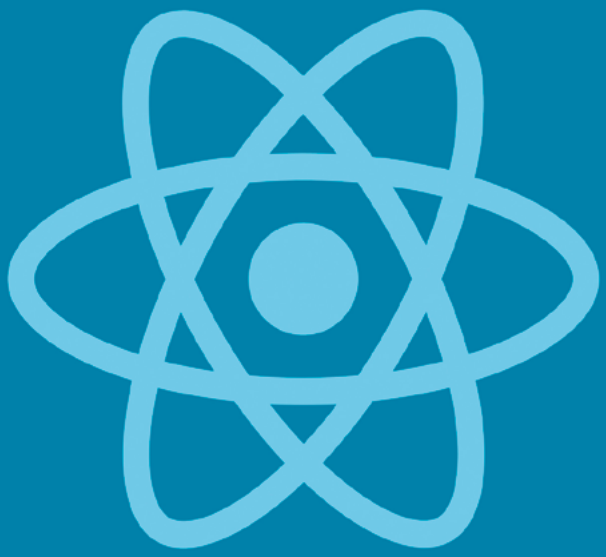

Inkl. React Native, Redux und Next.js

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
- Komponentenarchitektur, Hooks, State-Management und Formulargestaltung
- Tests, Performance, Routing, Server Side Rendering mit Next.js, Authentifizierung, KI-Applikationen u. v. m.

3., aktualisierte und erweiterte Auflage

Alle Beispielprojekte zum Download

Rheinwerk Computing

Liebe Leserin, lieber Leser,

React bietet Frontend-Entwicklerinnen und -Entwicklern alles, was sie für moderne Webanwendungen brauchen. Die Bibliothek hat sich zum Standard für performante Benutzeroberflächen entwickelt – vom klassischen Single-Page-Frontend bis hin zu Fullstack-Anwendungen mit Next.js. Gleichzeitig ist das Ökosystem in den letzten Jahren deutlich gewachsen, was den Einstieg nicht unbedingt einfacher macht.

In diesem Buch unterstützt Sie JavaScript-Experte Sebastian Springer dabei, diese Hürde sicher zu meistern. Er führt Sie strukturiert in die Grundlagen von React ein und zeigt Ihnen, wie Sie Anwendungen mit modernen Werkzeugen wie Next.js, React Native und TypeScript entwickeln. Auch typische Fragen aus der Praxis – etwa zur Architektur Ihrer Anwendung, zur Auswahl geeigneter Tools oder zum sinnvollen Einsatz von Bibliotheken – werden beantwortet. Anhand anschaulicher Beispielprojekte setzen Sie das Gelernte direkt um und lernen, worauf es im Projektalltag ankommt.

Darüber hinaus profitieren Sie von zahlreichen weiterführenden Themen: von Testing, Performance und Routing über State-Management mit Redux, moderne Serverkommunikation und Fullstack-Entwicklung mit Next.js bis hin zu Authentifizierung und ersten KI-gestützten Anwendungen. Dank des modularen Aufbaus können Sie gezielt auf einzelne Themen zugreifen und das Buch auch als Nachschlagewerk nutzen.

Zum Schluss noch ein Hinweis in eigener Sache: Dieses Buch wurde mit großer Sorgfalt geprüft und produziert. Sollten Sie dennoch Fehler finden, Fragen zum Buch haben, Anregungen, Lob oder Kritik loswerden wollen, schreiben Sie mir gerne jederzeit. Ich freue mich auf den Austausch mit Ihnen.

Und nun wünsche ich Ihnen bei der Entwicklung mit React viel Spaß und Erfolg!

Ihr Felix Jüstel
Lektorat Rheinwerk Computing

felix.juestel@rheinwerk-verlag.de
www.rheinwerk-verlag.de
Rheinwerk Verlag • Rheinwerkallee 4 • 53227 Bonn

Auf einen Blick

Impressum

Dieses E-Book ist ein Verlagsprodukt, an dem viele mitgewirkt haben, insbesondere:

Autor Sebastian Springer
Lektorat Felix Jüstel
Fachgutachten Philip Ackermann
Copy-Editing Friederike Daenecke, Zülpich
Typografie & Layout Vera Brauner
Satz E-Book III-satz, Kiel
Herstellung E-Book Nadine Preyl
Covergestaltung Silke Braun
Coverbild Shutterstock: 753412768 © baranq

Bibliografische Information der Deutschen Nationalbibliothek:
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.dnb.de* abrufbar.

ISBN 978-3-367-11545-7

3., aktualisierte und erweiterte Auflage 2026

Rheinwerk Verlag GmbH • Rheinwerkallee 4 • 53227 Bonn
service@rheinwerk-verlag.de

Informationen zu unserem Verlag und Kontaktmöglichkeiten finden Sie auf unserer Verlagswebsite *www.rheinwerk-verlag.de*. Dort können Sie sich auch umfassend über unser aktuelles Programm informieren und unsere Bücher und E-Books bestellen.

Inhalt

Materialien zum Buch

Auf der Webseite zu diesem Buch stehen folgende Materialien für Sie zum Download bereit:

- **Alle Codebeispiele aus dem Buch**

Gehen Sie auf *www.rheinwerk-verlag.de/6299*. Klicken Sie auf den Reiter **Materialien**. Sie sehen die herunterladbaren Dateien samt einer Kurzbeschreibung des Dateiinhalts. Klicken Sie auf den Button **Herunterladen**, um den Download zu starten. Je nach Größe der Datei (und Ihrer Internetverbindung) kann es einige Zeit dauern, bis der Download abgeschlossen ist.

