Inkl. React Native und Redux

SPA, Props, Component, Hooks, SSR, TypeScript, Flow, PWA,

```
export function createContext<T>(
  defaultValue: T,
  calculateChangedBits: ?(a: T, b: T) => number
): ReactContext<T> {
  if (calculateChangedBits === undefined) {
    calculateChangedBits = null;
  } else {
    if (__DEV__) {
      warningWithoutStack(
        calculateChangedBits === null ||
          typeof calculateChangedBits === 'function',
        'createContext: Expected the optional second
        argume
          'funct
```

Sebastian Springer

# React

## Das umfassende Handbuch

- Für Einsteiger und Fortgeschrittene
- Umgang mit Komponentenarchitektur und Virtual DOM
- Tests, Debugging, Material Design Components, Router, Serverkommunikation, i18n, Server-Side Rendering u. v. m.

Alle Beispielprojekte zum Download

Rheinwerk Computing

# Vorwort

React steht für mich für eine schnelle und leichtgewichtige, aber trotzdem professionelle Webentwicklung. Die Bibliothek lässt Ihnen bei der Entwicklung viele Freiheiten, was den Aufbau und die Gestaltung einer Applikation angeht. Das ist Fluch und Segen zugleich. Gerade für Einsteiger wird es an dieser Stelle schwierig: Wo soll ich anfangen? Wie strukturiere ich meine Applikation? Wie löse ich konkrete Problemstellungen? Welche Bibliotheken und Hilfsmittel benötige ich für die Entwicklung meiner Applikation? Das sind nur einige Fragen, die man sich zu Beginn auf dem Weg mit React stellt, und genau an dieser Stelle setzt dieses Buch an. Zusammen implementieren wir eine vollständige Applikation, die zahlreiche Problemstellungen aus dem Praxisalltag abdeckt. Dabei lernen Sie nicht nur React selbst, sondern Teile des Ökosystems um die Bibliothek herum kennen.

Egal ob Sie erst in das Thema React einsteigen oder schon Erfahrung damit haben, ich möchte Sie einladen, dieses Buch als Gelegenheit zur aktiven Arbeit mit React zu nutzen. Versuchen Sie die Applikation selbst zu bauen, setzen Sie die verschiedenen Anforderungen um, und lassen Sie sich von den Codebeispielen und Lösungsansätzen für eigene Lösungen inspirieren. Es gibt kaum eine bessere Strategie, sich tiefer in ein Thema einzuarbeiten, als die Technologie oder das Werkzeug selbst zu verwenden, auch einmal einen Fehler zu machen und daraus zu lernen.

Für die Arbeit mit diesem Buch sollten Sie über ein solides Grundwissen in HTML, CSS und JavaScript verfügen. Falls Sie sich an der einen oder anderen Stelle unsicher sind oder sich fragen, was ein bestimmtes Sprachelement genau macht, lege ich Ihnen das Mozilla Developer Network unter *https://developer.mozilla.org/de/* ans Herz. Hierbei handelt es sich um eine umfangreiche aktuelle Referenz für alle Webtechnologien. Arbeiten Sie lieber mit Büchern, kann ich Ihnen an dieser Stelle das JavaScript-Handbuch von Philip Ackermann empfehlen. Ansonsten empfehle ich Ihnen, neugierig zu sein und Dinge auszuprobieren. Fragen Sie sich: Was passiert, wenn ich an dieser Stelle dieses oder jenes tue? Probieren Sie es aus, öffnen Sie die Entwicklerwerkzeuge Ihres Browsers, und sehen Sie sich die Auswirkungen Ihres Experiments an. Der Vorteil der frontendseitigen Webentwicklung ist, dass Sie außer dem Frontend in Ihrer Applikation nichts weiter kaputt machen können, und auch das können Sie durch den Einsatz eines Versionskontrollsystems wie Git auf ein Minimum reduzieren, da Sie immer wieder auf einen funktionierenden Stand zurückwechseln können. Neben diesen Experimenten sollten Sie auch versuchen, die Beispielapplikation eigenständig weiterzuentwickeln oder eine eigene Applikation zu bauen.

Dieses Buch ist sowohl für den Einstieg in React als auch als Nachschlagewerk für den täglichen Gebrauch gedacht. Sie können die Beispiele entweder selbst nachvollzie-

hen, indem Sie den Quellcode selbst schreiben, oder sie laden sich den Code herunter und passen ihn nach Ihren Wünschen an. Für mich ist eigentlich nur wichtig, dass Sie selbst mit React arbeiten, die Bibliothek und ihre Möglichkeiten kennenlernen und viel Spaß dabei haben.

Eine der häufigsten Fragen im Zusammenhang mit JavaScript-Bibliotheken und -Frameworks ist, welche/welches das beste ist. Natürlich ist React eine gute Wahl, wenn es um die Implementierung eines Web-Frontends geht. Allerdings sind andere Lösungen wie Angular oder Vue deshalb nicht schlechter. Versuchen Sie, sich an dieser Stelle selbst ein Bild zu machen, geben Sie den verschiedenen Ansätzen eine Chance. Für mich hat sich React in der Praxis in mehreren kleinen, aber auch großen Projekten bewährt.

## Aufbau des Buchs

Das Buch besteht aus zwei Teilen, die sich in insgesamt 18 Kapitel unterteilen. Der erste Teil des Buchs beschäftigt sich mit React selbst, der zweite Teil beleuchtet das Ökosystem der Bibliothek mit verschiedenen Problemstellungen aus dem Projektalltag.

Der erste Teil beginnt mit einer Einleitung in React, die Ihnen das Grundwissen und die wichtigsten Begriffe und Konzepte erläutert (**Kapitel 1**), und einer Erklärung, wie Sie React installieren und verwenden können (**Kapitel 2**). Anschließend lernen Sie in **Kapitel 3** und **Kapitel 4** die verschiedenen Arten von React-Komponenten kennen. Sie erfahren, wie Sie den Komponentenbaum Ihrer Applikation aufbauen und wie die Daten in diesem Baum fließen. Weiterführende Konzepte wie Higher-Order Components und Render Props ergänzen dieses Grundlagenwissen. **Kapitel 5** widmet sich der Hooks-API, einer neueren Erweiterung von React, die die Art und Weise, wie eine Applikation aufgebaut wird, entscheidend beeinflusst. Mit TypeScript lernen Sie in **Kapitel 6** ein Werkzeug kennen, das Sie bei der Umsetzung unterstützt und Ihnen zusätzliche Sicherheit im Entwicklungsprozess bietet.

React unterstützt verschiedene Ansätze, wenn es um das Styling von Komponenten geht. **Kapitel 7** stellt Ihnen einige dieser Ansätze vor und zeigt Ihnen, wie Sie sie in Ihre Applikation integrieren können. Ein weiteres Hilfsmittel im Zusammenhang mit der Qualitätssicherung einer Applikation behandelt **Kapitel 8**, in dem es um das Formulieren von Unittests geht. **Kapitel 9** schließt den ersten Teil des Buchs mit einem Blick auf Formulare ab. Mithilfe von Formularen geben Sie Ihren Benutzern die Möglichkeit, aktiv mit der Applikation zu interagieren und Daten zu produzieren.

Der zweite Teil des Buchs beginnt mit einem Kapitel über die Integration externer Komponentenbibliotheken (**Kapitel 10**). Am Beispiel von Material-UI sehen Sie, wie





Sie existierende Komponenten in Ihre Applikation integrieren können. In **Kapitel 11** sehen Sie, wie Sie mithilfe des React-Routers innerhalb Ihrer Single-Page-Applikation navigieren und damit unterschiedliche Teilbäume Ihrer Applikation rendern können. **Kapitel 12** und **Kapitel 13** beschäftigen sich mit dem zentralen State-Management in einer Applikation mit der Bibliothek Redux und mit der Behandlung von Seiteneffekten mit verschiedenen asynchronen Middleware-Implementierungen wie beispielsweise Redux-Thunk.

In **Kapitel 14** lernen Sie, wie Sie in Ihrer React-Applikation eine GraphQL-Schnittstelle ansprechen können. Diese Abfragesprache stellt eine flexible Alternative zu den herkömmlichen RESTful-Schnittstellen dar, die üblicherweise in der Webentwicklung zum Einsatz kommen. **Kapitel 15** bindet mit `react-intl` eine weitere externe Bibliothek in die Applikation ein, um Internationalisierung zu unterstützen. Hier erfahren Sie neben der reinen Übersetzung von Zeichenketten auch mehr über den Umgang mit Zahlen und Datumswerten.

Die letzten drei Kapitel beschäftigen sich mit unterschiedlichen Umgebungen, die von einer React-Applikation berührt werden. Den Beginn macht **Kapitel 16** mit einer Einführung in das serverseitige Rendering, um die Performance einer Applikation zu verbessern. Anschließend erweitern Sie Ihre Applikation um einige Features, sodass sie zu einer Progressive Web App wird, die sich auf dem System eines Benutzers installieren und auf möglichst vielen Systemen ausführen lässt (**Kapitel 17**). Die Idee von React auf unterschiedlichen Systemen wird im letzten Kapitel, **Kapitel 18**, mit React Native noch einen Schritt weitergeführt, indem Sie erfahren, wie Sie mit dieser Bibliothek native Apps mit React umsetzen können.

## Download der Codebeispiele

Sämtliche Codebeispiele dieses Buches sind auf der Website des Verlages unter *www.rheinwerk-verlag.de/4848* zum Download verfügbar.

Sollten Sie Probleme bei der Umsetzung haben, oder sollte ich trotz sorgfältiger Kontrolle einen Fehler übersehen haben, können Sie mich gerne unter *react@sebastian-springer.com* kontaktieren.

## Danksagung

Ich möchte mich bei allen Personen bedanken, die mich beim Schreiben dieses Buchs unterstützt haben. Allen voran bei Philip, der, wie schon bei meinem Node.js-Buch, die Begutachtung übernommen und viele Anmerkungen und Tipps beigesteuert hat.

Außerdem danke ich Petra Biedermann für den sprachlichen Feinschliff.

Auch dem ganzen Team vom Rheinwerk Verlag möchte ich danken und hier vor allem Anne Scheibe und Stephan Mattescheck für die Betreuung.

Schließlich möchte ich auch noch meiner Frau Alexandra herzlichen Dank für ihre Geduld und Unterstützung sagen.

**Sebastian Springer**

Aßling

# Kapitel 2
# Die ersten Schritte im Entwicklungsprozess

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

In diesem Kapitel geht es um den Entwicklungsprozess von der Initialisierung der Applikation (auf verschiedene Arten) über die Konfiguration der Entwicklungsumgebung und das Debuggen der Applikation bis hin zum Bauen der Applikation.

Der erste Schritt in die Welt eines Frameworks ist meist mit relativ 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.

Im 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 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 natürlich nur für einen sehr begrenzten Umfang und für kleinere Experimente. Sobald Sie an einem umfangreicheren Funktionsumfang arbeiten, sollten Sie den Quellcode der Applikation lokal vorhalten.

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

### 2.1.1 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 bei anderen vergleichbaren Plattformen haben Sie drei Editoren, je einen für HTML, CSS und JavaScript. Außerdem wird Ihnen das Ergebnis der Ausführung Ihres Codes in einem weiteren Abschnitt des Fensters angezeigt. Die Größe der einzelnen Sektionen lässt sich per Drag and Drop variieren, ebenso lässt sich das Layout der gesamten Plattform anpassen.

Als angemeldeter Benutzer können Sie Ihre Experimente speichern und haben über ein Dashboard eine Übersicht über die gespeicherten Projekte. Die Anmeldung kann entweder über ein Konto bei CodePen direkt oder aber über einen Login über Twitter, GitHub oder Facebook erfolgen.

