackermann',
  isbn: '978-3836286299',
  rating: 5,
};
const App: React.FC = () => {
  return <Form onSave={(book) => console.log(book)} book={book} />;
};

export default App;
```

Listing 10.15 Einbindung der »Form«-Komponente (»src/App.tsx«)

Betätigen Sie den SUBMIT-Button, gibt Ihnen diese Implementierung den zu speichernden Datensatz auf der Browserkonsole aus. Das Ergebnis sehen Sie in Abbildung 10.2.

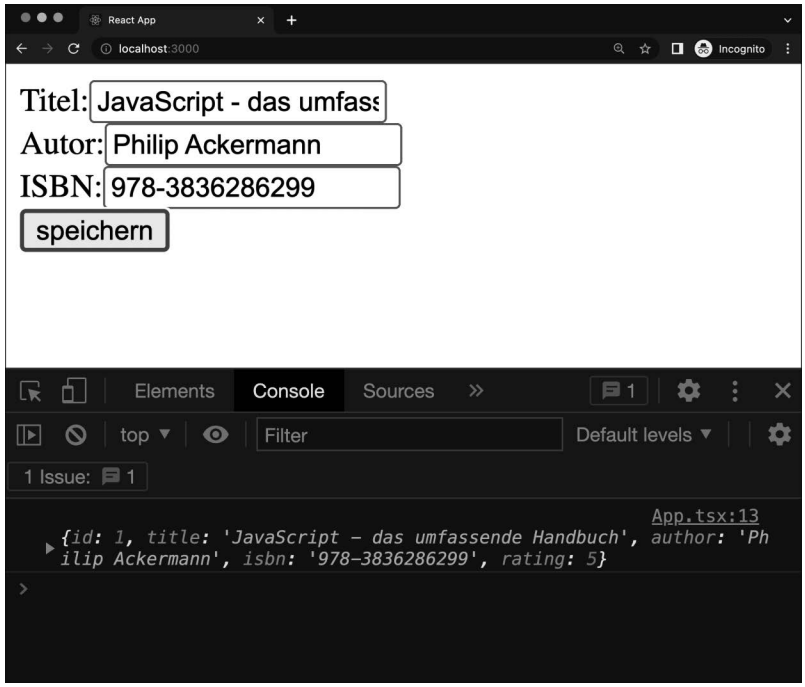


Abbildung 10.2 Anzeige des Formulars im Browser

Möchten Sie das Anlegen eines Datensatzes testen, müssen Sie die `book`-Prop entfernen. Dadurch rendert React die Komponente mit dem initialen Datensatz.

Mit diesem Stand funktioniert die `Form`-Komponente zwar; bei einer Änderung an der Komponente müssen Sie die Funktionalität jedoch jedes Mal erneut manuell testen. Also ist es an der Zeit, die Features der Komponente durch Unittests abzusichern. Dafür erzeugen Sie im *src*-Verzeichnis eine neue Datei mit dem Namen *Form.spec.tsx* und implementieren dort je einen Test für das Anlegen und Modifizieren von Datensätzen. Die Tests rendern die Komponente, interagieren mit ihr und prüfen schließlich, ob die Funktion in der `onSave`-Prop korrekt ausgeführt wurde. Den Quellcode dieser Tests finden Sie in Listing 10.16:

```
import { fireEvent, render, screen } from '@testing-library/react';
import Form from './Form';

describe('Form', () => {
  it('should create a new Book', () => {
    const onSave = jest.fn();
```

```
    render(<Form onSave={onSave} />);
    fireEvent.change(screen.getByTestId('title'), {
      target: { value: 'Design Patterns' },
    });
    fireEvent.change(screen.getByTestId('author'), {
      target: { value: 'Erich Gamma' },
    });
    fireEvent.change(screen.getByTestId('isbn'), {
      target: { value: '978-0201633610' },
    });
    fireEvent.click(screen.getByTestId('submit'));
    expect(onSave).toHaveBeenCalledWith({
      title: 'Design Patterns',
      author: 'Erich Gamma',
      isbn: '978-0201633610',
    });
  });

  it('should modify an existing Book', () => {
    const book = {
      id: 2,
      title: 'Clean Code',
      author: 'Robert Martin',
      isbn: '978-0132350884',
      rating: 4,
    };
    const onSave = jest.fn();
    render(<Form onSave={onSave} book={book} />);
    fireEvent.change(screen.getByTestId('author'), {
      target: { value: 'Robert C. Martin' },
    });
    fireEvent.click(screen.getByTestId('submit'));
    expect(onSave).toHaveBeenCalledWith({
      id: 2,
      title: 'Clean Code',
      author: 'Robert C. Martin',
      isbn: '978-0132350884',
      rating: 4,
    });
  });
});
```

Listing 10.16 Tests für die »Form«-Komponente (»src/Form.spec.tsx«)

Der erste Test rendert die Komponente ohne `book`-Prop und mit einer Spy-Funktion, die er über die `onSave`-Prop an die Komponente übergibt. Nach dem Ausfüllen aller Felder und dem Absenden des Formulars prüft der Test, ob die Spy-Funktion mit dem korrekten Objekt aufgerufen wurde.

Der zweite Test sorgt dafür, dass die Komponente mit einem bestehenden Datensatz gerendert wird, und übergibt dabei ebenfalls eine Spy-Funktion. Der Test modifiziert lediglich den Namen des Autors und sendet das Formular ab. Der Test prüft auch hier, dass die Modifikation übernommen wurde, und stellt sicher, dass alle übrigen Eigenschaften des Objekts unverändert geblieben sind.

Der TDD-Zyklus (siehe Kapitel 9) besagt, dass es bei grünen Tests erlaubt ist, Refactorings durchzuführen. Die `Form`-Komponente ist verhältnismäßig umfangreich und bietet Potenzial für Verbesserungen. Sie können beispielsweise mit einem Custom Hook die Logik von der Darstellung trennen. Dafür legen Sie eine neue Datei mit dem Namen *useForm.ts* im *src*-Verzeichnis an und implementieren dort die Funktion `useForm`:

```
import { useState, useEffect, ChangeEvent, FormEvent } from 'react';
import { InputBook } from './Book';

const initialBook: InputBook = {
  title: '',
  author: '',
  isbn: '',
};

function useForm(
  onSave: (book: InputBook) => void,
  inputBook?: InputBook
): {
  book: InputBook;
  handleChange: (event: ChangeEvent<HTMLInputElement>) => void;
  handleSubmit: (event: FormEvent<HTMLFormElement>) => void;
} {
  const [book, setBook] = useState<InputBook>(initialBook);

  useEffect(() => {
    if (inputBook) {
      setBook(inputBook);
    }
  }, [inputBook]);
```

```
  function handleChange(event: ChangeEvent<HTMLInputElement>): void {
    setBook((prevBook) => {
      return { ...prevBook, [event.target.name]: event.target.value };
    });
  }

  function handleSubmit(event: FormEvent<HTMLFormElement>): void {
    event.preventDefault();
    onSave(book);
  }

  return {
    book,
    handleChange,
    handleSubmit,
  };
}

export default useForm;
```

Listing 10.17 Custom Hook zum Auslagern der Logik aus der »Form«-Komponente (»src/useForm.ts«)

Um den `useForm`-Hook zu erstellen, schneiden Sie die komplette Logik aus der Komponente aus und fügen sie in die neue Funktion ein. Im ersten Schritt müssen Sie dann dafür sorgen, dass alle benötigten Strukturen korrekt importiert werden. Hier hilft Ihnen in der Regel Ihre Entwicklungsumgebung, die die benötigten `import`-Statements für Sie erstellt. Anschließend definieren Sie die noch fehlenden Strukturen `onSave` und `inputBook` als Parameter der Funktion und legen den Rückgabewert fest, der den State, also `book`, und die beiden Funktionen `handleChange` und `handleSubmit` in einem Objekt zusammenfasst.

Mit dieser Vorbereitung können Sie die Hook-Funktion in Ihre `Form`-Komponente integrieren. Wie das funktioniert, sehen Sie in Listing 10.18:

```
import React from 'react';
import { Book, InputBook } from './Book';
import useForm from './useForm';

type Props = {
  onSave: (book: InputBook) => void;
  book?: Book;
};
```

```
const Form: React.FC<Props> = ({ onSave, book: inputBook }) => {
  const { book, handleSubmit, handleChange } = ↩
      useForm(onSave, inputBook);

  return (
    <form className="Form" onSubmit={handleSubmit}>
      <div>
        <label htmlFor="title">Titel:</label>
        <input
          type="text"
          id="title"
          name="title"
          value={book.title}
          onChange={handleChange}
          data-testid="title"
        />
      </div>
      <div>
        <label htmlFor="author">Autor:</label>
        <input
          type="text"
          id="author"
          name="author"
          value={book.author}
          onChange={handleChange}
          data-testid="author"
        />
      </div>
      <div>
        <label htmlFor="isbn">ISBN:</label>
        <input
          type="text"
          id="isbn"
          name="isbn"
          value={book.isbn}
          onChange={handleChange}
          data-testid="isbn"
        />
      </div>
      <div>
        <button type="submit" data-testid="submit">
          speichern
        </button>
```

```
      </div>
    </form>
  );
};

export default Form;
```

Listing 10.18 Einbindung des »useForm«-Hooks in die »Form«-Komponente (»src/Form.tsx«)

Weil die `useForm`-Funktion Ihnen alle benötigten Strukturen zur Verfügung stellt, können Sie diese mit einem Destructuring-Statement extrahieren und müssen nur dafür sorgen, dass Sie die `onSave`-Funktion und das optionale `inputBook`-Objekt korrekt an die Funktion übergeben.

Ob Ihre Komponente nach der Änderung noch weiterhin funktioniert, können Sie überprüfen, indem Sie die Komponente manuell im Browser testen oder Ihre Tests ausführen.

10.2.1 Synthetic Events

Die Event-Handler-Funktionen von React erhalten nicht die nativen Event-Objekte des Browsers, sondern Wrapper-Objekte, die sogenannten *Synthetic Events*. Diese sorgen dafür, dass sich die Events über alle Browser konsistent verhalten. Vor Version 17 von React geschah das Wrappen der Events auch aus Performancegründen. Die Events wurden von React wiederverwendet und die Eigenschaften zurückgesetzt, sobald der Event-Handler ausgeführt wurde. Das hatte zur Konsequenz, dass Sie in Operationen, wie beispielsweise beim Setzen des States mit einer Callback-Funktion, nicht mehr direkt auf die Eigenschaften der Event-Objekte zugreifen konnten, sondern sie zuvor in Variablen speichern mussten. Da diese Optimierung nicht den erwarteten Performanceschub brachte, sondern zu viel Verwirrung in der Community führte, hat sich das React-Team entschieden, dieses Event-Pooling zu entfernen.

10.3 Der Upload von Dateien

Bisher haben Sie mit einfachen Textfeldern gearbeitet. Doch bieten Formulare noch zahlreiche weitere Interaktionsmöglichkeiten. Eine der herausforderndsten sind Dateiuploads, da sie aus dem vorgestellten Schema ausbrechen. Als Schnittstelle zu den Benutzer*innen verwenden Sie für den Upload ein `input`-Element vom Typ `file`. Dieses Element nutzen Sie nur zum Lesen beim Absenden des Formulars und setzen es demnach als Uncontrolled Component.