Geleitwort des Fachgutachters

Die Webentwicklung schreitet kontinuierlich voran – und das schneller denn je. Was für Webentwickler*innen zugleich Fluch und Segen ist, hat sich in den letzten Jahren noch einmal deutlich beschleunigt. Neue Bibliotheken oder Frameworks entstehen in rasanter Geschwindigkeit, nicht zuletzt verstärkt durch den Einfluss von künstlicher Intelligenz auf die Softwareentwicklung. Erfahrene Entwickler*innen wissen jedoch: nicht jeder Hype hält, was er verspricht. Ein gesunder Pragmatismus und ein solides Verständnis grundlegender Konzepte bleiben entscheidend, um die richtigen technologischen Entscheidungen zu treffen. Eine davon ist zweifellos, sich mit der von Facebook entwickelten Bibliothek React zu beschäftigen.

Mittlerweile längst mehr als nur eine UI-Bibliothek, hat sich React zu einem zentralen Baustein moderner Webanwendungen entwickelt. Mit Konzepten wie Server Components, neuen Ansätzen zur Darstellung von Benutzeroberflächen und einer immer stärkeren Integration in Frameworks wie Next.js bleibt React auch über ein Jahrzehnt nach seiner Veröffentlichung hochrelevant.

Hinzu kommt, dass React bezüglich der dahinterstehenden Konzepte sehr durchdacht und deswegen wiederum außerordentlich performant ist und sich – gegenüber anderen großen Frameworks – in einem äußerst attraktiven Kosmos bewegt. Sei es der alternative Package Manager Yarn, der ebenfalls von Facebook entwickelt wird und von dem sich auch der offizielle Node.js Package Manager NPM schon bezüglich des ein oder anderen Features inspirieren lassen hat. Oder das Testing-Tool Jest, das sich dank Snapshot-Testing besonders für das Testen von React-Anwendungen eignet. Auch GraphQL, die flexiblere Alternative zu REST, stammt von Facebook und fügt sich nahtlos in React-Anwendungen ein. Dies in Summe ist auch der Grund, warum React gegenüber anderen vergleichbaren Bibliotheken und Frameworks weiterhin meine persönliche Präferenz bleibt.

Für Neueinsteiger*innen bringt all das natürlich auch Herausforderungen mit sich. Konzepte wie Redux, JSX und Hooks müssen verstanden werden, bevor sie sinnvoll eingesetzt werden können. Gleichzeitig gilt es, sich in einer zunehmend komplexen Tool-Landschaft zurechtzufinden.

Genau hier setzt die nun vorliegende, komplett überarbeitete 3. Auflage von Sebastian Springer an. Sie lernen, wie Sie React-Applikationen am besten aufsetzen, organisieren und planen, wie Sie Komponenten strukturieren und welche Best Practices Sie dabei beachten müssen. Besonders hervorzuheben ist erneut die didaktische Qualität: Die Inhalte sind klar strukturiert, bauen logisch aufeinander auf und bleiben dabei stets praxisnah. Sebastian gelingt es, sowohl Einsteiger*innen abzuholen als auch erfahrene Entwickler*innen auf den neuesten Stand zu bringen.

Als Fachgutachter hatte ich erneut die Gelegenheit, das Manuskript vorab zu lesen und kann sagen: Auch diese Auflage setzt neue Maßstäbe. Die Integration aktueller Themen

wie Künstliche Intelligenz, die sorgfältige Überarbeitung bestehender Inhalte sowie deren gezielte inhaltliche Vertiefung machen dieses Buch zu einem rundum gelungenen Werk. Mein Respekt, Sebastian! Und herzlichen Glückwunsch zu dieser neuen Auflage!

Ihnen, liebe Leserinnen und Leser, viel Spaß mit dem Buch und viel Erfolg beim Eintauchen in die Welt von React.

Philip Ackermann
Chief Technology Officer bei Cedalo GmbH

Vorwort

React hat sich als feste Größe in der Frontendwelt etabliert und ist als eine der drei großen Lösungen neben Angular und Vue kaum noch wegzudenken. Dabei steht React für mich für eine leichtgewichtige und flexible Art der Entwicklung, die dennoch hoch professionell ist. React lässt Ihnen beim Aufbau und der Gestaltung Ihrer Applikation ein hohes Maß an Freiheit. 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 gerade erst dabei sind in React einzusteigen oder schon Erfahrung damit haben, ich möchte Sie einladen, dieses Buch als Gelegenheit zur aktiven Arbeit mit React zu nutzen. Arbeiten Sie mit den Codebeispielen, erweitern Sie sie oder bauen Sie Ihre eigenen Applikationen. Versuchen Sie verschiedene Anforderungen umzusetzen, 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 nachvollziehen, 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, Vue, Svelte und noch viele weitere 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 sowohl kleinen, aber auch großen Projekten bewährt.

Diese zweite Auflage spiegelt die Weiterentwicklung von React und dem Ökosystem um die Bibliothek wider. Außerdem ist sehr viel Feedback von LeserInnen der ersten Auflage eingeflossen. So setzt das Buch nicht mehr auf ein durchgängiges und gegen Ende des Buches sehr umfangreiches Beispiel, sondern auf kleinere Beispielapplikationen. Diese beschäftigen sich zwar alle mit dem Thema Bücherverwaltung, bauen jedoch nicht linear aufeinander auf.

Bevor ich Ihnen kurz noch den Aufbau dieses Buches vorstelle, möchte ich Ihnen noch einen Hinweis mit auf den Weg geben: Die Welt der JavaScript-Frameworks und -Bibliotheken ist extrem schnelllebig und so werden Sie feststellen, dass sich die Versionsnummern und teilweise auch die Features der hier vorgestellten Bibliotheken verändern werden. Deshalb liefere ich Ihnen in diesem Buch auch Hintergrundinformationen zu Konzepten und Architekturmustern, die es Ihnen ermöglichen sich auf Veränderungen einzustellen und sich sehr schnell in neue Features und Bibliotheken einzuarbeiten.

Aufbau des Buchs

Das Buch besteht aus drei Teilen, die sich in insgesamt 26 Kapitel unterteilen. Der erste Teil des Buchs beschäftigt sich mit React selbst und allem, was Sie wissen müssen, um eine Applikation zu implementieren. Der zweite Teil beleuchtet das Ökosystem der Bibliothek mit verschiedenen Problemstellungen aus dem Projektalltag und den dazu passenden Bibliotheken. Im dritten Teil erfahren Sie, wie Sie React außerhalb seines angestammten Ökosystems im Browser nutzen können und implementieren PWAs, native mobile Apps und Fullstack-Applikationen mit Next.js.

