2 Schnelldurchgang – React im Überblick

Bevor wir in die Details von React gehen, möchten wir dir in diesem Kapitel die wichtigsten Features von React im Schnelldurchgang an einem sehr einfachen Beispiel zeigen. Das Beispiel kannst du auf der Plattform CodeSandbox, die Online-Editoren für JavaScript anbietet, nachvollziehen, ohne dass du dafür etwas bei dir lokal auf deinem Computer installieren musst. (CodeSandbox selbst ist übrigens auch mit React gebaut.)

Vorbereitung

Um das Beispiel zu entwickeln, kannst du in CodeSandbox ein neues React-Projekt anlegen. Dazu öffnest du *https://codesandbox.io/*, klickst auf »Create Sandbox« und wählst dann unter »Popular Templates« »React« aus. Daraufhin wird ein Projekt angelegt, in dem die Abhängigkeiten auf die React-npm-Module bereits hinzugefügt sind und einige weitere Dateien angelegt sind.

Je nach Einstellung findest du auf der linken Seite einen Explorer mit allen Dateien des Projekts, in der Mitte einen Editor (der sich wie VS Code verwenden lässt!) und rechts die Darstellung deiner Anwendung in einem simulierten Browser. Sobald du den Code deiner Anwendung veränderst, wird die Anwendung übersetzt (z.B. JSX nach ES5) und die Darstellung im simulierten Browser automatisch aktualisiert.

Für diesen Schnelleinstieg wollen wir eine ganz einfache Hello-World-Anwendung bauen, die nur aus einer einzigen Komponente besteht. Komponenten sind das zentrale Element in React und wie wir später noch sehen werden, sind Anwendungen in React nichts weiter als eine Menge von Komponenten.

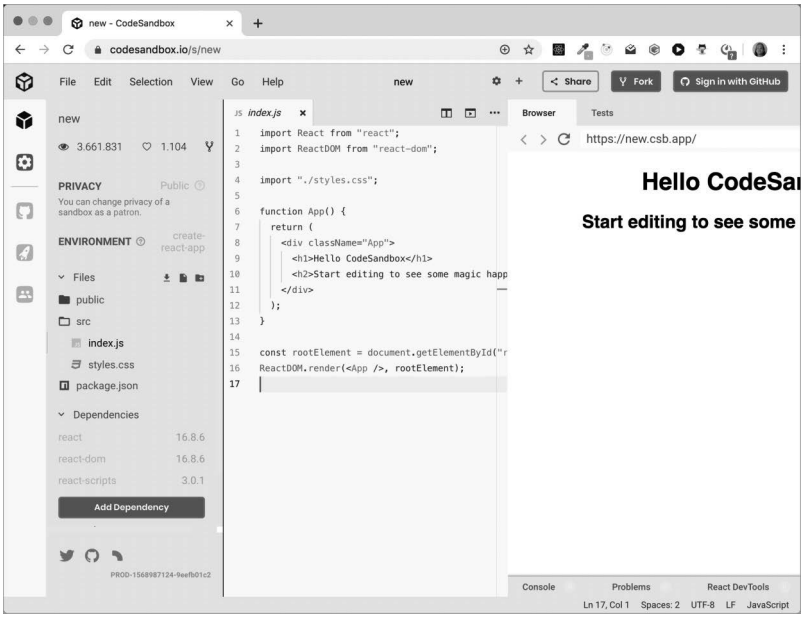

Abb. 2–1
Der Online-Editor CodeSandbox mit Datei-Explorer, Editor und Vorschau der Anwendung

Die Komponente unserer Anwendung, `Greeter`, besteht aus einem Eingabefeld, in dem ein Gruß eingegeben werden kann. Der eingegebene Gruß darf eine bestimmte Länge nicht überschreiten. Die Anzahl der verbleibenden Zeichen steht unterhalb des Eingabefeldes. Außerdem gibt es einen Knopf, der den aktuellen Gruß ausgibt. Aktiv ist der Knopf allerdings nur, wenn der Gruß »gültig« ist, das bedeutet, er enthält mindestens ein Zeichen und die Länge ist kleiner oder gleich der erlaubten Maximallänge.