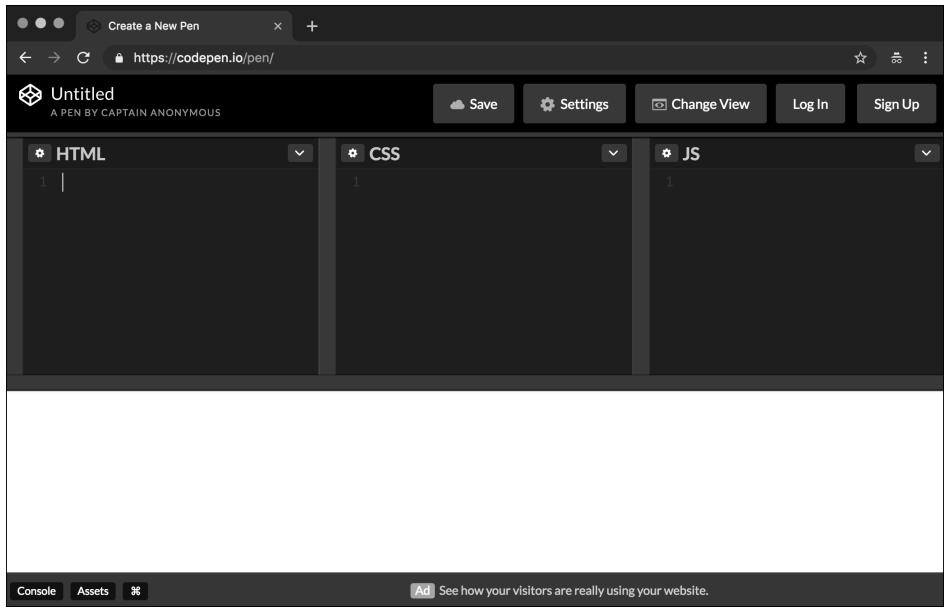


**Abbildung 2.1** CodePen

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 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, die in bestimmten Browsern noch nicht implementiert sind, zu simulieren. Babel an sich stellt lediglich die Compiler-Infrastruktur zur Verfügung und kann durch Plugins erweitert werden. Jedes dieser Plugins ist für einen bestimmten Aspekt wie beispielsweise die Unterstützung von Arrow-Funktionen verantwortlich. 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. 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 2.1** Container für die React-Applikation

Im JavaScript-Editor nutzen Sie das `ReactDOM`-Objekt, das Ihnen von der `react-dom`-Bibliothek zur Verfügung gestellt wird, um die Applikation mithilfe der `render`-Methode darzustellen. Dieser Methode übergeben Sie als erstes Argument ein JSX-Element, das das Wurzelelement Ihrer Applikation bildet. Das zweite Argument ist der Container, in den die Applikation eingehängt werden soll. Mit `document.getElementById` lokalisieren Sie den zuvor erzeugten Container. Das JSX-Element verweist auf eine Komponente, die Sie mit der Funktion `Greet` definieren.

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

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

**Listing 2.2** Quellcode der React-Applikation

Auf das Attribut `name`, das Sie der Komponente übergeben, können Sie innerhalb der Komponente zugreifen und so den Namen anzeigen. Die Anzeigesektion wird nach kurzer Zeit aktualisiert, sodass Sie das Ergebnis sehen, das wie in Abbildung 2.2 aussehen sollte.

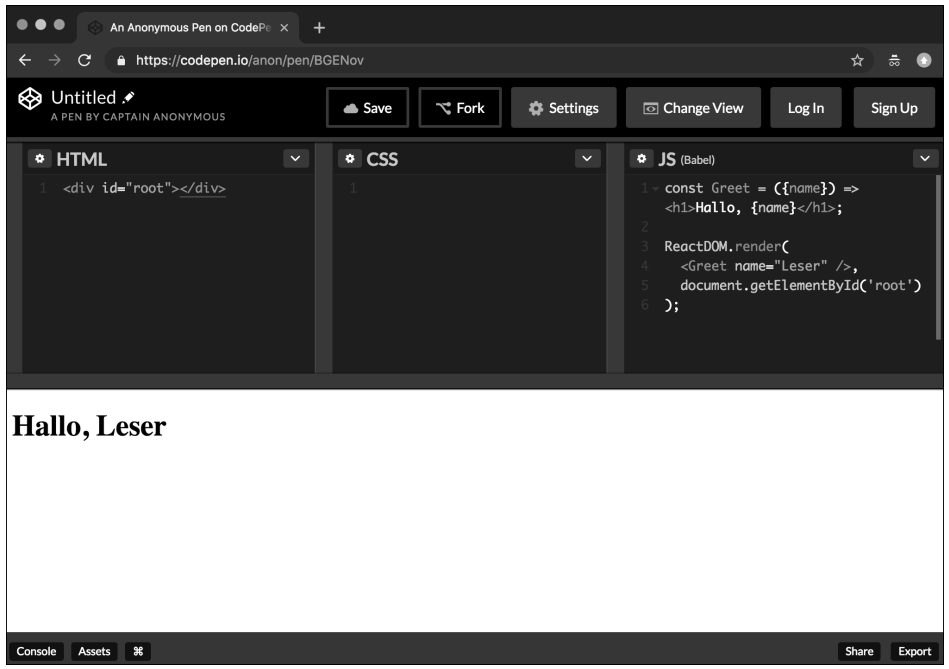


**Abbildung 2.2** React-Applikation in CodePen

Die Umsetzung auf Plattformen wie CodePen eignet sich nur für kleinere Experimente und Versuche. Arbeiten Sie an einer größeren Applikation, sollten Sie den Quellcode auf ihrem lokalen System entwickeln. Im weiteren Verlauf dieses Kapitels werde ich Ihnen weitere Möglichkeiten vorstellen, Anwendungen umzusetzen.

## 2.2 Lokale Entwicklung

Dem Vorteil, dass eine Plattform wie CodePen immer und überall verfügbar ist und Sie darüber Ihren Code mit anderen Entwicklern weltweit teilen können, steht der Nachteil gegenüber, dass das Debugging in einer solchen Umgebung nur recht 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. Wie das funktioniert, erfahren Sie in den folgenden Abschnitten.







### 2.2.1 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 schwerfällig sei. Wo 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.

Für die Verwendung von React in einer kleinen lokalen Anwendung gehen Sie ähnlich vor wie bei der Konfiguration eines Playgrounds wie beispielsweise CodePen. Zunächst erzeugen Sie eine HTML-Datei mit dem Namen *index.html*.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Meine React-Applikation</title>
    <script
      src="https://unpkg.com/react@16/umd/react.development.js"
      crossorigin
    ></script>
    <script
      src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"
      crossorigin
    ></script>
    <script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
    <script src="index.js" type="text/babel"></script>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

**Listing 2.3** Einstiegsdatei mit direkter React-Einbindung (»index.html«)

In Listing 2.3 sehen Sie die Struktur dieser Einstiegsdatei. Zu Beginn müssen Sie dafür sorgen, dass die Bibliotheken `react` und `react-dom` geladen werden. Dies erreichen Sie über das CDN *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.js*-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.

Der Inhalt der *index.js*-Datei entspricht dem Quellcode, den Sie im ersten Beispiel in CodePen als JavaScript ausgeführt haben. Listing 2.4 fasst diesen nochmal für Sie zusammen.

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

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

**Listing 2.4** JavaScript-Quellcode der lokalen React-Applikation

Laden Sie die *index.html*-Datei lokal in Ihren Browser, sehen Sie lediglich eine weiße Seite. Bei einem Blick in die JavaScript-Konsole Ihres Browsers sehen Sie eine Fehlermeldung, dass die Datei *index.js* nicht geladen werden konnte. Das liegt daran, dass Babel versucht, diese Datei über einen asynchronen Request zu laden, und solche Anfragen aus Sicherheitsgründen nicht über das *file://*, sondern nur über *http://*-Protokoll ausgeführt werden können.

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

**Lokaler Webserver**

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 trägt den Namen `http-server` und ist als NPM-Paket verfügbar.

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

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 -yes
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.

Führen Sie den `http-server` wie beschrieben aus, 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.

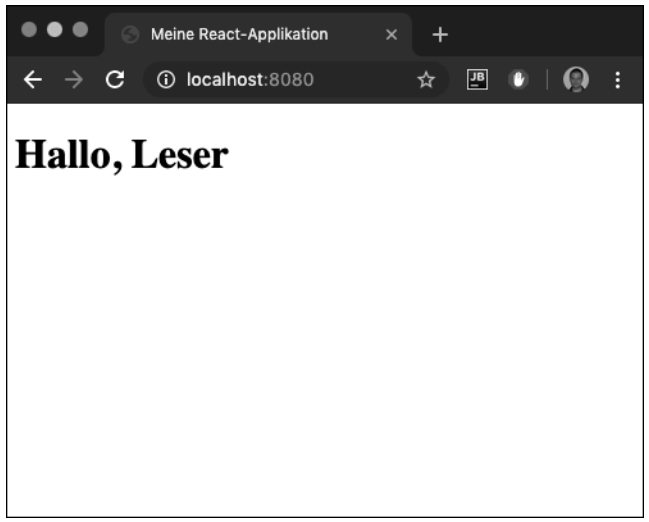


**Abbildung 2.3** Lokale Ausführung der React-Applikation

## 2.3 Der Einstieg in die Entwicklung mit React

Nahezu alle großen JavaScript-Frameworks verfügen über Kommandozeilenwerkzeuge, die Ihnen Routineaufgaben 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 Webapplikationen 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.

### 2.3.1 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 ist auf der V8-Engine aufgebaut, die auch im Chrome-Browser zum Einsatz kommt. Die Plattform verfügt über eine Reihe von Kernmodulen, die Ihnen als Entwickler 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
npm -v
```

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, also die korrekte Vervollständigung von begonnenen Variablen- und Methodennamen, bietet.

#### 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 – also Chrome, Firefox, Edge oder Safari – verfügen. Die Beispiele in diesem Buch, sofern sie spezifische Browserfeatures betreffen, beziehen sich wegen dessen hohen Verbreitungsgrads auf Chrome. Funktionalitäten wie die Entwicklerwerkzeuge finden Sie auch in den anderen Browsern in ähnlicher Form und einem ähnlichen Umfang wieder.

React unterstützt alle modernen Browser. Falls Sie ältere Versionen wie beispielsweise den Internet Explorer in Version 9 bis 11 unterstützen wollen, müssen Sie verschiedene Polyfills installieren. Diese sind im Paket `react-app-polyfill` zusammengefasst und lassen sich über den NPM oder Yarn installieren. Anschließend müssen Sie die für Ihren Browser passende Version lediglich noch einbinden. Listing 2.5 demonstriert dies am Beispiel des Internet Explorers 10.

```
import 'react-app-polyfill/ie9';
```

**Listing 2.5** Polyfills für den Internet Explorer 10 laden

### 2.3.2 Installation von Create React App

Die bisher vorgestellten Varianten, 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://facebook.github.io/create-react-app/*. Im Gegensatz zu anderen CLI-Werkzeugen wie beispielsweise dem Angular-CLI, die Sie über den gesamten Lebenszyklus einer Applikation begleiten, unterstützt Sie Create React App lediglich beim Erstellungsprozess der Applikation.

Bei der Verwendung von Create React App müssen Sie kaum etwas beachten; 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 weit verbreitete Pakete für Standardaufgaben wie 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 React Scripts.

Ein weiteres Komfortfeature, das Ihnen *Webpack* 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.

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, 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.

#### Globale Installation über NPM

Die erste Variante mit Create React App, die ich Ihnen vorstellen möchte, nutzt den NPM. Mit dem Kommando

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

**Listing 2.6** Create React App mit NPM global installieren

installieren Sie Create React App global auf Ihrem System, sodass Ihnen das Werkzeug systemweit auf der Kommandozeile zur Verfügung steht. Nach der Installation können Sie mit dem Befehl `create-react-app` Ihre Applikation initialisieren lassen. Hierfür müssen Sie lediglich das Projektverzeichnis angeben. Dabei kann es sich entweder um einen absoluten oder um einen relativen Pfad handeln.