Der erste Teil beginnt mit einer Einführung 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 den Umgang mit React-Komponenten kennen. Sie erfahren, wie Sie den Komponentenbaum Ihrer Applikation aufbauen, wie die Daten in diesem Baum fließen und sehen wie der Lebenszyklus einer Komponente funktioniert. In Kapitel 4 erfahren Sie, wie Sie TypeScript in Ihre React-Applikation integrieren können und was Sie dabei beachten müssen.

In Kapitel 5 stelle ich Ihnen den Lebenszyklus einer Komponente und weitere Konzepte wie Higher-Order Components, Render-Props und die Context-API vor. Kapitel 6 beschäftigt sich mit der Anbindung von Serverschnittstellen an Ihr Frontend. In Kapitel 7 erfahren Sie, wie Sie mit React Formulare umsetzen können, über die Benutzer mit Ihrer Applikation interagieren können. Kapitel 8 widmet sich der Hooks-API, einer Erweiterung von React, die die Art und Weise, wie eine Applikation aufgebaut wird, entscheidend beeinflusst. React unterstützt verschiedene Ansätze, wenn es um das Styling von Komponenten geht. Kapitel 9 stellt Ihnen einige dieser Ansätze vor und zeigt Ihnen, wie Sie sie in Ihre Applikation integrieren können. Ein wichtiges Hilfsmittel im Zusammenhang mit der Qualitätssicherung einer Applikation behandelt Kapitel 10, in dem es um das Formulieren von Unittests geht.

Der zweite Teil des Buchs beginnt mit einem Kapitel über die Integration externer Komponentenbibliotheken (Kapitel 11). Am Beispiel von Material-UI und Shadcn/ui sehen Sie, wie Sie existierende Komponenten in Ihre Applikation integrieren können. In Kapitel 12 erfahren Sie, wie Sie mithilfe des React-Routers innerhalb Ihrer Single-Page-Applikation navigieren und damit unterschiedliche Teilbäume Ihrer Applikation rendern können. Mit dem TanStack Router lernen Sie eine weitere Alternative für clientseitiges Routing kennen. Kapitel 13 behandelt mit eigenen Komponentenbibliotheken ein wichtiges Thema, wenn es darum geht Komponenten über Applikationsgrenzen wiederzuverwenden. Kapitel 14 und Kapitel 15 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 16 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 17 bindet mit `react-i18next` 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. Kapitel 18 widmet sich den Performanceverbesserungen in einer React-Applikation mit Memoisierung, dem React Compiler und Lazy Loading. In Kapitel 19 sehen Sie, wie Sie Authentifizierung in Ihrer React-Applikation umsetzen können.

Der dritte Teil des Buchs beschäftigt sich mit verschiedenen Umgebungen, in denen Sie React nutzen können. Den Anfang macht Kapitel 20 mit Progressiven Web Apps, also offlinefähigen und installierbaren Web-Applikationen. Kapitel 21 geht noch einen Schritt weiter und stellt Ihnen vor, wie Sie mithilfe von React Native Apps für mobile Endgeräte umsetzen können.

Die folgenden Kapitel beschäftigen sich mit Fullstack React Applikationen auf Basis von Next.js. In Kapitel 22 lernen Sie die Grundlagen von Next.js kennen. Kapitel 23 beschäftigt sich mit Dateisystemkonventionen, Client Components und Caching-Strategien in Next.js. In Kapitel 24 lernen Sie mit den Server Functions von Next.js ein weiteres wichtiges Feature für die Userinteraktion in Fullstack-Applikationen kennen. Kapitel 25 stellt Ih-

nen die wichtigsten Optimierungen von Next.js für performante Applikationen vor. Im Kapitel 26 erfahren Sie, wie Sie React in KI-Applikationen verwenden können.

Download der Codebeispiele

Sämtliche Codebeispiele dieses Buches sind auf der Website des Verlages unter *www.rheinwerk-verlag.de/6299* 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 wieder einmal bei einem meiner Bücher, die Begutachtung übernommen und viele Anmerkungen und Tipps beigesteuert hat.

Außerdem danke ich Friederike Daenecke für den sprachlichen Feinschliff.

Auch dem ganzen Team vom Rheinwerk Verlag möchte ich danken und hier vor allem Felix Jüstel für die hervorragende Betreuung.

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

Sebastian Springer
Aßling

Kapitel 1
Die ersten Schritte mit React

React ist eine JavaScript-Bibliothek, deren Schwerpunkt auf der Entwicklung von interaktiven Benutzeroberflächen im Web liegt. Dieses Kapitel soll Ihnen den Einstieg in die Welt von React erleichtern.

Sicher kennen Sie Facebook, eines der größten sozialen Netzwerke der Welt. Meta, das Unternehmen, das hinter diesem Netzwerk steht, hat für Facebook eine spezialisierte JavaScript-Bibliothek entwickelt, um die größten Probleme mit der Darstellung eines solchen sozialen Netzwerks zu lösen. Das Kernstück von Facebook ist der Newsfeed – und hier liegt auch die größte Herausforderung, die bei der Umsetzung entsteht: die große Menge an Daten, ihre Anzeige im Browser und ihre Aktualisierung. Da die Frameworks und Bibliotheken, die seinerzeit auf dem Markt verfügbar waren, die Anforderungen von Facebook nicht oder nur ungenügend erfüllten, entschloss sich das Unternehmen, eine eigene Bibliothek zu schreiben – das war die Geburtsstunde von React.