Beispiel: die Greeter-Komponente

Der erste Schritt unserer Anwendung sieht wie folgt aus, du kannst diesen Code so, wie er da steht, in die `index.js`-Datei deiner Sandbox einsetzen:

```
import React from "react";
export default function Greeter(props) {
  const [greeting, setGreeting] = React.useState("");

  function handleGreetClick() {
    alert(`Hello, ${greeting}`);
  }

  const charsRemaining = props.maxLength - greeting.length;
  const greetingInvalid = greeting.length === 0
                                || charsRemaining < 0;

  return (
    <div>
      Greeting:
```

```
        <input value={greeting}
          onChange={e => setGreeting(e.target.value)} />
        <span>{charsRemaining}</span>

        <button disabled={greetingInvalid}
                  onClick={handleGreetClick}>Greet
        </button>
      </div>
    );
  }
```

Sehen wir uns den Code der Komponente Schritt für Schritt an. Eine Komponente in React ist eine JavaScript-Funktion. In dieser Komponentenfunktion ist alles enthalten, was die Komponente benötigt, um sich darstellen zu können: sowohl die Logik als auch die UI.

Beschreibung der UI mit JSX

Die UI, die unsere Komponente darstellen soll, findest du im `return`-Statement der Komponente. Dort haben wir mit einer HTML-artigen Syntax die Elemente beschrieben, die zur Laufzeit im Browser dargestellt werden sollen. Dieser HTML-artige Code, JSX genannt, wird zur Build-Zeit von einem Compiler (Babel oder TypeScript) in gültiges JavaScript übersetzt. Für uns hat dieser Ansatz den Vorteil, dass wir keine Template-Sprache lernen und verwenden müssen (auch wenn dieses Vorgehen am Anfang sicherlich gewöhnungsbedürftig und vielleicht sogar abschreckend sein mag).

Variablen verwenden

Innerhalb des JSX-Codes können wir auf JavaScript-Variablen und -Funktionen zugreifen. Im Button, der den Gruß anzeigen soll, geben wir damit an, ob der Button aktiv oder inaktiv sein soll. Außerdem geben wir eine Callback-Funktion an, die ausgeführt werden soll, sobald auf den Button geklickt wird. Diese Eigenschaften eines Elementes werden in React Properties genannt. Sowohl `disabled` als auch `onClick` sind somit Properties des Button-Elements, genauso wie `value` und `onChange` Properties des `input`-Felds sind.

Properties

Auch unsere Komponente selbst nimmt ein Property entgegen, mit dem der Verwender der Komponente angeben kann, wie lang der Gruß maximal sein kann (`maxLength`). Properties werden einer Komponente über das erste Funktionsargument, `props`, in einem Objekt übergeben, sodass wir in unserem Beispiel auf das einzige Property der Komponente mit dem Ausdruck `props.maxLength` zugreifen können und die Anzahl verbleibender Zeichen sowie das Enablement des Buttons berechnen können.

Zustand einer Komponente

An dem `input`-Element geben wir mit dem `onChange`-Property eine Callback-Funktion an, die aufgerufen wird, wenn sich das `input`-Feld zur Laufzeit ändert, also beispielsweise, weil ein Zeichen eingegeben wurde. Da React über keine Möglichkeit verfügt, Daten aus der UI automatisch an ein Model zu binden (2-Wege-Databinding), müssen wir uns selber darum kümmern.

State

Das Model einer Komponente wird in React Zustand oder State genannt und wird mit der Funktion `React.useState` erzeugt, der wir als Parameter einen initialen Wert für den Zustand übergeben (in unserem Beispiel einen Leerstring). Diese Funktion liefert uns ein Array mit zwei Einträgen zurück: den aktuellen Wert des Models (in unserem Fall der String mit dem Gruß) und eine *Setter-Funktion* zum Ändern des Models.