```
create-react-app supertrumpf
```

**Listing 2.7** Initialisierung der Applikation

Führen Sie das Kommando aus Listing 2.7 auf der Kommandozeile Ihres Systems aus, wird im aktuellen Verzeichnis ein Unterverzeichnis mit dem Namen *supertrumpf* erzeugt, in dem die Applikation erstellt wird.

**Der Node Package Manager**

Der Name dieses Werkzeugs ist etwas irreführend. NPM ist ein Paketmanager für JavaScript, den Sie sowohl für 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*-Verzeichnis 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 ls`: Listet die aktuell installierten Pakete auf.
- `npm init`: Erzeugt eine *package.json*-Datei.

### Initialisierung mit »npm init« und »npx«

Die globale Installation von Create React App auf Ihrem System ist häufig nicht nötig, da es sich dabei um ein Werkzeug handelt, das Sie nur zum Aufsetzen Ihrer Applikation benötigen. Die globale Installation hat einen weiteren Nachteil: Sie müssen für sämtliche Applikationen die gleiche globale Version eines Werkzeugs verwenden. Bei Create React App fällt dies nicht so sehr ins Gewicht, da das entscheidende Paket `react-scripts` ist. Generell geht die Tendenz vor allem aus diesem Grund weg von globalen Installationen. Eine Alternative bietet das Kommando `npm init`. Mit

```
npm init react-app supertrumpf
```

**Listing 2.8** Initialisierung einer React-Applikation mit »npm init«







erzeugen Sie ebenfalls ein lokales Verzeichnis *supertrumpf*, in dem Ihre Applikation initialisiert wird. Ist Create React App nicht auf Ihrem System installiert, wird es temporär heruntergeladen, ausgeführt und anschließend wieder verworfen. Normalerweise verwenden Sie das Kommando `npm init`, um lediglich eine neue *package.json*-Datei zu erzeugen. In diesem Fall übergeben Sie zusätzlich den Initialisierer `react-app`. NPM verwendet in diesem Fall das Hilfswerkzeug `npx`, um das Paket mit dem Namen `create-` und der Bezeichnung des Initialisierers zu herunterzuladen und es auszuführen. Alternativ können Sie `npx` auch direkt nutzen. In diesem Fall lautet der zugehörige Befehl:

```
npx create-react-app supertrumpf
```

Die Wirkung beider Kommandos ist äquivalent.

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

### Globale Installation mit Yarn

Analog zur globalen Installation mit NPM können Sie mit den folgenden Kommandos Ihre Applikation auch mit Yarn initialisieren lassen:

```
yarn global add create-react-app
create-react-app supertrumpf
```

Yarn installiert, ebenso wie NPM, die ausführbare Datei des Pakets in ein Verzeichnis, das im Suchpfad des Systems liegt. Anschließend wird das Werkzeug verwendet, um die Applikation zu initialisieren.

**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/en/docs/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 als Entwickler 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 mit Yarn mit dem Kommando `yarn add react`.

**Hinweis**

Im weiteren Verlauf dieses Buchs wird Yarn als Paketmanager verwendet. Sämtliche Beispiele können Sie jedoch mit geringen Anpassungen auch mit NPM nachvollziehen.

#### Initialisierung mit »yarn create«

Zu guter Letzt gibt es auch bei Yarn noch eine Variante der Projektinitialisierung, bei der Sie Create React App nicht auf Ihrem System installieren müssen. 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 supertrumpf
```

Egal für welche Variante der Installation Sie sich entschieden haben, verfügen Sie nach der Ausführung des jeweiligen Kommandos 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 wie beispielsweise die Betriebssystem- und Browserversionen ausgegeben.
- `--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.
- `--typescript`: Seit Version 2.1.0 von Create React App wird TypeScript als Alternative zu nativem JavaScript unterstützt. Mit dieser Option werden sowohl TypeScript als auch weitere benötigte Bibliotheken installiert.

Vor allem die neueren Optionen wie `--use-pnp` oder `--typescript` zeigen an, 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 `yarn outdated` im Verzeichnis Ihrer Applikation ausführen.

Um beispielsweise von Version 2.1.0 auf Version 2.1.1 zu aktualisieren, verwenden Sie den folgenden Befehl auf der Kommandozeile im Verzeichnis Ihrer Applikation:

```
yarn add --exact react-scripts@2.1.1
```

**Listing 2.9** Aktualisieren des »react-scripts«-Pakets

**2.1.1 (October 31, 2018)**

Happy Halloween 🎃 👻 ! This spooky release brings a treat: decorator support in TypeScript files!

🐛 **Bug Fix**

- `babel-preset-react-app`
  - #5659 Add support for decorators. (@Timer)
- `react-scripts`
  - #5621 fix 'Duplicate string index signature' in ProcessEnv. (@xiaoxiangmoe)

🚀 **Enhancement**

- `babel-preset-react-app`
  - #5659 Add support for decorators. (@Timer)

📝 **Documentation**

- #5658 Update making-a-progressive-web-app.md. (@jakeboone02)
- #5635 Update minimum node version to 8.10 in README. (@iansu)
- #5629 Add link to cra-ts migration guide. (@Vinnl)

🏠 **Internal**

- `react-error-overlay`
  - #4709 Expose `reportRuntimeError`. (@hipstersmoothie)
- `babel-plugin-named-asset-import`
  - #5575 add tests for named-asset-imports plugin. (@NShahri)
- `react-scripts`
  - #5651 Make serviceWorker config argument optional in typescript. (@eddedd88)

**Committers: 8**

- Andrew Lisowski (hipstersmoothie)
- Eduardo Duran (eddedd88)
- Ian Sutherland (iansu)
- Jake Boone (jakeboone02)

**Abbildung 2.4** Auszug aus dem Changelog von Create React App







Ein solches Update sollten Sie jedoch nicht einfach 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. 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 `yarn outdated`, um über neue Releases informiert zu werden. Die Entwickler von React pflegen 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 hier nicht jede Stelle in Ihrer Applikation von Hand anpassen müssen, existiert das Projekt `react-codemod`. Die Kombination aus JSCodeshift und den Codemod-Scripts erlaubt Ihnen, bestimmte Änderungen an Ihrem Quellcode automatisiert durchführen lassen. Was zunächst etwas gespenstisch klingt, wird im großen Stil bei Facebook an produktivem Code regelmäßig durchgeführt.

Bei Änderungen von zentralen Schnittstellen ist die Anpassung des Quellcodes von Hand keine Option, auch die Manipulation über reguläre Ausdrücke gelangt schnell an ihre Grenzen, also implementierten die Entwickler von Facebook das Werkzeug JSCodeshift. Es verwendet recast, um einen Abstract Syntax Tree, kurz 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.

### 2.3.3 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 2.10 sehen Sie, wie Sie statt Port `3000` alternativ Port `8181` nutzen können.

  ```
  PORT=8181 yarn start
  ```

  **Listing 2.10** 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, kommen 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 immer, 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, also eine Konsolenausgabe ohne Warnungen, zu verfolgen.
- `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.
- `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. Dieser kann mithilfe des `test`-Scripts ausgeführt werden.
- `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:

```
yarn start
```

**Listing 2.11** 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, wie Sie sie in Abbildung 2.5 sehen.

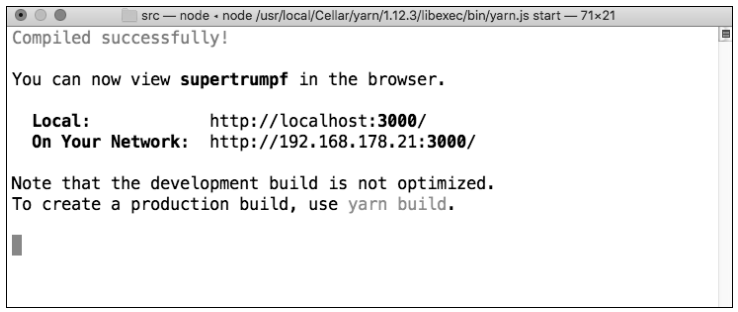


**Abbildung 2.5** Konsolenausgabe von »yarn start«

Die Ausführung von `yarn 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.

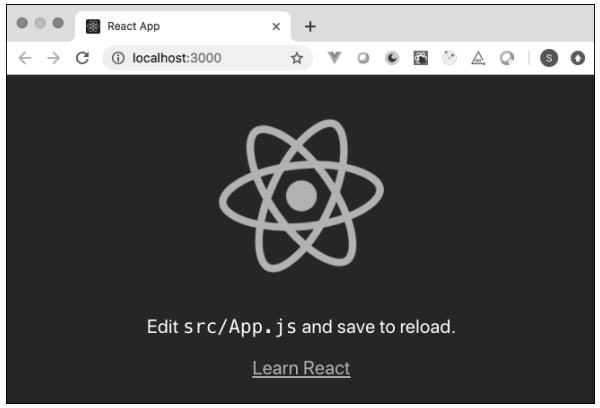


**Abbildung 2.6** Initiale React-Applikation

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

### 2.3.4 Serverkommunikation im Entwicklungsbetrieb

Gerade in der Entwicklung ist eine strikte Trennung zwischen Frontend und Backend von Vorteil. Nutzen Sie den Webpack Dev Server, müssen Sie sich zum Beispiel nicht um das Neuladen Ihrer Applikation kümmern. Eine solche Applikation kommt jedoch in der Regel nicht ohne Backend aus, das sich um die Persistierung von Daten und weitere Aspekte wie beispielsweise die Authentifizierung von Benutzern kümmert. Dieses wird allerdings nicht über den Dev Server ausgeliefert, sondern als separater Serverprozess ausgeführt. 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.

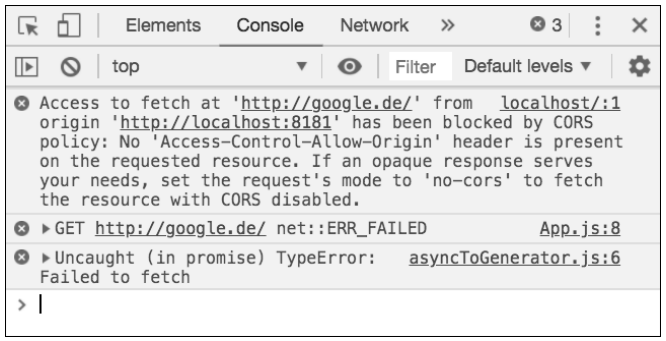


**Abbildung 2.7** Fehlermeldung beim Zugriff auf ein Backend

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 2.12** 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«.







### 2.3.5 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 2.13.

```
HTTPS=true yarn start
```

**Listing 2.13** 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.

## 2.4 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, finden Sie in diesem Abschnitt eine kurze Erklärung zu den verschiedenen Dateien und Verzeichnissen.

Im Basisverzeichnis Ihrer Applikation liegen 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.
- *yarn.lock*: In der *yarn.lock*-Datei werden die exakten Versionen und die Integritäts-Hashes der installierten Abhängigkeiten und deren Unterabhängigkeiten festgehalten.

Neben den Dateien befindet 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, da diese Änderungen nicht an andere Entwickler verteilt werden und sie so verlorengehen.
- *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 meisten 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 Applikationspaketes 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 2.14 am Beispiel des Favicons sehen.

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

**Listing 2.14** 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 2.15 sehen.

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

**Listing 2.15** 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 `yarn test` ausführen können. In der *index.css*-Datei finden Sie allgemeine Styledefinitionen wie 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.