1.1 Was ist React?

React ist eine Bibliothek für die Implementierung von Web-Frontends. Es deckt jedoch nur den View-Layer ab, also die Darstellung der Oberfläche. Bezüglich der Struktur der Applikationslogik und weiterer architektonischer Zusammenhänge macht React keine Vorgaben. Dadurch ergeben sich im Vergleich zu anderen Frameworks und Bibliotheken deutlich mehr Freiheiten bei der Gestaltung der Strukturen. Diese Freiheit hat allerdings den Nachteil, dass es gerade für Einsteiger schwierig ist, den Umgang mit React zu erlernen und für größere Applikationen eine solide Architektur aufzubauen. Diese höhere Einstiegshürde, kombiniert mit einer steilen Lernkurve, ist auch einer der größten Kritikpunkte an React. Der anfängliche Mehraufwand zahlt sich während Entwicklung jedoch schnell wieder aus.

Eine der ersten Ressourcen für Ihren Lernpfad ist die offizielle Website von React mit zahlreichen weiterführenden Informationen und einem Tutorial, die Sie unter *https://react.dev/* finden.

Den Einstieg in die Welt von React erleichtern zahlreiche Werkzeuge, wie beispielsweise *Vite* mit seinen Applikations-Templates. Sie müssen sich also um das initiale Projekt-Setup nicht selbst kümmern. Doch dazu später noch mehr. Wie viele andere JavaScript-Bibliotheken und -Frameworks ist auch React ein Open-Source-Projekt. Die Entwickler

haben sich für die MIT-Lizenz entschieden. Bei ihr handelt es sich um eine etablierte Open-Source-Lizenz, die die private und kommerzielle Nutzung erlaubt, die Angabe der Lizenz und des Copyrights erfordert und jegliche Haftung ausschließt. Den Quellcode von React verwaltet das Entwicklungsteam auf GitHub. Das Repository erreichen Sie unter *https://github.com/facebook/react*. Für eine einfache Installation steht Ihnen React als NPM-Paket zur Verfügung. *NPM* ist der aktuell größte Paketmanager der Welt. Er wird für nahezu sämtliche JavaScript-Projekte verwendet und stellt ein öffentliches Repository sowie ein Kommandozeilenwerkzeug zur Paketverwaltung zur Verfügung. Alternativ zu NPM können Sie *Yarn* oder *pnpm* nutzen. React macht Ihnen hier keinerlei Vorgaben.

Vite

Vite ist ein modernes Build-Werkzeug für JavaScript-Projekte, das schnelle Entwicklungs- und Build-Zeiten ermöglicht und verschiedene Frameworks und Bibliotheken unterstützt. Die wichtigsten Features sind der Entwicklungsserver mit *Hot Module Replacement*, der dafür sorgt, dass Änderungen am Code sofort im Browser sichtbar werden, und der Bundler, der optimierte Builds für den Produktivbetrieb erzeugt. Die Webseite des Projekts mit vielen weiteren Informationen finden Sie unter *https://vite.dev/*.

Eine der Besonderheiten von React ist, dass Sie die Bibliothek nicht nur im Browser, sondern auch auf anderen Plattformen nutzen können. Beispiele dafür sind native mobile Apps oder die Kommandozeile. Dieser plattformunabhängige Ansatz erklärt auch, warum die Bibliothek auf mehrere Pakete aufgeteilt ist. So können plattformspezifische Bestandteile wie der *Renderer*, der für die Darstellung der Applikation zuständig ist, mit wenig Aufwand ausgetauscht werden, während die übrige Struktur erhalten bleibt. Im Verlauf dieses Kapitels lernen Sie mit dem Renderer und dem *Reconciler* zwei wesentliche Bestandteile von React noch genauer kennen. Letzterer kümmert sich um das Auffinden der Unterschiede zwischen zwei Applikationszuständen für eine möglichst performante Darstellung der Applikation.

1.1.1 Single-Page-Applikationen

In der Regel nutzen Sie React zur Implementierung von *Single-Page-Applikationen (SPA)*. Das Schlüsselkonzept dieser Art von Anwendungen beruht darauf, dass der Browser nicht vollständig neu geladen werden muss, wenn ein neuer Zustand der Applikation angezeigt werden soll. Daher rührt auch der Name »Single Page«, da die Applikation im Grunde genommen aus nur einer einzelnen HTML-Seite besteht. Mithilfe von JavaScript passen Sie diese so an, dass sie den jeweils gültigen Zustand der Applikation widerspiegelt. SPAs haben jedoch den Nachteil, dass der initiale Ladevorgang deutlich länger dauert als bei einer traditionellen Webseite. Der Grund ist, dass Sie alle Ressourcen laden müssen, die für die Darstellung der Applikation erforderlich sind. Für dieses Problem existieren je-

doch mit *Lazy Loading* und *Server-Side Rendering* etablierte Lösungen, die Sie im Verlauf dieses Buches noch kennenlernen werden.

Für die Benutzer einer Webapplikation ist der Single-Page-Ansatz deutlich angenehmer, da die Übergänge zwischen den Sichten deutlich flüssiger sind und sich eher wie native, auf dem System installierte Applikationen anfühlen. Das Gegenstück zum Single-Page-Ansatz sind Multi-Page-Applikationen, bei denen der Browser die einzelnen Ansichten der Applikation bei jedem Seitenwechsel vollständig neu lädt. Für eine React-Applikation kommt eine Multi-Page-Architektur nur in seltenen Ausnahmefällen infrage, da bei jedem Ladevorgang der gesamte Quellcode von React geladen werden muss. Dieses Problem lässt sich zwar durch den Einsatz des Browsercaches minimieren, sodass nur der erste Ladevorgang mehr Zeit in Anspruch nimmt und alle folgenden wesentlich schneller ablaufen. Nach dem Laden des Quellcodes muss React dann aber noch ausgeführt werden und die sichtbare Struktur der Applikation aufbauen.

Hinzu kommt, dass ein Reload im Browser dazu führt, dass der aktuelle Zustand der Applikation im Frontend verworfen und anschließend neu aufgebaut werden muss. Das hätte für eine React-Applikation zur Folge, dass Sie den Zustand sämtlicher Komponenten serverseitig oder im Webstorage des Browsers festhalten müssen. In einer Single-Page-Applikation bleibt der Zustand der Applikation während einer Browsersitzung erhalten, und Sie können auf den Inhalt des Speichers zugreifen und dort Objekte ablegen.