Ändern des Zustands löst Rendern aus

Das Besondere an dem Zustand ist: Sobald wir diesen mit der zurückgelieferten Setter-Funktion ändern, wird die gesamte Komponentenfunktion von React erneut aufgerufen und ausgeführt, sodass sie ihre aktualisierte UI zurückliefern kann.

Da wir die Setter-Funktion als Reaktion auf Änderungen im Eingabefeld aufrufen, wird die Komponentenfunktion folglich nach jeder Veränderung im Eingabefeld neu aufgerufen. Bei diesen folgenden Aufrufen der `Greeter`-Funktion liefert `useState` nun jeweils den aktualisierten Wert des Models zurück (und nicht mehr den initialen Wert), den React für uns im Hintergrund verwaltet und aufbewahrt.

Innerhalb unserer Funktion können wir abhängig vom Zustand nun eine komplett neue UI beschreiben. Dabei übernehmen wir zum Beispiel den Wert (aktueller Gruß) in das `input`-Feld (`value`-Property) und setzen das Enablement des Buttons neu. Auch die Anzeige der verbleibenden Zeichen für den Gruß wird aktualisiert.

Deklarative UI

Eine Stärke dieses Vorgehens, der deklarativen Beschreibung der UI und des erneuten Ausführens der kompletten Funktion nach einer einzelnen Änderung, ist es, dass wir nicht in inkonsistente Zustände in der UI kommen können. Alle UI-Elemente, die den Zustand oder Teile davon oder davon abgeleitete Werte (Anzahl verbleibender Zeichen) anzeigen, werden in jedem Fall vollständig aktualisiert.

Nach dem Ausführen der Funktion aktualisiert React anhand der zurückgelieferten UI die Darstellung im Browser. Und obwohl wir immer die komplette UI der Komponente an React zurückgeben, nimmt React nur so wenige Änderungen im DOM wie nötig vor, damit dort die Änderungen so schnell und so effizient wie möglich erfolgen.

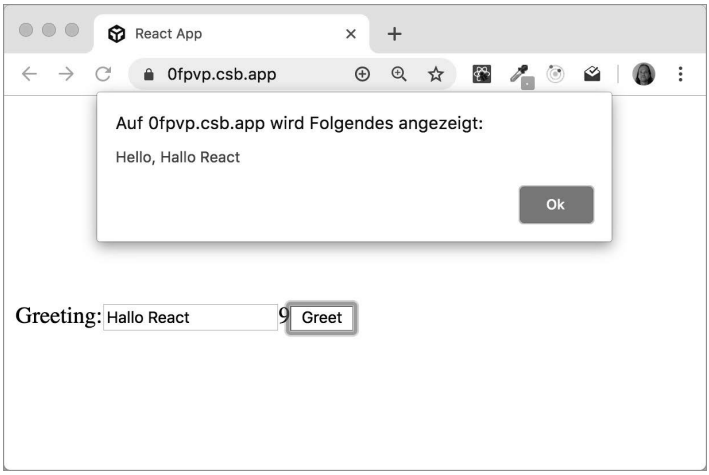

Abb. 2–2
Die Hello-World-Anwendung

Rendern der Anwendung im Browser

ReactDOM.render

Beim Starten unserer Anwendung müssen wir React nun noch angeben, dass unsere Komponente überhaupt dargestellt werden soll. Dazu verwenden wir die Funktion `ReactDOM.render`. Dieser Funktion übergeben wir unsere `Greeter`-Komponente, ebenfalls in der JSX-Notation, wie oben schon bei `button`, `input` etc. gesehen. Dabei geben wir auch das Property `maxLength` an. Außerdem erwartet die Funktion als zweiten Parameter einen DOM-Knoten aus der aktuellen Seite, unterhalb dessen die Komponente dargestellt werden soll. In unserem Beispiel ist dieser DOM-Knoten ein `div`-Element mit der Id `root`, den du in der erzeugten index.html-Datei im `public`-Ordner der Sandbox findest.