## 2.5 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`+⇧+`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.

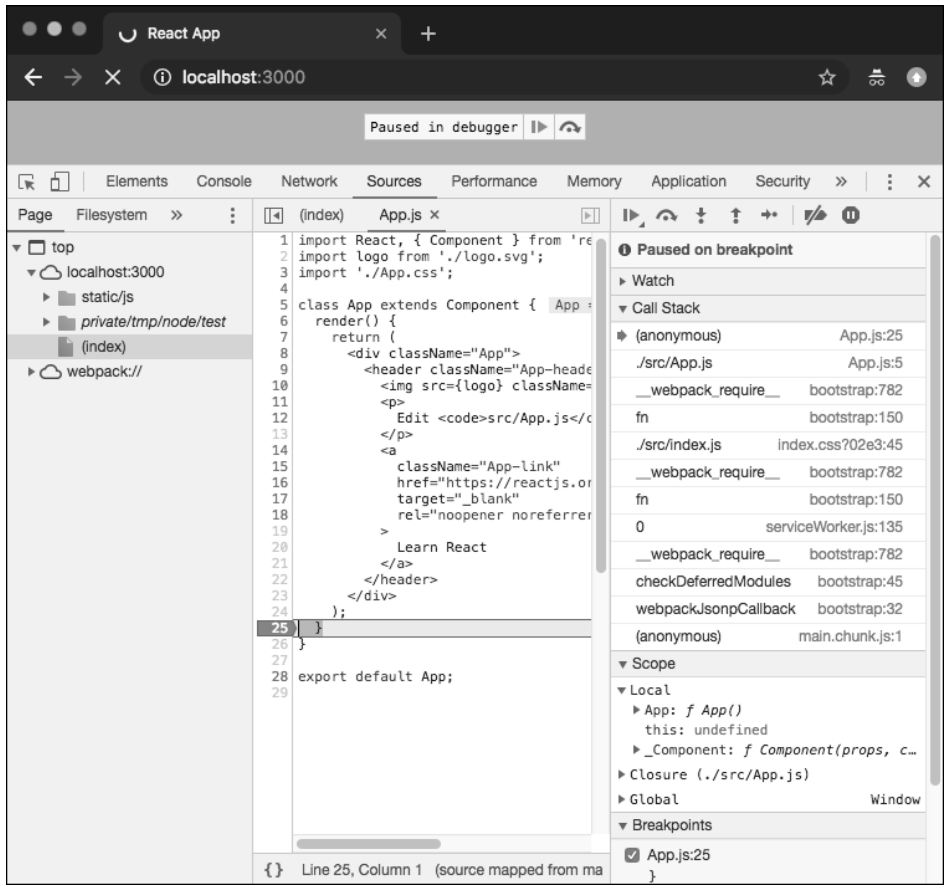


**Abbildung 2.8** Angehaltener Debugger in einer React-Applikation

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**: in die aufzurufende Funktion hineinspringen. Diese Aktion fügt einen Eintrag zum Callstack hinzu.
- **Step out**: aus der aktuellen Funktion herausspringen. 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. Diese finden Sie im Fall von Visual Studio Code in Form eines zusätzlichen IDE-Plugins unter *https://github.com/microsoft/vscode-chrome-debug*, und auch die Dokumentation von WebStorm enthält einen eigenen Abschnitt unter *https://www.jetbrains.com/help/webstorm/debugging-javascript-in-chrome.html* hierfür.







### 2.5.1 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 einen Tab mit der Bezeichnung REACT. Hier sehen Sie die Struktur Ihrer Applikation und können die einzelnen Komponenten inspizieren sowie deren Props und State überprüfen. Wie die React Dev Tools im Browser aussehen, zeigt Ihnen Abbildung 2.9.

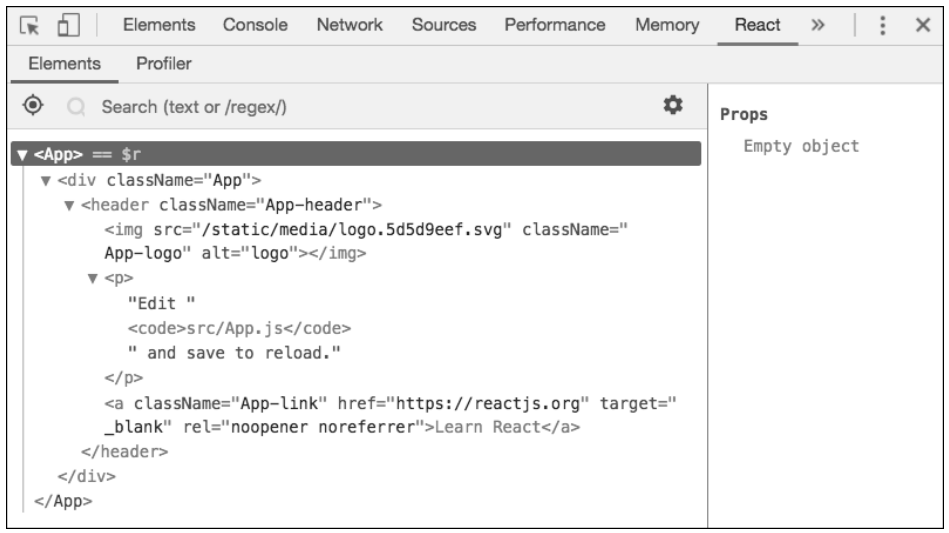


**Abbildung 2.9** Die React Dev Tools in Chrome

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

## 2.6 Applikation bauen

Im Entwicklungsbetrieb wird der Quellcode zwar durch den Webpack Dev Server modifiziert, sodass auch ältere Browser den Quellcode ausführen können. Optimiert wird der Code hier jedoch noch nicht. Um Ihre Applikation für den Produktivbetrieb vorzubereiten, führen Sie den Befehl `yarn build` aus.

Als Ergebnis wird ein neues Verzeichnis mit dem Namen *build* erstellt. In diesem Verzeichnis befinden sich sämtliche Ressourcen, die benötigt werden, um die Applikation auszuführen. 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 darüber ausliefern. Eine der einfachsten Lösungen hierfür bietet das NPM-Paket `serve`. Dieses installieren Sie mit dem Kommando `yarn global add serve` und führen dann den Server mit dem Befehl `serve -s build` im Wurzelverzeichnis Ihrer Applikation aus.

## 2.7 Zusammenfassung

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

- Sie kennen jetzt die verschiedenen Möglichkeiten, React in eine Applikation einzubinden.
- Mit Playgrounds wie CodePen lassen sich einfache Experimente mit React durchführen, indem die Bibliothek direkt eingebunden wird. 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 Kommandozeilen-Optionen sowie Umgebungsvariablen beeinflussen.
- Alle modernen Browser verfügen über leistungsstarke Entwicklerwerkzeuge, die Ihnen bei der Fehlersuche helfen. Außerdem existieren mit den React Dev 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 9
# Formulare in React

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



Bisher hat sich die Interaktion eines Benutzers mit Ihrer Applikation auf einfache Klick-Events auf bestimmte Elemente beschränkt. 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 Formik, wie Sie externe Bibliotheken zur Formularvalidierung einsetzen können.

## 9.1 Uncontrolled Components

Die einfachste Art, mit Formularen umzugehen, sind *Uncontrolled Components*. Der Name stammt von der Tatsache, 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: Auf sie greifen Sie, wie in alten jQuery-Applikationen, über eine Referenz auf den Wert zu.

### 9.1.1 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 Elemente Ihrer Applikation zuzugreifen.

Den Kern der Refs in React bildete bisher die Methode `createRef`. Sie wurde mit der Einführung der Hooks-API um den Ref-Hook ergänzt. Die `createRef`-Methode erzeugt eine ungebundene Referenz, der Sie ein konkretes Element zuweisen können. Dies geschieht über die `ref`-Attribut eines Elements. Zur Demonstration des Umgangs mit Refs sowie mit Uncontrolled Components in React implementieren wir im Folgenden ein einfaches Anmeldeformular, das wir im weiteren Verlauf dieses Buchs in die Workflows der Applikation integrieren werden. Da die Applikation nun zunehmend wächst, kümmern wir uns zunächst um eine bessere Strukturierung des Quellcodes. Bisher liegen alle für die Applikation relevanten Dateien direkt im *src*-Verzeichnis. Diese Lösung funktioniert allerdings nur für sehr kleine Applikationen, da durch eine wachsende Zahl die Übersicht schnell verlorengeht. Zur Strukturierung Ihrer Applikation können Sie verschiedene Ansätze verfolgen: Entweder strukturieren Sie nach dem Typ der einzelnen Elemente, oder Sie gruppieren den Code Ihrer Applikation nach Fachlichkeit.

**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 Hilfsdateien wie die *selectRandomProperty.ts*.

**Fachliche Gruppierung**

Bei der fachlichen Gruppierung bildet die Verzeichnisstruktur die einzelnen fachlichen Module der Applikation ab. Für die Beispielapplikation bedeutet dieser Ansatz, dass Sie beispielsweise ein Verzeichnis *game* erzeugen, das die für das Spiel benötigten Dateien enthält. Das Anmeldeformular könnten Sie im Verzeichnis *Login* platzieren und die Strukturen zur Verwaltung der Applikation in einem *admin*-Verzeichnis.

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 9.1 sehen Sie die Implementierung des Formulars als Klassenkomponente.

```
import React from 'react';

export default class Login extends React.Component {
  username = React.createRef<HTMLInputElement>();
  password = React.createRef<HTMLInputElement>();

  render() {
    return (
      <form>
        <div>
          <label htmlFor="">Benutzername:</label>
          <input type="text" id="username" ref={this.username} />
        </div>
        <div>
          <label htmlFor="">Passwort:</label>
          <input type="password" id="password" ref={this.password} />
        </div>
        <button type="submit">anmelden</button>
      </form>
    );
  }
}
```

**Listing 9.1** Basisformular mit Uncontrolled Components (»src/login/Login.tsx«)

Die Basis des Anmeldeformulars bildet die JSX-Struktur in der `render`-Methode. 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 wird ein `submit`-Button verwendet. 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 werden mithilfe der `createRef`-Funktion von React erzeugt. Diese Operation findet außerhalb der `render`-Methode statt, da Sie ansonsten bei jedem Render-Vorgang eine neue Referenz erzeugen würden. Bei der Verwendung von TypeScript handelt es sich bei `createRef` um eine generische Funktion, 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 können Sie die `Login`-Komponente bereits testweise in die `App`-Komponente integrieren. In Listing 9.2 sehen Sie den Quellcode der `render`-Methode der `App`-Komponente.

```
render() {
  return (
    <DarkMode.Provider value={this.state.darkMode}>
      <Login />
      <button onClick={this.toggleDarkMode}>Toggle Dark Mode</button>

      <Game title="Supertrumpf" />
    </DarkMode.Provider>
  );
}
```

**Listing 9.2** Einbindung der »Login«-Komponente (»src/App.tsx«)

### Autofocus eines Eingabefelds mit Refs

Mit Refs lassen sich Features wie beispielsweise ein Autofocus eines Feldes umsetzen. 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. In der `componentDidMount`-Methode der `Login`-Komponente führen Sie die `focus`-Methode aus. Diese steht ihnen innerhalb der `current`-Eigenschaft des `Ref`-Objekts zur Verfügung. Listing 9.3 enthält die Anpassung an der `Login`-Komponente.

```
import React from 'react';

export default class Login extends React.Component {
  username = React.createRef<HTMLInputElement>();
  password = React.createRef<HTMLInputElement>();

  componentDidMount() {
    this.username.current!.focus();
  }

  render() {
    return (
      <form>…</form>
    );
  }
}
```

**Listing 9.3** 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 beispielsweise `undefined` sein könnte.

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 gerade im Zusammenhang mit Formularen eine ganze Reihe weiterer Event-Handler. Im Fall des Login-Formulars reagieren Sie auf das `submit`-Event, das Sie entweder durch Betätigen der ↵-Taste oder 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 } from 'react';

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

export default class Login extends React.Component<Props> {
  username = React.createRef<HTMLInputElement>();
  password = React.createRef<HTMLInputElement>();

  componentDidMount() {
    this.username.current!.focus();
  }

  handleSubmit = async (event: FormEvent<HTMLFormElement>) => {
    event.preventDefault();
```

```
      this.props.onLogin(
        this.username.current!.value,
        this.password.current!.value
      );
    };

    render() {
      return (
        <form onSubmit={this.handleSubmit}>
          …
        </form>
      );
    }
}
```

**Listing 9.4** Behandlung des »submit«-Events in der »Login«-Komponente (»src/login/Login.tsx«)

Wie Sie in Listing 9.4 sehen, wird der `handleSubmit`-Methode automatisch die Objektrepräsentation des 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 Eventobjekts 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. Nachdem eine Komponente lediglich für einen bestimmten Aspekt verantwortlich ist, kümmert sich die `Login`-Komponente auch nur um die Darstellung des Anmeldeformulars und das zugehörige Event-Handling, nicht aber das Senden der Anmeldedaten zum Server. Diese Aufgabe übernimmt die Funktion `onLogin`, die über die Props in die Komponente hineingereicht wird.

Damit Sie die Implementierung der `Login`-Komponente auch testen können, binden Sie sie in die `App`-Komponente ein. In Listing 9.5 finden Sie eine einfache Implementierung für die Anmeldung.