React nutzt die Charakteristiken einer SPA aus, um die Usability zu verbessern und an Performance zu gewinnen, sodass sich Webapplikationen wie native Applikationen anfühlen. Bis React jedoch den Optimierungsgrad erreicht hatte, den es heute aufweist, erhielt die Bibliothek zahlreiche Verbesserungen und Erweiterungen.

1.1.2 Die Geschichte von React

Im Vergleich mit seinen wichtigsten Konkurrenten liegt React in der Mitte, wenn Sie die Erscheinungsdaten der initialen Releases betrachten. Die erste Version von *Angular* kam 2009 auf den Markt, *Vue.js* wurde im Jahr 2014 veröffentlicht. React wird seit 2011 bei Facebook eingesetzt und hat seitdem eine bewegte Geschichte durchlebt. Ursprünglich wurde es beim zentralen Element von Facebook, dem Newsfeed, eingesetzt. 2012 kam React dann bei Instagram, das von Meta aufgekauft wurde, zum Einsatz. Die Entscheidung für den Einsatz von React trug auch entscheidend zur Abstraktion der Bibliothek bei.

React im Vergleich zu den anderen Frameworks

Spricht man von SPA-Frameworks, so sind meist die »großen Drei« gemeint, also Angular, React und Vue. Obwohl alle drei auf einer komponentenbasierten Architektur aufbauen, unterscheiden sie sich in ihren Implementierungsdetails teilweise deutlich. Sehr vereinfacht ausgedrückt, lassen sich die drei Lösungen wie folgt einordnen:

- **Angular:** Der Fokus von Angular liegt auf einer sauberen Architektur. Das Framework bringt bereits die meisten Features mit, die Sie für die Umsetzung benötigen. Durch diesen Ansatz verlieren Sie etwas an Freiheit bei der Entwicklung, die Vorgaben und Best Practices erleichtern jedoch auch die Arbeit.
- **React:** React verfolgt einen leichtgewichtigen und flexiblen Ansatz. Sie erhalten lediglich die Grundfunktionalität, mit der Sie die grafische Oberfläche Ihrer Applikation aufbauen können. Für weitere Features müssen Sie auf das umfangreiche Ökosystem zurückgreifen. Auch architektonisch macht React kaum Vorgaben.
- **Vue:** Was Architekturvorgaben und Flexibilität angeht, ist Vue zwischen Angular und React angesiedelt. Sie erhalten eine leichtgewichtige Architektur mit nur wenigen Vorgaben, und auch um Vue hat sich eine sehr aktive Community gebildet.

Die Entstehung von React

Die Geschichte von React beginnt im Jahr 2011 unter dem Namen FaxJS. Jordan Walke entwickelte mit dieser Bibliothek einen Prototyp von React. Eines der Kernelemente von FaxJS war das nahtlose Rendern, egal ob serverseitig oder clientseitig; auch schon diese Version war unabhängig von der Umgebung. Außerdem spielte die Reaktivität eine herausragende Rolle. Bei Änderungen an den Daten der Applikation sollte sich das Frontend automatisch anpassen. Die schnelle Darstellung von Inhalten und geringe Ladezeiten waren ein weiteres Merkmal von FaxJS. Die Bibliothek verfolgte, wie auch React, einen komponentenorientierten und deklarativen Ansatz für den grafischen Aufbau von Applikationen. FaxJS wiederum wurde von XHP inspiriert, einem HTML-Komponentenframework.

2012 entwickelte Jordan Walke React als vollständig neues Projekt, aber mit den Grundideen von FaxJS.

Im Jahr 2015 wurde mit *React Native* neben dem Browser mit nativen Apps eine weitere Plattform von React unterstützt.

Der Sprung von Version 0.x auf Version 15

React folgt, wie die meisten JavaScript-Bibliotheken, dem Semantic-Versioning-Ansatz. Lange Zeit wurde React in der Version 0.x im Rahmen von Minor-Updates entwickelt. Eine 0.x-Version steht für eine eingeschränkte Stabilität. Diese Vorgehensweise hat sich auch bei anderen Projekten, beispielsweise bei Node.js, bewährt. Auch hier befand sich das Projekt über mehrere Jahre in dieser Entwicklungsphase. Der Sprung auf eine Major-Version, wie im Fall von React auf 15, sollte den Verwendern signalisieren, dass das Projekt uneingeschränkte Produktionsreife und Stabilität erreicht hat.

Semantic Versioning

Der Semantic-Versioning-Ansatz arbeitet mit drei Versionsnummern, die üblicherweise so aussehen: 19.2.3. Die einzelnen Versionsnummern, die durch Punkte getrennt werden, haben eine festgelegte Bedeutung:

- **Major-Version:** Das ist die erste Versionsnummer, im Beispiel die 19. Ändert sich diese Versionsnummer, müssen Sie mit sogenannten *Breaking Changes* rechnen. Das sind größere und inkompatible Änderungen, bei denen ältere Versionen Ihrer Applikation ohne Eingriffe in den Quellcode nicht mehr funktionieren. Beispiele für einen Major-Change sind das Entfernen einer API, das Umbenennen einer Funktion oder grundlegende Änderungen in der Architektur.
- **Minor-Version:** Dies ist die zweite Versionsnummer, im Beispiel die 2. Sie wird erhöht, wenn neue Features hinzukommen, die jedoch abwärtskompatibel sind. Eine wichtige Anforderung an ein Minor-Release ist, dass bestehender Code ohne Anpassungen weiterhin funktioniert. Beispiele sind hier neue, optionale Optionen, neue Funktionen oder zusätzliche Komponenten.
- **Patch-Version:** Das ist die letzte Versionsnummer, im Beispiel die 3. Diese Versionsnummer wird für Bugfixes verwendet, die keine neuen Features einführen. Die API der Bibliothek bleibt unverändert und auch das Verhalten bleibt gleich.