Rendern

Zum Rendern der Komponente füge nun bitte den Import für `ReactDOM` und den entsprechenden `render`-Aufruf in der `index.js`-Datei hinzu:

```
import React from "react";
import ReactDOM from "react-dom";

export default function Greeter(props) {
  ...
}

ReactDOM.render(<Greeter maxLength={20} />,
  document.getElementById("root")
);
```

Nun wird die Komponente im Browser von CodeSandbox angezeigt und du kannst sie direkt dort ausprobieren. Eine Sandbox mit der fertigen Beispielanwendung findest du unter *https://codesandbox.io/s/react-buch-schnell-01-0fpvp*.

2.1 Zusammengefasst: Unterschiede zwischen Hooks- und Klassen-API

In diesem Buch verwenden wir fast ausschließlich die Hooks-API, die erstmals Ende 2018 vorgestellt und im Februar 2019 mit React 16.8 in einer stabilen Version veröffentlicht wurde.

Hooks sind Funktionen, mit denen du nahezu alle React-Features in Funktionskomponenten verwenden kannst, die zuvor der Klassen-API vorbehalten waren (zum Beispiel Arbeiten mit State). Falls du schon die Klassen-API, zum Beispiel aus der ersten Auflage dieses Buchs, kennst, geben wir dir hier zur groben Orientierung eine kurze Gegenüberstellung zwischen Klassen- und Hooks-API:

- Zustand/State einer Komponente: Dafür ist der `useState`-Hook (siehe Kapitel 4) zuständig. Das State-Objekt gibt es in Funktionskomponenten nicht, für einen komplexeren Zustand kannst du den `useReducer`-Hook verwenden (siehe Kapitel 6).
- Die Lifecycle-Methoden aus den Klassenkomponenten haben kein direktes Pendant in der Hooks-API. Stattdessen verwendest du den `useEffect`-Hook (siehe Kapitel 7).
- Die Funktionalität von Pure Components sowie der `shouldComponentUpdate`-Methode können durch den `useMemo`-Hook umgesetzt werden (siehe Abschnitt 5.4).
- Als Ersatz für die Verwendung der Provider-Komponente, um auf React Context zuzugreifen, gibt es den `useContext`-Hook (siehe Abschnitt 9.6).
- Higher-Order Components (HOCs) gibt es weiterhin, allerdings gibt es mit der Möglichkeit, eigene Hooks zu bauen, eine API, die möglicherweise einfacher und besser zu verstehen ist. Möglicherweise werden HOCs deswegen künftig seltener eingesetzt werden.
- Die Aufgaben der `connect`-Funktion von Redux können durch die Hooks `useSelector` und `useDispatch` übernommen werden (siehe Kapitel 10).
- Die `withRouter`-HOC aus dem React-Router-Projekt sowie die `render`- und `component`-Properties der `Route`-Komponente werden durch diverse neue Hooks ersetzt (siehe Kapitel 9).

Um die Auswirkungen auf den Code einer React-Anwendung zu sehen, haben wir die fertige Beispielanwendung im Repository auch in einer Variante mit der Klassen-API abgelegt. So kannst du React-Code mit Klassen- und Hooks-API vergleichen.

2.2 Ein React-Projekt beginnen mit Create React App

Um ein ernsthaftes React-Projekt zu beginnen und zu entwickeln ist eine ganze Reihe an Infrastrukturkomponenten erforderlich:

- Neuerer JavaScript-Code, der (noch) nicht von allen Browsern verstanden wird, muss in eine ältere JavaScript-Sprachversion zurückübersetzt werden, die von allen Browsern, auf denen deine Anwendung laufen soll, unterstützt wird. Das ist in der Regel ES5. Zudem muss der React-proprietäre JSX-Code (siehe Kapitel 4) in gültiges und verständliches JavaScript übersetzt werden. Zum Kompilieren des Sourcecodes wird in der Regel Babel oder TypeScript verwendet.
- Auch für die Arbeit mit JavaScript-Modulen wird ein Tooling benötigt. Zum einen verstehen nicht alle Browser das native JavaScript-Modulsystem, das 2015 eingeführt wurde. Zum anderen gibt es weitere Modulsysteme, in denen Module vorliegen können, die du möglicherweise in deiner Anwendung verwendet möchtest. Ein populäres Modulsystem ist das aus NodeJS stammende RequireJS, für das es eine Reihe von Modulen gibt, die auch im Browser laufen und die du bei Bedarf in deiner Anwendung verwenden kannst. Um deine eigenen und die von dir verwendeten Module für den Browser aufzubereiten, wird ein Bundler benötigt. Während ein Compiler sich um das Übersetzen eines Moduls von einer JavaScript-Sprachversion in eine ältere kümmert, sorgt der Bundler dafür, dass deine Module, unabhängig davon welches Modulsystem du verwendest, auch im Browser funktionieren. Eines der prominentesten Bundler ist Webpack.
- Für das Testen deiner Anwendung benötigst du ein Testbibliothek, das in der Lage ist, React-Komponenten zu testen. Die Tests sollen natürlich regelmäßig – zum Beispiel im CI-Build – ausgeführt werden.
- Während du deine Anwendung entwickelst, benötigst du einen Webserver, über den du deine Anwendung testweise betreiben kannst. Darüber sollten Änderungen, die du vornimmst, auch sofort im Browser sichtbar werden, ohne dass du viel Mühe damit hättest. Du benötigst also Werkzeuge für die Automatisierung. Hier kommt der Webpack Development Server ins Spiel, der dafür sorgt, dass bei Änderungen an deinem Sourcecode die Anwendung neu kompiliert und im Browser aktualisiert wird.
- Neben der Konfiguration dieser Tools für den Entwicklungsprozess benötigt man in der Regel auch eine spezialisierte Konfiguration für das Bauen einer Anwendung, die dann auch in Produktion ausgeliefert wird. Während bei der Entwicklung eher die Build-

Dauer und der Entwicklungskomfort, etwa zum Debuggen, im Vordergrund stehen, ist beim Bauen der Anwendung für die Produktion eher entscheidend, dass die fertigen Artefakte zum Beispiel möglichst klein werden.

Das Einrichten eines Projekts mit all den benötigten Tools sowie das manuelle Pflegen ihrer unterschiedlichen Versionen und Konfigurationen kann sehr zeitaufwendig sein. Aus diesem Grund gibt es das Projekt Create React App[1], das dir diese Aufgaben abnimmt. Mit Create React App kannst du ein neues React-Projekt erzeugen lassen. Das Projekt enthält dann fertige und sehr ausgereifte Konfigurationen für Babel (auf Wunsch TypeScript) und Webpack. Auch Unterstützung für CSS ist enthalten und es werden für alle Artefakte SourceMaps erzeugt, sodass du trotz proprietärem JSX-Code weiterhin zum Beispiel im Browser debuggen kannst (siehe Abb. 2–3). Darüber hinaus enthält das generierte Projekt Konfigurationen für einen Produktionsbuild, der nach den jeweils gültigen Best Practices implementiert ist und sehr gute Ergebnisse erzielt.

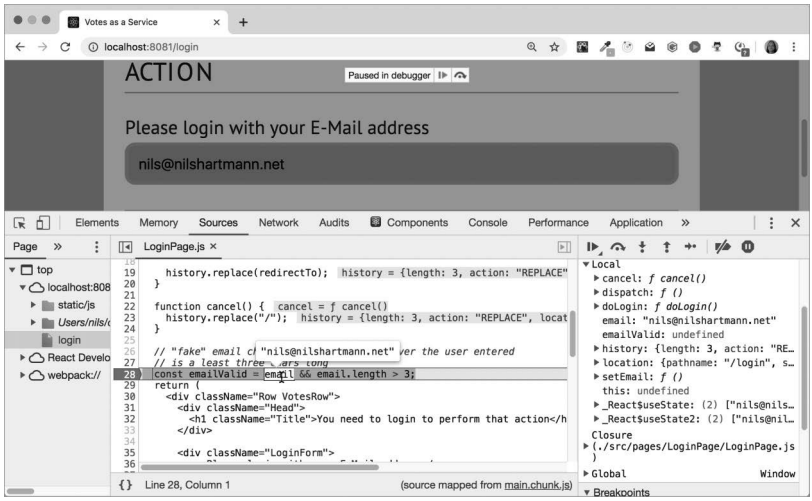

Abb. 2–3
Debuggen einer React-Komponente in Chrome

Um ein neues Projekt zu starten, kannst du mit dem Tool npx (Bestandteil von npm) create-react-app ausführen. Dazu öffnest du eine Kommandozeile und gehst in das Verzeichnis, in dem das neue Projekt (in einem neuen Verzeichnis) angelegt werden soll:

```
npx create-react-app PROJEKT_NAME
```