```
import React from 'react';
import update from 'immutability-helper';

import './App.css';
import Game from './game/Game';
import DarkMode from './game/DarkMode';
import axios from 'axios';
import Login from './login/Login';
```





```
interface State {
  darkMode: boolean;
  loggedIn: boolean;
}

export default class App extends React.Component<{}, State> {
  state = {
    darkMode: false,
    loggedIn: false,
  };

  toggleDarkMode = () => {…};

  handleLogin = async (username: string, password: string) => {
    const result = await axios.post('http://localhost:3001/login', {
      username,
      password,
    });
    if (result) {
      this.setState(prevState =>
        update(prevState, { loggedIn: { $set: true } })
      );
    }
  };

  render() {
    return (
      <DarkMode.Provider value={this.state.darkMode}>
        {this.state.loggedIn && (
          <>
            <button onClick={this.toggleDarkMode}>Toggle Dark Mode</button>
            <Game title="Supertrumpf" />
          </>
        )}
        {!this.state.loggedIn && <Login onLogin={this.handleLogin} />}
      </DarkMode.Provider>
    );
  }
}
```

**Listing 9.5** Einbindung der »Login«-Komponente

Zur Unterstützung des Anmeldeprozesses müssen Sie den State der `App`-Komponente um die `loggedIn`-Eigenschaft erweitern. Diese weist standardmäßig den Wert `false` auf und sorgt dafür, dass im Template die `Login`-Komponente gerendert wird. Die `handleLogin`-Methode, die Sie an die `Login`-Komponente durch die `onLogin`-Prop übergeben, sorgt dafür, dass die Anmeldeinformationen an den Server gesendet werden und die `state`-Eigenschaft bei einer erfolgreichen Anmeldung auf den Wert `true` gesetzt wird. Dadurch wird das eigentliche Spiel dargestellt.

Die Serverimplementierung muss für den Login einen Endpunkt */login* zur Verfügung stellen und dort die Kombination aus Benutzernamen und Passwort überprüfen. Ist die Kombination korrekt, wird der Wert `true` zurückgegeben, andernfalls `false`. Diese Art der Umsetzung ist relativ einfach und wird üblicherweise nicht im Produktivbetrieb verwendet. In Kapitel 13, »Umgang mit Asynchronität und Seiteneffekten in Redux«, lernen Sie mit der tokenbasierten Authentifizierung eine bessere Variante kennen.

Mit der aktuellen Implementierung decken Sie jedoch nur den Erfolgsfall ab. Tritt ein Fehler auf, erfährt der Benutzer 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 dem Benutzer 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 den Benutzer nicht mehr unmittelbar bei der Eingabe, aber noch vor einer potenziell zeitintensiven Serveranfrage. In diesem Fall erfolgt die Prüfung im `submit`-Handler, und abhängig vom Ergebnis wird entweder eine Fehlermeldung angezeigt oder die Daten an den Server übermittelt.

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. Außerdem sollten Sie Benutzereingaben zusätzlich zur clientseitigen Validierung in jedem Fall auch serverseitig prüfen, da eine clientseitige Überprüfung vom Benutzer umgangen werden kann.

Im Beispiel prüfen Sie nun, ob der Benutzer sowohl einen Benutzernamen als auch ein Passwort angegeben hat. Außerdem müssen Sie mit einem Fehlschlag der Anmeldung umgehen können. Zur Visualisierung von Fehlern verfügt die Komponente über einen `error`-State. Dieser weist im Erfolgsfall eine leere Zeichenkette auf; erst wenn ein Fehler auftritt, wird die entsprechende Fehlermeldung gesetzt. Zusätzlich kann von der Elternkomponente eine `error`-Prop gesetzt werden, um anzuzeigen, dass die Anmeldung fehlgeschlagen ist. Die angepasste Version der `Login`-Komponente sehen Sie in Listing 9.6.



```
import React, { FormEvent } from 'react';
import update from 'immutability-helper';

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

interface State {
  error: string;
}

export default class Login extends React.Component<Props, State> {
  username = React.createRef<HTMLInputElement>();
  password = React.createRef<HTMLInputElement>();

  state = {
    error: '',
  };

  componentDidMount() {
    this.username.current!.focus();
  }

  handleSubmit = async (event: FormEvent<HTMLFormElement>) => {
    event.preventDefault();
    let error = 'Bitte Benutzernamen und Passwort angeben.';
    if (this.username.current!.value && this.username.current!.value) {
      error = '';
      this.props.onLogin(
```

```
        this.username.current!.value,
        this.password.current!.value
      );
    }
    this.setState(prevState => update(prevState, { error: { $set: error } }));
};

render() {
  return (
    <form onSubmit={this.handleSubmit} className="login">
      {(this.props.error !== '' || this.state.error !== '') && (
        <div className="error">
          {this.props.error} {this.state.error}
        </div>
      )}
      <div>
        <label htmlFor="">Benutzername:</label>
        <input type="text" id="username" ref={this.username} />
      </div>
      <div>
        <label htmlFor="">Passwort:</label>
        <input type="password" id="password" ref={this.password} />
      </div>
      <button type="submit">anmelden</button>
    </form>
  );
 }
}
```

**Listing 9.6** Anzeige von Fehlermeldungen

Damit ein Fehler bei der Anmeldung korrekt an die `Login`-Komponente übergeben wird, müssen Sie die `App`-Komponente, wie in Listing 9.7 zu sehen ist, noch etwas anpassen.

```
…
interface State {
  darkMode: boolean;
  loggedIn: boolean;
  error: string;
}

export default class App extends React.Component<{}, State> {
  state = {
```





```
    darkMode: false,
    loggedIn: false,
    error: '',
  };

  toggleDarkMode = () => {…};

  handleLogin = async (username: string, password: string) => {
    const result = await axios.post('http://localhost:3001/login', {
      username,
      password,
    });
    let loggedIn = false;
    let error = 'Anmeldung fehlgeschlagen';
    if (result.data === true) {
      loggedIn = true;
      error = '';
    }
    this.setState(prevState =>
      update(prevState, {
        loggedIn: { $set: loggedIn },
        error: { $set: error },
      })
    );
  };

  render() {
    return (
      <DarkMode.Provider value={this.state.darkMode}>
        {this.state.loggedIn && (
          <>
            <button onClick={this.toggleDarkMode}>Toggle Dark Mode</button>
            <Game title="Supertrumpf" />
          </>
        )}
        {!this.state.loggedIn && (
          <Login onLogin={this.handleLogin} error={this.state.error} />
        )}
      </DarkMode.Provider>
    );
  }
}
```

**Listing 9.7** Anpassungen an der »App«-Komponente

Die App-Komponente verfügt nun ebenfalls über einen error-State, der entweder eine leere Zeichenkette im Erfolgsfall oder eine Fehlermeldung bei einer fehlgeschlagenen Anmeldung enthält. Diese Information wird an die Login-Komponente übergeben.

Bei dieser Version des Anmeldeprozesses handelt es sich lediglich um einen Zwischenschritt; in Kapitel 12, wenn 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 potenzielle Benutzer Ihrer Applikation nicht von der Ansicht abgeschreckt werden. Das Login-Formular weist bereits den Klassennamen login auf, sodass Sie problemlos ein einfaches Stylesheet anwenden können. Zu diesem Zweck legen Sie die Datei *login.scss* im *login*-Verzeichnis an, in dem sich auch die Login-Komponente befindet. Den Quellcode des Stylesheets finden Sie in Listing 9.8.

```
.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: 120px;
    display: inline-block;
  }
  .error {
    color: red;
  }
}
```

**Listing 9.8** 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 wie in Abbildung 9.1 aussehen.

Refs können nicht nur, wie im vorangegangenen Beispiel gezeigt, in Klassenkomponenten, sondern dank der Hook-API auch recht komfortabel in Funktionskomponenten verwendet werden.





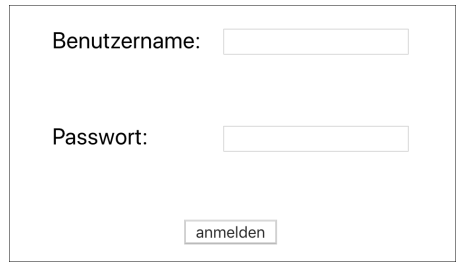


**Abbildung 9.1** Anmeldeformular

### Verwendung von Refs in Funktionskomponenten



Mit der Hook-API wurde neben den drei Basis-Hooks auch der Ref-Hook eingeführt. Dieser ermöglicht die Verwendung von Refs in Funktionskomponenten.

```
import React, { FormEvent, useRef, useState, useEffect, useCallback,} from 'react';
import update from 'immutability-helper';

import './Login.scss';

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

export default function Login({ error, onLogin }: Props) {
  const username = useRef<HTMLInputElement>(null);
  const password = useRef<HTMLInputElement>(null);

  const [localError, setLocalError] = useState('');

  useEffect(() => {
    username.current!.focus();
  });

  const handleSubmit = useCallback((event: FormEvent<HTMLFormElement>) => {
    event.preventDefault();
    let error = 'Bitte Benutzernamen und Passwort angeben.';
    if (username.current!.value && username.current!.value) {
      error = '';
      onLogin(username.current!.value, password.current!.value);
    }
    setLocalError(prevState => update(prevState, { $set: error }));
  }, []);
```

```
  return (
    <form onSubmit={handleSubmit} className="login">
      {(error !== '' || localError !== '') && (
        <div className="error">
          {error} {localError}
        </div>
      )}
      <div>
        <label htmlFor="">Benutzername:</label>
        <input type="text" id="username" ref={username!} />
      </div>
      <div>
        <label htmlFor="">Passwort:</label>
        <input type="password" id="password" ref={password} />
      </div>
      <button type="submit">anmelden</button>
    </form>
  );
}
```

**Listing 9.9** »Login«-Komponente als Funktionskomponente (»src/login/Login.tsx«)

Listing 9.9 enthält den Quellcode der `Login`-Komponente, die als Funktionskomponente umgesetzt wurde. Mit einer Kombination aus State-, Effect-, Callback- und Ref-Hook erreichen Sie die gleiche Funktionalität wie mit der Klassenkomponente in etwas weniger Codezeilen. Außerdem müssen Sie nicht weiter mit dem Kontext der Klassenkomponente arbeiten, was den Code etwas leichter verständlich macht.

Die `useRef`-Funktion ist als generische Funktion implementiert und akzeptiert den Typ des Elements, auf das die Ref verweisen soll. Der Wert `null`, den Sie beim Aufruf der Funktion übergeben, stellt den Initialwert der Ref dar.

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

## 9.2 Controlled Components

*Controlled Components* kommen in der Beispielapplikation für die Verwaltung der Daten des Spiels zum Einsatz. Zu diesem Zweck implementieren Sie ein Formular, mit dessen Hilfe Sie neue Spielkarten anlegen und bereits 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. Diesen internen Komponenten-State können Sie sowohl mit





Klassenkomponenten als auch mit Funktionskomponenten abbilden. Im ersten Schritt implementieren Sie eine Formularkomponente; diese binden Sie in einer `Admin`-Komponente ein, die wiederum direkt in die `App`-Komponente integriert wird. Für den Entwicklungsbetrieb deaktivieren Sie zunächst die Anmeldemaske und das eigentliche Spiel. Die einzelnen Teile der Applikation integrieren Sie später, wenn wir uns dem React Router und der Navigation innerhalb der Applikation widmen. Als Vorbereitung erzeugen Sie zunächst ein Verzeichnis mit dem Namen *admin*, in dem Sie die Komponenten für die Verwaltung gruppieren. In diesem Verzeichnis erstellen Sie eine Datei mit dem Namen *Form.tsx*. Den Kern der Formularimplementierung bildet die `Form`-Komponente, die Sie als Funktionskomponente umsetzen.