Eine Besonderheit beim Semantic-Versioning-Ansatz sind 0.x-Versionen. Sie erlauben auch in Minor-Versionen Breaking Changes und stehen für den schnellen Entwicklungsfortschritt zu Beginn eines Projekts.

React wurde seit jeher produktiv auf der Facebook-Plattform verwendet. Hier kommen die neuen Versionen auch heute noch vor dem eigentlichen Release schon zum Einsatz, sodass sie zum Release-Zeitpunkt bereits ausgiebig in der Praxis getestet sind. Dies und der bis zu diesem Zeitpunkt langjährige produktive Einsatz sollte durch den Sprung auf die Version 15 noch weiter unterstrichen werden.

Mit dem Schritt auf Version 15 im April 2016 ergaben sich einige signifikante Änderungen an React selbst. So wurde beispielsweise die Unterstützung für den Internet Explorer 8 entfernt. Der Entwicklungsprozess wurde durch die Einführung von *Requests for Comments*, kurz *RFCs*, transparenter. Eine Änderung beziehungsweise ein Änderungswunsch an React wird seit der Einführung dieses Entwicklungsprozesses als RFC veröffentlicht und zur Diskussion gestellt. Diese RFCs finden Sie auf GitHub unter *https://github.com/reactjs/rfcs*. Außerdem werden, ebenfalls zur Verbesserung der Transparenz, die *Meeting Notes* des React-Core-Teams veröffentlicht.

Die wichtigste technische Neuerung in React 15 war der deutlich verbesserte Umgang mit dem DOM des Browsers. Hier wurde nun statt der bisher verwendeten `innerHTML`-Methode die `createElement()`-Methode verwendet. Außerdem entfiel die Notwendigkeit, Text in `span`-Elemente zu wrappen, und die Unterstützung von SVG wurde komplettiert.

React 16 – der nächste große Schritt

Lange Zeit fieberte die React-Gemeinde dem Erscheinen von Version 16 entgegen. Das Herzstück dieser Version war der neue Reconciler *Fiber*. Der Reconciler ist der Algorithmus, der die Unterschiede zwischen der aktuellen und der nächsten Version der grafischen Oberfläche berechnet. Fiber sollte einen deutlichen Performanceschub mit sich bringen und – was noch viel wichtiger ist – den Weg für zukünftige Entwicklungen und neue Features ebnen. Dies wurde nötig, da der *Stack*-Reconciler gerade bei Animationen an seine Grenzen stieß. Websites wie *https://isfiberreadyyet.com* verdeutlichen die Stimmung zu dieser Zeit. In Abbildung 1.1 sehen Sie die Ansicht der Webseite am 26.03.2017.

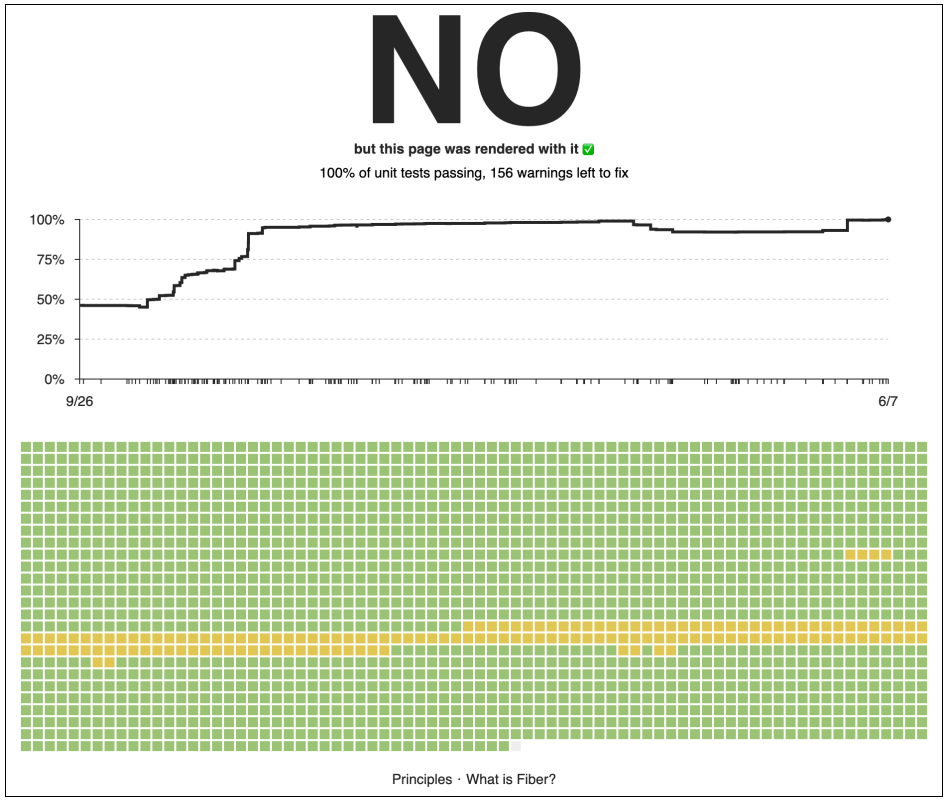

Abbildung 1.1: https://isfiberreadyyet.com/ am 26.03.2017

Die Webseite beantwortet die Frage, ob das neue Release bereits verfügbar ist, mit einem prominent sichtbaren »No«. Außerdem sehen Sie den Status der Unit-Tests der Bibliothek. Dabei handelt es sich um kurze, automatisierte Tests, die einzelne Funktionen oder Komponenten isoliert überprüfen, um ihre korrekte Funktionsweise sicherzustellen.

Am 26.09.2017 war es dann endlich so weit: React 16 wurde der Öffentlichkeit als neue stabile React-Version zur Verfügung gestellt.