1 *https://facebook.github.io/create-react-app/*

Mit diesem Aufruf erstellt Create React App ein neues Verzeichnis und legt darin die `package.json`-Datei mit den notwendigen Abhängigkeiten an. Außerdem werden schon einige beispielhafte Source-Dateien angelegt, sodass du gleich mit dem Entwickeln anfangen kannst.

ESLint

Bestandteil des erzeugten Projekts ist auch die Konfiguration von ESLint[2], einem populären Linter für JavaScript. Ein Linter ist ein Tool zur statischen Codeanalyse, das Probleme im Sourcecode aufdecken kann (z.B. Verwendung nicht deklarierter Variablen und ungenutzte Variablen). ESLint lässt sich mit Regeln beliebig konfigurieren und es gibt auch fertige Regeln für die Entwicklung von React-Anwendungen. Einige Regeln sind in dem mit Create React App erzeugten Projekt bereits eingetragen, sodass du bei typischen Codeproblemen Hinweise darauf beim Build bekommst. Viele IDEs und Editoren (z.B. VS Code oder Webstorm) zeigen dir die Hinweise auch direkt beim Bearbeiten einer Datei an.

Abb. 2–4
Darstellung von Problemen aus ESLint in Visual Studio Code

Auch unsere Beispielanwendung haben wir initial mit Create React App erzeugt und nur die generierten Source-Artefakte gelöscht bzw. für unsere Zwecke angepasst.

2 *https://eslint.org/*

2.2.1 Progressive Web Apps mit React

Wenn du ein Projekt mit Create React App erstellst, ist der dabei erzeugte Code bereits für die Entwicklung einer Progressive Web App[3] (PWA) ausgelegt. Progressive Web Apps sind Webanwendungen, die über Features verfügen, die in der Vergangenheit nativen Anwendungen vorbehalten waren. Dazu gehört zum Beispiel die Möglichkeit, die Anwendung auf dem Homescreen eines Mobilgerätes installieren zu können. Dazu muss die Anwendung eine Manifestdatei enthalten, die zum Beispiel darüber Auskunft gibt, welchen Titel die Anwendung anzeigen und welches Icon sie verwenden soll. PWAs sind außerdem auch offline-fähig, sodass sie auch geöffnet und verwendet werden können, wenn keine Netzverbindung besteht. Dazu werden *Service Worker*[4] verwendet, die Daten cachen können.

Beide Artefakte, das Manifest und ein Service Worker, werden von Create React App automatisch erzeugt. Der generierte Service Worker ist aber per Default nicht aktiviert, da Entwicklung und Betrieb von Anwendungen mit einem Service Worker nicht unproblematisch ist. Du kannst den Service Worker aber in der generierten `index.js`-Datei jederzeit einschalten.

Damit hast du eine gute Basis, um deine React-Anwendung bei Bedarf als Progressive Web App zu bauen.

2.2.2 Hintergrund: react-scripts

Eine Besonderheit von Create React App bzw. den Projekten, die mit Create React App erzeugt wurden, ist, dass es die Abhängigkeiten auf die Tools (Babel, Webpack etc.) nicht direkt in deine `package.json`-Datei einträgt. Stattdessen wird ein Verweis auf ein Modul mit dem Namen `react-scripts` eingetragen. Dieses Modul wiederum enthält dann sämtliche Konfigurationen und bestimmt, welche Tools in welcher Version zum Einsatz kommen. Das Modul wird von dem Team gepflegt, das Create React App entwickelt. Gibt es neue Versionen der abhängigen Tools, zum Beispiel eine neue Webpack-Version, oder Verbesserungen in der Konfiguration, zum Beispiel eine Optimierung für den Build-Prozess, wird `react-scripts` von den Entwicklern angepasst und eine neue Version veröffentlicht. Du kannst dann von den Neuerungen auch in deinem bestehenden Projekt profitieren, indem du dort einfach die Version von `react-scripts` in deiner `package.json`-Datei anpasst.