```
import React from 'react';
import Animal from '../game/Animal';
import useCardAdmin from './useCardAdmin';
import { Label, Row, Form as StyledForm} from './Form.styles';

interface Props {
  onSubmit: (animal: Animal) => void;
  animal?: Animal;
}

export default function Form({ onSubmit, animal: initialAnimal }: Props) {
  const [animal, changeProperty] = useCardAdmin(initialAnimal);
  return (
    <StyledForm
      onSubmit={e => {
        e.preventDefault();
        onSubmit(animal);
      }}
    >
      <Row>
        <Label htmlFor="name">Name:</Label>
        <input
          type="text"
          id="name"
          value={animal.name}
          onChange={changeProperty}
        />
      </Row>
      {Object.keys(Animal.properties).map(property => {
        let value = (animal as any)[property];
        value = value === 0 ? '' : value;
        return (
```

```
            <Row key={property}>
              <Label htmlFor={property}>
                {Animal.properties[property].label}:
              </Label>
              <input
                type="text"
                id={property}
                value={value}
                onChange={changeProperty}
              />
            </Row>
          );
        })}
        <div>
          <button type="submit">speichern</button>
        </div>
      </StyledForm >
    );
}
```

**Listing 9.10** Aufbau der Formularkomponente (»src/admin/Form.tsx«)

Das Formular wird mithilfe von Styled Components gestaltet. Den zugehörigen Quellcode speichern Sie im selben Verzeichnis wie die `Form`-Komponente in der Datei *Form.styles.ts*. Listing 9.11 enthält den Quellcode.

```
import styled, { css } from 'styled-components';

export const Form = styled.form`
  display: flex;
  flex-direction: column;
  align-items: flex-start;
  div {
    padding: 3px;
  }
`;

export const Label = styled.label`
  width: 120px;
  display: inline-block;
`;

export const Row = styled.div`
  &:nth-child(2n) {
```





```
    background-color: #ccc;
  }
`;
```

**Listing 9.11** Styling der »Form«-Komponente (»src/admin/Form.styles.ts«)

Doch nun zurück zur `Form`-Komponente. Da die Komponente Datensätze sowohl erzeugen als auch modifizieren können soll, definieren Sie eine optionale Prop mit dem Namen `animal`, die einen bestehenden Datensatz enthält, mit dessen Daten das Formular vorbefüllt wird. Als zweite verpflichtende Prop übergeben Sie eine Funktion zur Behandlung des `submit`-Events. Bei der Verknüpfung der Funktion mit dem `submit`-Event müssen Sie lediglich dafür sorgen, dass die Standardfunktionalität des Browsers mittels der `preventDefault`-Methode unterdrückt wird.

Das Formular selbst besteht aus einer Reihe von Eingabefeldern. Diese werden über eine Zwei-Wege-Verknüpfung mit der Komponente verbunden. Die `value`-Eigenschaft wird mit dem State-Wert der Komponente verknüpft. Über die `onChange`-Methode des Felds kann der State-Wert verändert werden. Sie können die Formularfelder einzeln auflisten und pro Feld einen State-Wert definieren sowie einen `onChange`-Handler. Um den Quellcode etwas kompakter zu halten, können Sie jedoch auch die `properties`-Eigenschaft der `Animal`-Klasse verwenden. Diese Eigenschaft enthält eine Liste der Werte, die Sie pro Karte verwenden können. Neben diesen Informationen benötigen Sie nur noch ein Eingabefeld für den Namen. Da es sich bei der `Form`-Komponente um eine Funktionskomponente handelt, die über keinen eigenen State verfügt, bilden Sie diesen über einen Custom Hook ab. Den Quellcode dieses Hooks aus Listing 9.12 speichern Sie in der Datei *useCardAdmin.ts*.

```
import { useState, useCallback, ChangeEvent } from 'react';
import update from 'immutability-helper';
import Animal from '../game/Animal';

export default function useCardAdmin(
  initialAnimal: Animal = new Animal('', '', 0, 0, 0, 0, 0)
): [Animal, (event: ChangeEvent<HTMLInputElement>) => void] {
  const [animal, setAnimal] = useState<Animal>(initialAnimal);

  const changeProperty = useCallback(
    (event: ChangeEvent<HTMLInputElement>): void => {
      setAnimal(animalState =>
        update(animalState, {
          [event.currentTarget.id]: { $set: event.currentTarget.value },
        })
      );
    },
```

```
    []
  );

  return [animal, changeProperty];
}
```

**Listing 9.12** Custom Hook für die Formularlogik (»src/admin/useAdminCard.ts«)

Die Hook-Funktion akzeptiert eine zu modifizierende `Animal`-Instanz. Übergeben Sie hier kein Objekt, wird ein neues Objekt erzeugt. Innerhalb des Custom Hooks wird ein neuer State mit der übergebenen Instanz erzeugt. Mit der `setAnimal`-Funktion kann der State aktualisiert werden. Der Custom Hook gibt ein Tupel aus dem State und der `changeProperty`-Funktion zurück, wobei die `Animal`-Instanz die Werte für die einzelnen Formularfelder liefert und die `changeProperty`-Funktion direkt als Handler für das `change`-Event im Formular verwendet werden kann. Als Argument erhält die Funktion die Objektrepräsentation des `change`-Events. Mit dieser Information können Sie in Kombination mit der `setAnimal`-Methode und dem `immutability-helper` den Eigenschaftswert verändern.

Damit Sie das Formular auch testen können, müssen Sie es in Ihre Applikation einbinden. Bei dem Formular handelt es sich um eine reine Presentational Component, sie enthält neben der in einen Custom Hook ausgelagerten Formularlogik keine weitere Logik. Das bedeutet, dass die einbindende Komponente sich um das Speichern der Daten kümmern muss. Für den aktuellen Test reicht es aus, wenn Sie in Listing 9.13 der `onSubmit`-Funktion ein `console.log`-Statement ausführen. In Listing 9.13 finden Sie eine stark vereinfachte Version der `App`-Komponente.

```
import React from 'react';

import Form from './admin/Form';

export default function App() {
  return <Form onSubmit={animal => console.log(animal)} />;
}
```

**Listing 9.13** »App«-Komponente mit Formulareinbindung (»src/App.tsx«)

Werfen Sie nun einen Blick auf Ihre Applikation im Browser, sehen Sie eine Ansicht wie in Abbildung 9.2.

Versuchen Sie jedoch, eine Eingabe zu machen, erhalten Sie auf der Browserkonsole Warnungsmeldungen.

Beim Umgang mit Formularen haben Sie sehr häufig mit Browser-Events wie beispielsweise dem Change- oder Klick-Event zu tun. React weist an dieser Stelle eine





Optimierung auf, die als *Synthetic Events* bezeichnet wird und die Sie kennen sollten, da Sie in einigen Fällen zu Problemen führt.

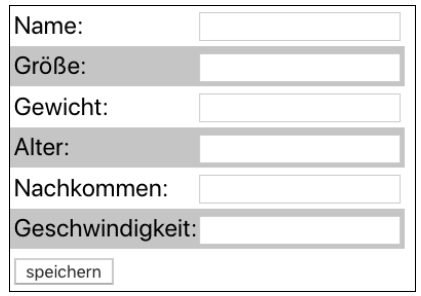


**Abbildung 9.2** Formularansicht



### 9.2.1 Synthetic Events

Die Event-Handler-Funktionen von React erhalten nicht die nativen Eventobjekte des Browsers, sondern Wrapper-Objekte, die sogenannten Synthetic Events. Diese sorgen dafür, dass sich die Events über alle Browser konsistent verhalten. Das Wrappen der Events geschieht jedoch primär aus Performancegründen. Die Events werden von React wiederverwendet und die Eigenschaften zurückgesetzt, sobald der Event-Handler ausgeführt wurde. Für Sie bedeutet das, dass Sie in Operationen, wie beispielsweise beim Setzen des States mit einer Callback-Funktion, nicht mehr direkt auf die Eigenschaften der Eventobjekte zugreifen können, sondern sie zuvor in Variablen speichern müssen. In Listing 9.14 sehen Sie die modifizierte Version des `CardAdmin`-Hooks, mit der diese Warnmeldungen nicht mehr auftreten.

```
import { useState, useCallback, ChangeEvent } from 'react';
import update from 'immutability-helper';
import Animal from '../game/Animal';

export default function useCardAdmin(
  initialAnimal: Animal = new Animal('', '', 0, 0, 0, 0, 0)
): [Animal, (event: ChangeEvent<HTMLInputElement>) => void] {
  const [animal, setAnimal] = useState<Animal>(initialAnimal);

  const changeProperty = useCallback(
    (event: ChangeEvent<HTMLInputElement>): void => {
      const id = event.currentTarget.id;
      let value = event.currentTarget.value;
      setAnimal(animalState =>
        update(animalState, {
          [id]: { $set: value },
        })
```

```
      );
    },
    []
  );

  return [animal, changeProperty];
}
```

**Listing 9.14** Korrekter Umgang mit Synthetic Events (»src/admin/useCardAdmin.ts«)

Für die Verwaltung der Karten des Spiels fehlt mit dem Dateiupload noch ein entscheidendes Feature.

## 9.3 Upload von Dateien

Der Upload von Dateien gestaltet ich etwas anders als die Formularbehandlung, mit der Sie bisher gearbeitet haben. Als Schnittstelle zum Benutzer hin verwenden Sie ein `input`-Element vom Typ `file`. Dieses Element wird nur zum Lesen verwendet, Sie setzen es demnach als Uncontrolled Component. In Listing 9.15 finden Sie den angepassten Quellcode der Formularkomponente.

```
import React from 'react';
import Animal from '../game/Animal';
import useCardAdmin from './useCardAdmin';
import { Label, Row, Form as StyledForm } from './Form.styles';

interface Props {
  onSubmit: (animal: Animal) => void;
  animal?: Animal;
}

export default function Form({ onSubmit, animal: initialAnimal }: Props) {
  const [animal, changeProperty] = useCardAdmin(initialAnimal);
  return (
    <StyledForm
      onSubmit={e => {
        e.preventDefault();
        onSubmit(animal);
      }}
    >
      <Row>
        <Label htmlFor="name">Name:</Label>
        <input type="text" id="name" value={animal.name}
```





```
          onChange={changeProperty} />
      </Row>
      <Row>
        <Label htmlFor="image">Bild:</Label>
        <input type="file" id="image" onChange={changeProperty} />
      </Row>
      {Object.keys(Animal.properties).map(property => {…})}
      <div>
        <button type="submit">speichern</button>
      </div>
    </StyledForm>
  );
}
```

**Listing 9.15** Integration des Fileuploads in das Formular (»src/admin/Form.tsx«)

Neben dieser Änderung müssen Sie den `CardAdmin`-Hook dahingehend anpassen, dass die hochgeladene Datei korrekt behandelt wird. Listing 9.16 zeigt die erforderlichen Änderungen am Quellcode.