Mit dem neuen Reconciliation-Algorithmus, auf dessen Funktionsweise ich in diesem Kapitel noch näher eingehen werde, und den folgenden Minor-Releases wurden zahlreiche Verbesserungen und Erweiterungen in den Kern von React aufgenommen:

- **React 16.2:** Der Lebenszyklus von Komponenten war bis zu dieser Version nicht darauf ausgerichtet, dass die Erzeugung des Komponentenbaums abgebrochen werden konnte. Mit neuen Lifecycle-Methoden sollte dies möglich werden. Ein möglicher Grund für einen solchen Abbruch ist beispielsweise eine höher priorisierte Änderung, wie sie bei Animationen zum Einsatz kommt. Ein zweites erwähnenswertes Feature in dieser Version ist die Finalisierung der Context-API, die zum Austausch von Informationen zwischen Komponenten über die Baumstruktur hinweg zum Einsatz kommt. Diese Schnittstelle wurde mit einigen Anpassungen so umgebaut, dass sie komfortabler zu benutzen ist.
- **React 16.4:** Eines der wichtigsten Merkmale von React ist, dass die Bibliothek plattformunabhängig ist, also sowohl auf mobilen Geräten als auch auf Desktopsystemen eingesetzt werden kann. Um diesem Umstand Rechnung zu tragen, wurde die Unterstützung von Pointer-Events in die Bibliothek aufgenommen. Damit werden Touchoberflächen und Stifte auch nativ als Eingabegeräte unterstützt.
- **React 16.6:** Ein bedeutendes Thema in der Entwicklung von React ist die Performance der Bibliothek. Mit der `lazy()`-Funktion wurde der erste Teil des Suspense-Features umgesetzt, das den Namen *Suspense for Lazy Loading* trägt. Es ermöglicht das asynchrone Nachladen von Inhalten. Die `lazy()`-Funktion lädt Komponenten asynchron. Suspense rendert in der Zwischenzeit einen Platzhalter. Mit der statischen `contextType`-Eigenschaft einer Komponente wird der Zugriff auf die Context-API noch komfortabler gestaltet.
- **React 16.8:** Mit diesem Release wurde die Hooks-API eingeführt. Hinter diesem zunächst unscheinbar anmutenden Feature steckt jedoch eine erhebliche Aufwertung der Funktionskomponenten von React. Mit Hooks ist es möglich, in einer Funktionskomponente sowohl den Lifecycle der Komponente als auch einen eigenen lokalen State zu implementieren, was vor diesem Feature lediglich den Klassenkomponenten vorbehalten war. Zum Thema Hooks erfahren Sie in Kapitel 8, »Die Hooks-API von React«, noch mehr. Mit diesem Feature wird die Entkopplung von Logik und State von der Darstellung noch einen Schritt weitergetrieben. Trotz der erweiterten Möglichkeiten brachte die Einführung der Hooks-API keine Breaking Changes mit sich, sodass sich auch diese Erweiterung in eine Reihe von Minor-Updates einreiht, die der Roadmap der Entwicklung zu einer performanteren und besser zu benutzenden Bibliothek folgen.
- **Version 16.x:** Ein weiteres Feature, das auf die Verbesserung der Performance zielt, ist der *Concurrent Mode*, der auch als *Async-Mode* bezeichnet wird. Mit diesem Feature wird die Komponentenhierarchie der Applikation in einem separaten Prozess berechnet, ohne den Hauptprozess des Browsers zu blockieren. Mit der `lazy()`-Funktion wurde der erste Teil des Suspense-Features umgesetzt, das das Nachladen von Kom-

ponenten ermöglichte. Das Feature *Suspense for Data Fetching* sorgt dafür, dass auch andere Serveranfragen, wie beispielsweise asynchrone `fetch()`-Aufrufe zum Auslesen von Daten, mit einem Platzhalter versehen werden können.

Das Entwicklerteam musste während der Entwicklung der neuen Features, insbesondere des Concurrent Modes, feststellen, dass die ursprüngliche Roadmap zu optimistisch geplant war. Das Team wollte die neuen Features zuerst gründlich testen, bevor sie in das stabile Release aufgenommen wurden. Deshalb befand sich der Concurrent Mode über lange Zeit in einem experimentellen Stadium, in dem er gesondert aktiviert werden musste.

React 17 – das featurelose Update

Der Semantic-Versioning-Ansatz sieht vor, dass ein Major-Update Breaking Changes beinhalten kann. Umso mehr hat das Release der Version 17 von React die Community überrascht, da es mit der Überschrift »No New Features« angekündigt wurde. Der Schwerpunkt dieses Releases lag darauf, Upgrades auf neue Versionsnummern in bestehenden Applikationen zu erleichtern. Mit diesem Release hat das Entwicklerteam die *Gradual Upgrades* eingeführt, ein Feature, das es Ihnen ermöglicht, Ihre Applikation Stück für Stück auf eine neue Version zu heben.

Eine weitere Änderung unter der Haube bestand darin, dass React für die Eventbehandlung seine Eventlistener nicht mehr auf Dokument-Ebene, sondern auf dem Wurzelknoten der App registriert.

Mit den *Server Components* schlug React eine Richtung ein, um Client und Server näher zusammenzubringen. Dieses Feature soll den interaktiven Charakter der React-Komponenten, wie sie im Browser zum Einsatz kommen, mit der Performance der Serverseite verbinden. Dieses Feature wurde, wie die meisten anderen großen neuen Funktionalitäten, zunächst als experimentelles Feature in die Bibliothek aufgenommen.

React 18 – Concurrent React

Das wichtigste Feature der 18. Version von React ist der *Concurrent Renderer*. Dieses Feature ermöglicht es React, mehrere Versionen der grafischen Oberfläche vorzubereiten. Dieser Renderer ändert die Prinzipien des bisherigen Prozesses und ermöglicht es der Bibliothek, den Renderingvorgang zu unterbrechen, ihn später fortzuführen oder einen neuen Vorgang zu starten. Für die Benutzer einer Applikation wirkt sich das in einer responsiveren UI aus, da die Applikation unmittelbarer auf Benutzerinteraktionen reagieren kann. Mit diesen Verbesserungen wurden auch neue Optimierungen für React eingeführt, wie das automatische Batching, bei dem mehrere State Updates in einem Rerender zusammengefasst werden. Außerdem gab es neue Features wie Transitionen und einen neuen Streaming Server Renderer.