---

3 *https://developers.google.com/web/progressive-web-apps/*
4 *https://developer.mozilla.org/de/docs/Web/API/Service_Worker_API/*

Anpassen der Konfiguration

Die Kehrseite der Medaille ist, dass du von der Konfiguration weitgehend abgeschnitten bist. Bis auf wenige Parameter kannst du die Konfiguration nicht beeinflussen. Zwar ist die Konfiguration sehr gut und für viele Einsatzzwecke auch völlig ausreichend, aber natürlich kann es immer vorkommen, dass du doch an der Konfiguration arbeiten musst. Für diesen Fall gibt es zwei Möglichkeiten: Zum einen kannst du das Projekt `react-scripts` auf GitHub klonen, anpassen und eine eigene Variante für dein Projekt davon erzeugen, die du dann in die `package.json`-Datei einträgst. Sofern du später feststellst, dass – z.B. durch eine neuere Version – `react-scripts` doch wieder für dich ausreichend ist, kannst du wieder auf `react-scripts` umstellen.

»Auswerfen« von react-scripts

Falls dieser Ansatz für dich nicht ausreichend ist, kannst du das Projekt `react-scripts` »auswerfen« lassen. Dieses Verfahren heißt in Create React App »eject«. Damit werden alle Konfigurationen und Abhängigkeitsbeschreibungen direkt in dein Projektverzeichnis kopiert und die Abhängigkeit auf `react-scripts` entfernt. Dein Projekt sieht dann so aus, als ob dort direkt (wie in einem »traditionellen« Projekt) die Konfigurationen und Abhängigkeiten eingetragen worden wären, sodass du dort beliebig Anpassungen machen kannst. Der Nachteil dieses Ansatzes ist, wenn du einmal »ausgeworfen« hast, kommst du nicht mehr zurück. Du bist dann von den Neuerungen in `react-scripts` abgeschnitten.

2.2.3 React-Anwendungen in Produktion betreiben

React-Anwendungen bestehen aus HTML-, JavaScript- und CSS-Dateien. Dazu kommen unter Umständen weitere statische Assets wie Bilder oder Fonts.

Wenn du dein Projekt mit Create React App erzeugt hast, kannst du mit dem `npm`-Script `build` dir eine fertige Anwendung bauen lassen, die für den produktiven Einsatz optimiert ist, unter anderem sind JavaScript- und CSS-Code minifiziert und unter Umständen auch auf mehrere Dateien aufgeteilt:

```
npm run build
```

Die erzeugen Dateien werden in das `build`-Verzeichnis geschrieben. Zum Veröffentlichen deiner Anwendung über einen Webserver musst du dieses Verzeichnis auf deinen Webserver kopieren und ihn als Web-Root für den gewünschten Server, die Domain oder den Pfad festlegen.

2.3 Zusammenfassung

In diesem Kapitel haben wir uns anhand einer sehr einfachen Beispielanwendung die grundlegenden Konzepte von React im Code angesehen.

- Komponenten in React werden als Funktionen geschrieben, die von außen Parameter (Properties) bekommen und intern einen Zustand halten. Die Komponente ist verantwortlich, dafür eine UI zu erzeugen.
- Mit dem Online-Editor CodeSandbox kannst du ohne etwas installieren zu müssen, React ausprobieren. Den Editor kannst du auch gut benutzen, um mit anderen über Code zu diskutieren (zum Beispiel für Bug-Reports).
- »Echte« React-Projekte kannst du mit Create React App automatisiert anlegen. Projekte, die damit begonnen werden, verfügen über eine vollständige Konfiguration für den Build-Prozess sowie die Verwaltung der Abhängigkeiten.