```
import { useState, useCallback, ChangeEvent } from 'react';
import update from 'immutability-helper';
import Animal from '../game/Animal';

export default function useCardAdmin(
  initialAnimal: Animal = new Animal('', '', 0, 0, 0, 0, 0)
): [Animal, (event: ChangeEvent<HTMLInputElement>) => void] {
  const [animal, setAnimal] = useState<Animal>(initialAnimal);

  const changeProperty = useCallback(
    (event: ChangeEvent<HTMLInputElement>): void => {
      const id = event.currentTarget.id;
      let value: string | File;
      value = event.currentTarget.value;
      if (id === 'image') {
        value = event.currentTarget.files![0];
      }
      setAnimal(animalState => {
        const newState = update(animalState, {
          [id]: { $set: value },
        });
        return newState;
      });
    },
```

```
    []
  );

  return [animal, changeProperty];
}
```

**Listing 9.16** Unterstützung des Dateiuploads im »CardAdmin«-Hook (»src/admin/useCardAdmin.ts«)

Bei den Texteingabefeldern greifen Sie über `currentTarget.value` auf den Wert zu. Im Gegensatz dazu befindet sich die Referenz auf die hochgeladene Datei im `files`-Array des `currentTarget`-Objekts. Mit diesem Array lässt sich auch ein Multi-File-Upload umsetzen. Da es sich hier um den Upload einer einzelnen Datei handelt, können Sie auf das erste Element des Arrays zugreifen und dieses verwenden. Der eigentliche Dateiupload findet, wie das Absenden des Formulars, nicht direkt in der Formularkomponente statt. Damit entkoppeln Sie die Serverkommunikation von der Formulardarstellung und erhalten die Möglichkeit, die Serverkommunikation zu einem späteren Zeitpunkt mit überschaubarem Aufwand auszutauschen. Damit Sie den Dateiupload auch testen können, passen Sie die `App`-Komponente wie in Listing 9.17 an.

```
import React from 'react';

import Form from './admin/Form';
import axios from 'axios';

export default function App() {
  return (
    <Form
      onSubmit={animal => {
        const data = new FormData();
        data.append('name', animal.name);
        data.append('image', animal.image);
        data.append('size', animal.size.toString());
        data.append('weight', animal.weight.toString());
        data.append('age', animal.age.toString());
        data.append('offspring', animal.offspring.toString());
        data.append('speed', animal.speed.toString());
        axios.post('http://localhost:3001/card', data, {
          headers: {
            'content-type': 'multipart/form-data',
          },
        });
```





```
      }}
    />
  );
}
```

**Listing 9.17** Dateiupload (»src/App.tsx«)

Die Formulardaten werden hier in eine Instanz der `FormData`-Klasse gekapselt. Die Daten können mithilfe der `append`-Methode hinzugefügt werden. Die verfügbaren Typen sind Strings und Files. Das `FormData`-Objekt wird anschließend mithilfe von Axios zum Server gesendet. Wichtig ist an dieser Stelle, dass Sie den `content-type` auf `multipart/form-data` setzen, damit der Server korrekt mit der Anfrage umgehen kann.

Als nächster Schritt steht bei der Formularimplementierung die Validierung der Eingaben an. Diese können Sie, wie beim Beispiel mit den Uncontrolled Components, selbst übernehmen. In diesem Fall ist ein sofortiges Feedback an den Benutzer noch einfacher möglich, da der `change`-Handler bei jeder Eingabe ausgeführt wird und Ihnen so die Möglichkeit bietet, zu reagieren. Da React für die Validierung von Formularelementen keine Funktionalität vorsieht, bleibt es Ihnen überlassen, diese vollständig zu implementieren. Dabei müssen Sie auch auf Grenzfälle wie initial leere Felder oder Ähnliches eingehen. Da es sich hierbei um eine Standardaufgabe handelt, sollten Sie in der Regel auf eine etablierte Bibliothek zurückgreifen, die Ihnen die Arbeit abnimmt und auch Grenzfälle entsprechend abdeckt. Mit *Formik* möchte ich Ihnen im folgenden Abschnitt eine solche Bibliothek vorstellen und diese in das bestehende Formular integrieren.

## 9.4 Formularvalidierung mit Formik

Bevor Sie die eigentliche Formularvalidierung integrieren können, sollten Sie zunächst eine Anpassung an der `Animal`-Klasse vornehmen. Diese lässt aktuell bei den meisten Eigenschaften nur Zahlen zu, dies ist jedoch von Nachteil, wenn es um die Darstellung der Werte in einem Formular geht, da in diesem Fall mit der Zahl 0 als Initialwert gearbeitet wird und dieser dann in eine leere Zeichenkette umgewandelt werden muss. Aus diesem Grund lassen Sie als Werte für die numerischen Eigenschaften entweder Zahlen oder leere Zeichenketten zu. Die erforderliche Anpassung zeigt Ihnen Listing 9.18.

```
export default class Animal {
  static properties: { [key: string]: { label: string; unit: string } } = {
    size: { label: 'Größe', unit: 'm' },
    weight: { label: 'Gewicht', unit: 'kg' },
    age: { label: 'Alter', unit: 'Jahre' },
```

```
    offspring: { label: 'Nachkommen', unit: '' },
    speed: { label: 'Geschwindigkeit', unit: 'km/h' },
  };

  public id?: number;

  constructor(
    public name: string,
    public image: string,
    public size: number | '',
    public weight: number | '',
    public age: number | '',
    public offspring: number | '',
    public speed: number | ''
  ) {}
}
```

**Listing 9.18** Leerwerte in der »Animal«-Klasse (»src/game/Animal.ts«)

Mit dieser Änderung können Sie sich nun dem eigentlichen Formular widmen.

### 9.4.1 Erzeugung eines Validierungsschemas

Formik erlaubt zwei verschiedene Varianten der Validierung. Sie können als erste Variante selbst eine Funktion schreiben, die die aktuellen Formularwerte als Eingabe erhält und ein Objekt mit Fehlern zurückgibt. Dabei nutzen Sie die Namen der Formularelemente als Schlüssel und die Fehlermeldungen als Werte. Deutlich komfortabler, als die Validierungslogik selbst zu schreiben, ist die zweite Variante: die Verwendung einer zusätzlichen Bibliothek mit dem Namen `yup`. Diese Bibliothek können Sie zur Erzeugung eines Validierungsschemas verwenden. Yup ist als NPM-Paket verfügbar und kann mit dem Kommando `yarn add yup` installiert werden. Zusätzlich sollten Sie, wenn Sie TypeScript verwenden, die Typdefinition von Yup mit dem Kommando `yarn add -D @types/yup` installieren. In Listing 9.19 finden Sie den Quellcode für die Definition des Schemas für unser Formular.

```
import * as Yup from 'yup';

Yup.setLocale({
  mixed: {
    required: 'Pflichtfeld, bitte einen Wert eingeben',
    notType: 'Bitte nur Zahlen eingeben',
  },
  number: {
```





```
    moreThan: 'Bitte nur Zahlen größer als 0 eingeben',
  },
});

const validationSchema = Yup.object().shape({
  name: Yup.string().required(),
  image: Yup.mixed().required(),
  size: Yup.number()
    .moreThan(0)
    .required(),
  weight: Yup.number()
    .moreThan(0)
    .required(),
  age: Yup.number()
    .moreThan(0)
    .required(),
  offspring: Yup.number()
    .moreThan(0)
    .required(),
  speed: Yup.number()
    .moreThan(0)
    .required(),
});

export default validationSchema;
```

**Listing 9.19** Validierungsschema für das Formik-Formular (»src/admin/validationSchema.ts«)

Zunächst nutzen Sie die `setLocale`-Methode von Yup, um die Übersetzungen für die Fehlermeldungen zu definieren. Die Eigenschaften `mixed` und `number` geben die Schemas an, für die die jeweiligen Übersetzungen gelten. Unterhalb von `mixed` finden Sie beispielsweise die Eigenschaft `required`, die die Fehlermeldung für eine fehlgeschlagene Prüfung eines Pflichtfeldes enthält.

Die eigentliche Schemadefinition wird durch `Yup.object().shape` eingeleitet. Dieser Methode übergeben Sie ein Objekt, in dem Sie die Validierungsregeln festlegen. Sie können an dieser Stelle für alle Formularfelder Regeln definieren. Diese setzen sich aus einem Basisschema wie beispielsweise `Yup.string` oder `Yup.number` und weiteren Regeln zusammen. Das `mixed`-Schema bildet die Basisklasse für die weiteren Schemas. Es stellt Ihnen beispielsweise die Regel für Pflichtfelder zur Verfügung und vererbt sie an die anderen Schemas. Für die Überprüfung des Dateiuploads kommt beispielsweise das `mixed`-Schema zum Einsatz, da der Dateiupload weder dem Typ

`String` noch dem Typ `Number` entspricht. Beim Eingabefeld für den Namen handelt es sich um eine Zeichenkette und ein Pflichtfeld. Die Größe ist wiederum vom Typ `Number`; sie muss als positive Zahl angegeben und darf nicht leer gelassen werden. Diese Regeln gelten auch für die übrigen Eigenschaften. Damit die Formularkomponente übersichtlich bleibt, können Sie das Validierungsschema in eine eigene Datei mit dem Namen *validationSchema.ts* im *admin*-Verzeichnis speichern.

### 9.4.2 Styling der Formularkomponenten

Formik arbeitet als Wrapper um das eigentliche Formular und reichert es um zusätzliche Funktionalität, die für die Validierung benötigt wird, an. So können Sie beispielsweise feststellen, ob ein Benutzer bereits Eingaben in einem Formular getätigt hat und ob es sich dabei um fehlerhafte Eingaben handelt. Außerdem haben Sie Zugriff auf die jeweiligen Fehlermeldungen. Bevor Sie jedoch mit Formik arbeiten können, müssen Sie die Bibliothek zunächst installieren. Dies geschieht mit dem Kommando `yarn add formik`. Da Formik selbst in TypeScript implementiert ist, bringt es auch seine eigenen Typdefinitionen bereits mit, sodass Sie sie nicht erst manuell installieren müssen.

Formik können Sie aktuell entweder als Higher-Order Component oder als Implementierung mit Render Props verwenden. Die zweite Variante ist meiner Meinung nach etwas angenehmer zu verwenden, sodass wir im Folgenden auf diese Art setzen. Die Wrapper-Komponente von Formik verwaltet den Zustand des Formulars in ihrem lokalen State und erlaubt den Zugriff darauf. Die einzelnen Formularelemente können dann wie gewöhnliche Controlled Components behandelt werden. Der einzige Unterschied besteht darin, dass sowohl der `change`-Handler als auch der Wert des Elements von Formik stammen. Neben diesen beiden Strukturen arbeitet Formik außerdem mit dem `blur`-Event eines Formularelements, um festzustellen, wann der Benutzer das Feld verlässt. Das bedeutet, dass Sie bei jedem Formularelement den Wert, den `change`- und den `blur`-Handler setzen müssen. Um Ihnen diese unnötige Arbeit abzunehmen, stellt Ihnen Formik eine Reihe von Komponenten zur Verfügung, die diese Verbindung automatisch aufweisen. Die `Form`-Komponente kapselt das HTML-Formular, die `Field`-Komponente die einzelnen Eingabefelder, und die `ErrorMessage`-Komponente sorgt schließlich für die korrekte Anzeige der Fehlermeldung. In der bisherigen Implementierung des Formulars haben Sie mit Styled Components gearbeitet. Diese Bibliothek ermöglicht es Ihnen, nicht nur Elemente zu stylen, sondern auch Komponenten. Um das Formular und die Eingabefelder entsprechend zu stylen, wenden Sie in der Datei *Form.styles.ts* im *admin*-Verzeichnis die von Styled Components exportierte Funktion auf die `Formik`-Komponenten an und definieren entsprechende Styles. Wie das im Detail funktioniert, sehen Sie in Listing 9.20.





```
import styled, { css } from 'styled-components';
import { Form as FormikForm, Field as FormikField } from 'formik';

export const Form = styled(FormikForm)`
  display: flex;
  flex-direction: column;
  align-items: flex-start;
  div {
    padding: 3px;
  }
`;

export const Field = styled(FormikField)`
  &.error {
    border: 1px solid red;
  }
`;

export const Label = styled.label`…`;

export const Row = styled.div`…`;

export const Error = styled.div`…`;
```

**Listing 9.20** Styling der Formik-Komponenten (»src/admin/Form.styles.ts«)

Aus dem Formik-Paket importieren Sie die beiden Komponenten `Form` und `Field` und präfixen diese mit dem Wort `Formik`, damit Sie die beiden Namen für den Export verwenden können. Die Styles für das Formular können Sie aus der ursprünglichen Version übernehmen. Bei den einzelnen Feldern sorgen Sie dafür, dass sie, wenn sie mit der `error`-Klasse ausgezeichnet werden, mit einem roten Rahmen versehen werden.

### 9.4.3 Aufbau eines Formulars mit Formik

Für das eigentliche Formular können Sie den überwiegenden Teil der bestehenden Strukturen wiederverwenden. Da Formik den Zustand des Formulars selbst verwaltet und die Erzeugung der Controlled Components für Sie übernimmt, benötigen Sie den `CardAdmin`-Hook nicht mehr. In Listing 9.21 sehen Sie die gesamte Struktur des Formulars mit den gestylten Formik-Komponenten.

```
import React from 'react';
import Animal from '../game/Animal';
import { Label, Row, Form, Error, Field } from './Form.styles';
```

```
import { Formik, ErrorMessage } from 'formik';
import validationSchema from './validationSchema';

interface Props {
  onSubmit: (animal: Animal) => void;
  animal?: Animal;
}

export default function From({
  onSubmit,
  animal = new Animal('', '', '', '', '', '', ''),
}: Props) {
  return (
    <Formik
      initialValues={animal}
      validationSchema={validationSchema}
      onSubmit={(e, actions) => {
        onSubmit(e!);
        actions.setSubmitting(false);
      }}
    >
      {({ isSubmitting, errors, setFieldValue }) => (
        <Form>
          <Row>
            <Label htmlFor="name">Name:</Label>
            <Field
              id="name"
              type="text"
              name="name"
              className={errors.name && 'error'}
            />
            <ErrorMessage name="name" component={Error} />
          </Row>
          <Row>
            <Label htmlFor="image">Bild:</Label>
            <input
              type="file"
              id="image"
              onChange={event => {
                setFieldValue('image', event.currentTarget.files![0]);
              }}
```





```
            />
            <ErrorMessage name="image" component={Error} />
          </Row>
          {(Object.keys(Animal.properties) as (keyof Animal)[]).map(
            property => {
              return (
                <Row key={property}>
                  <Label htmlFor={property}>
                    {Animal.properties[property].label}:
                  </Label>
                  <Field
                    type="text"
                    id={property}
                    name={property}
                    className={errors[property] && 'error'}
                  />
                  <ErrorMessage name={property} component={Error} />
                </Row>
              );
            }
          )}
          <div>
            <button type="submit" disabled={isSubmitting}>
              speichern
            </button>
          </div>
        </Form>
      )}
    </Formik>
  );
}
```

**Listing 9.21** Formularstruktur mit Formik (»src/admin/Form.tsx«)

Für den Fall, dass ein Benutzer einen neuen Datensatz erzeugen möchte und keine `Animal`-Instanz als Prop übergeben wird, erzeugen Sie eine neue Instanz mit leeren Eigenschaftswerten. An die Stelle des `CardAdmin`-Hooks tritt die `Formik`-Komponente. Dieser übergeben Sie mithilfe der `initialValues`-Prop die `Animal`-Instanz. Aus dieser werden die initialen Formularwerte ausgelesen. Enthält diese Instanz die `id`-Eigenschaft, wird diese beibehalten und beim Absenden des Formulars übermittelt, sodass Sie zwischen neuen und bestehenden Datensätzen unterscheiden können. Mit der `validationSchema`-Eigenschaft übergeben Sie das zuvor definierte Validierungsschema

an Formik. Alternativ dazu können Sie über die `validate`-Prop die bereits erwähnte Validierungsfunktion übergeben, mit der Sie sich selbst um die Validierung kümmern. Die `onSubmit`-Prop enthält schließlich den `submit`-Handler, der ausgeführt wird, falls die Validierung keinen Fehler feststellen konnte.

Im Beispiel verwenden Sie Formik in seiner Render-Props-Implementierung. Die Alternative hierzu besteht aus der `withFormik`-Higher-Order-Component. Im Beispiel definieren Sie als Kindknoten der Formik-Komponente eine Funktion, die für den Aufbau der Formularstruktur zuständig ist. Diese Funktion hat Zugriff auf eine Reihe von Eigenschaften wie beispielsweise `isSubmitting`, mit der Sie feststellen können, ob das Formular aktuell abgesendet ist und verarbeitet wird, das `errors`-Objekt, das die einzelnen Fehlermeldungen für die Formularfelder enthält, oder die `setFieldValue`-Funktion, mit der Sie den Formik-State manuell verändern können.

Die `Form`-Komponente von Formik sorgt dafür, dass das umgebende HTML-Formular gerendert wird. Diese Komponente benötigt keine zusätzlichen Props, da sich Formik hierum selbst kümmert. Anhand der `name`-Eigenschaft und der zugehörigen Formularelemente lässt sich gut sehen, wie Sie die einzelnen Teile des Formulars aufbauen können. Die Sektion ist für das Styling in die `Row`-Komponente eingeschlossen, diese wurde aus der Basisimplementierung des Formulars unverändert übernommen. Das Gleiche gilt für das Label. Die `Field`-Komponente sorgt dafür, dass das `input`-Element in das Formular eingefügt wird. Das Verhalten dieser Komponenten können Sie mit einer Reihe von Props beeinflussen. Aber auch hier können Sie die bestehenden Props unverändert übernehmen. Ein Unterschied besteht in der `className`-Prop: Sie sorgt dafür, dass das `input`-Element mit einem roten Rahmen versehen wird, falls ein Fehler bei der Validierung festgestellt wurde.

Die `ErrorMessage`-Komponente ist schließlich für die Darstellung der Fehlermeldung verantwortlich. Mit der `name`-Eigenschaft geben Sie an, für welches Formularelement die Fehlermeldung angezeigt werden soll. Die `component`-Eigenschaft definiert, welche Komponente hierfür verwendet werden soll. Sie können hier entweder Elemente wie `div` oder `p` verwenden oder wie im Beispiel eigene Komponenten nutzen. Die Standardkomponenten geben Sie als Zeichenketten an, bei eigenen Komponenten übergeben Sie eine Referenz auf die Komponente.

Eine weitere Besonderheit weist der `submit`-Button auf: Hier nutzen Sie das `disabled`-Attribut, um den Button während der Validierung zu deaktivieren, um Mehrfachübermittlungen des Formulars zu vermeiden. Dies ist erforderlich, da die Validierung asynchron abläuft und der Benutzer so auch eine visuelle Rückmeldung erhält, falls der Prozess etwas länger dauern sollte.







### Den Dateiupload validieren

Der Validierung des Dateiuploads lässt sich nicht ohne Weiteres mit den Standardmitteln von Formik abdecken. Wie Sie wissen, verhält sich der `input`-Typ `file` in seinem Verhalten anders als die übrigen Eingabefelder, da Sie aus diesem Element nur lesen, aber es nicht schreiben können. Formik bietet Ihnen hierfür eine Lösung und ermöglicht es, den Fileupload als Uncontrolled Component zu verwalten. In diesem Fall müssen Sie jedoch auf die etwas ausführlichere Schreibweise ausweichen. Listing 9.22 wiederholt die Sektion des Formulars, die für den Dateiupload verantwortlich ist.

```
<Row>
  <Label htmlFor="image">Bild:</Label>
  <input
    type="file"
    id="image"
    onChange={event => {
      setFieldValue('image', event.currentTarget.files![0]);
    }}
  />
  <ErrorMessage name="image" component={Error} />
</Row>
```

**Listing 9.22** Dateiupload mit Formik (»src/admin/Form.tsx«)

Wichtig ist an dieser Stelle, dass Sie auf die Standard-Formularelemente ausweichen und diese im Fall einer Uncontrolled Component mit einem `change`-Handler versehen. Über die `setFieldValue`-Funktion können Sie manuell den State der `Formik`-Komponente manipulieren. Zu diesem Zweck geben Sie den Namen der Eigenschaft, in diesem Fall `image`, und den Wert des Feldes an. An diesen gelangen Sie über das Event-Objekt, die `currentTarget`-Eigenschaft und hier das `files`-Array.

Über das Validierungsschema und dort das `mixed`-Schema können Sie auch für einen Dateiupload bestimmte Regeln wie beispielsweise die verpflichtende Angabe festlegen. Die `ErrorMessage`-Komponente verhält sich dann wieder wie gewohnt und zeigt eine Fehlermeldung an, falls der Benutzer keine Datei hochgeladen hat. Mit diesen Anpassungen können Sie Ihre Formularimplementierung testen und sowohl neue Datensätze erzeugen als auch bestehende manipulieren. In Abbildung 9.3 sehen Sie eine Reihe verschiedener Fehlermeldungen, die bei fehlerhaften Eingaben angezeigt werden.

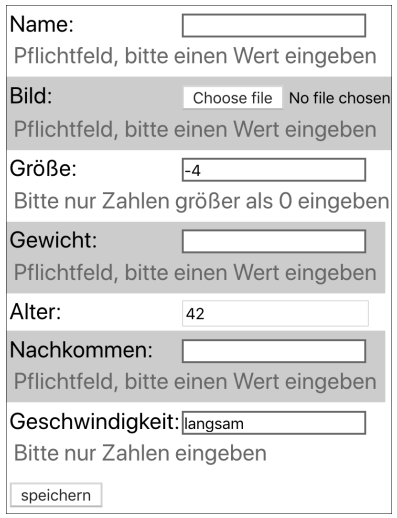


**Abbildung 9.3** Anzeige von Fehlermeldungen

Sobald Sie einen Fehler im Formular korrigieren, werden die Fehlermeldung sowie der rote Rahmen und das entsprechende Feld ausgeblendet, sodass der Benutzer sofortiges Feedback erhält. Das Gleiche passiert, falls der Benutzer aus einer gültigen Eingabe eine fehlerhafte macht, indem er beispielsweise beim Alter noch einen Buchstaben einfügt.

## 9.5 Zusammenfassung

In diesem Kapitel haben Sie erfahren, wie Sie Formulare in Ihre React-Applikation integrieren können, um dem Benutzer die Möglichkeit zu geben, Datensätze anzulegen und zu modifizieren.

- Sie wissen jetzt, was eine Uncontrolled Component ist und dass Sie sich um die Synchronisierung der Werte mit dem Zustand der Komponente selbst kümmern müssen.
- Bei den Controlled Components synchronisieren Sie den Zustand der Komponente mit dem Formular, indem Sie den Wert, der im Formularelement angezeigt wird, direkt aus dem State der Komponente entnehmen.
- Änderungen bei Controlled Components erfolgen über den Change-Handler des Formularelements.
- Neben Standardelementen wie Textfeldern, Checkboxen oder Select-Elementen unterstützt React auch Dateiuploads. Diese Elemente verhalten sich anders als die übrigen Elemente, da sie *read only* sind.





- React trifft keine Annahmen über die Validierung von Formularen. Sie müssen die Validierung entweder selbst implementieren oder greifen auf etablierte Lösungen wie beispielsweise Formik zurück.
- Sie können Formik entweder als Higher-Order Component oder als Render-Props-Implementierung verwenden.
- Mit Yup können Sie komfortabel Validierungsschemas definieren, die Sie für die Validierung einbinden können.
- Formik nimmt Ihnen die manuelle Implementierung von Controlled Components ab und fügt die erforderlichen Strukturen automatisch ein.
- Für Sonderfälle wie Dateiuploads müssen Sie den change-Handler selbst integrieren.



Im nächsten Kapitel erfahren Sie am Beispiel der Material-UI-Bibliothek, wie Sie Komponentenbibliotheken von Drittanbietern in Ihre Applikation einbinden und damit das Styling vereinfachen und neue Features hinzufügen können.

# Auf einen Blick

# Inhalt