Neben dem Concurrent Renderer bringt React 18 auch *Suspense for Data Fetching* für externe Frameworks mit, beispielsweise für Next.js.

Die Einführung der experimentellen Server Components mit React 18 hat den Weg für React als Fullstack-Framework geebnet.

React 19 – das Fullstack-Update

Das Entwicklungsteam von React hat lange Zeit an einer aktualisierten Version der offiziellen Dokumentation gearbeitet und damit einen neuen Weg eingeschlagen. Dieser hat teilweise für große Kritik aus der Community gesorgt. Der Grund dafür war, dass in der Dokumentation für den Einstieg in React sehr prominent die Verwendung von Fullstack-Frameworks wie *Next.js* oder des *React Routers* empfohlen wird. Diese Fullstack-Frameworks verbinden serverseitiges und clientseitiges Rendern, was für bestimmte Applikationen entscheidende Vorteile mit sich bringt. Eine Fullstack-Applikation ist jedoch nicht in jedem Fall sinnvoll, und React lässt sich auch weiterhin in einer klassischen SPA verwenden. Das Entwicklerteam hat auf die Kritik reagiert und hat die Dokumentation an den entsprechenden Stellen angepasst.

Mit React 19 haben die *Server Components* den experimentellen Status verlassen und wurden stabiler Bestandteil von React. Außerdem hielten die Server Functions und noch weitere Schnittstellen wie `useActionState()` Einzug, die für den Einsatz in einer Fullstack-Applikation gedacht sind. Mit diesen Anpassungen und durch die enge Zusammenarbeit mit dem Next.js-Team ist React eine Bibliothek, die sich hervorragend für den Einsatz in einer Fullstack-Umgebung eignet.

Neben diesen Schnittstellen haben die Entwickler die Hooks-API von React um zusätzliche Funktionen wie `useOptimistic()` erweitert, mit dem optimistische Updates möglich werden. Außerdem wurde die `use()`-Funktion eingeführt, mit der Ressourcen wie beispielsweise Promises ausgelesen werden können.

Wenn Sie einen Blick auf die Entwicklung von React werfen, wird schnell klar, dass der Fokus stark auf Performance, aber auch auf einer guten Benutzbarkeit der Schnittstellen liegt. Getrieben wird die Entwicklung zum einen von Facebook selbst, wo React strategisch zur Implementierung von Web-Frontends eingesetzt wird, aber auch von einer sehr großen und aktiven Community, die sich um diese Bibliothek gebildet hat.

1.2 Warum React?

In der clientseitigen Webentwicklung herrscht seit langer Zeit eine emotionale Diskussion darüber, welches Framework denn das beste sei. Zur Auswahl stehen neben den drei großen Lösungen Angular, Vue und React auch noch zahlreiche kleinere Lösungen, beispielsweise Svelte. Wie in anderen Bereichen der Softwareentwicklung gibt es auch hier keinen klaren Sieger, es gibt kein Richtig oder Falsch, sondern nur Alternativen. Und so ist auch React »nur« eine Alternative zu seinen Konkurrenten.

Allerdings verfolgt React mit seinen Features, der Architektur und seiner Philosophie ein ganz bestimmtes Ziel. Der Schwerpunkt von React liegt auf der Gestaltung von Benutzer-

oberflächen, die Bibliothek gibt Ihnen beim Entwickeln also die Freiheit, alle übrigen Aspekte der Frontend-Entwicklung selbst zu gestalten. React baut, wie die anderen Frameworks auch, auf einer Komponentenarchitektur auf, nur dass React-Komponenten sehr leichtgewichtig sind: Komponenten sind im einfachsten Fall Funktionen mit einem bestimmten Rückgabetyp.

Die Schattenseite dieser Freiheit ist, dass Sie für viele Standardprobleme bei der Entwicklung von Webapplikationen zusätzliche Bibliotheken benötigen. Typische Beispiele sind das Formularhandling oder das Routing. In diesen Fällen steht Ihnen nicht nur eine Standardlösung, sondern stehen mehrere Alternativen zur Auswahl. Das ist ein häufig angeführter Kritikpunkt, der es Einsteigern schwer macht, die richtige Wahl zu treffen.

Ein weiteres Merkmal von React ist, dass es keine vordefinierte Architektur gibt wie etwa bei Angular. Konzepte wie *Dependency Injection* oder *Services* sind React fremd. Sie haben die Freiheit, Ihre Architektur so zu gestalten, wie Sie sie für Ihre Applikation benötigen, und React unterstützt Sie dabei, macht jedoch keine strikten Vorgaben. Diese Tatsache ist jedoch nicht immer nur von Vorteil, da sie es gerade Einsteigern erschwert, ihren Weg in React zu finden. Aus diesem Grund bemühen sich sowohl das React-Team als auch die Community, es neuen Entwicklern und Entwicklerinnen möglichst einfach zu machen, React zu erlernen und die Bibliothek produktiv zu verwenden.

1.2.1 Der Release-Zyklus

Wie schon erwähnt, folgt React bei der Versionierung dem Semantic-Versioning-Ansatz. Im Gegensatz zu anderen Open-Source-Projekten sieht React aber nicht einen regelmäßigen Release-Zyklus vor, in dem beispielsweise halbjährlich neue Major-Releases veröffentlicht werden.

Die Entwickler von React sind darauf bedacht, dass Breaking Changes, sprich Major-Updates, auf ein Minimum reduziert werden. Das spiegelt sich auch in der Frequenz der letzten Major-Releases wider: React 15 wurde im April 2016 veröffentlicht, die Version 16 erschien im September 2017, React 17 im Oktober 2020, React 18 im März 2022 und React 19 schließlich im Dezember 2024.

Stehen Änderungen an der Schnittstelle an, wie beispielsweise im Lebenszyklus der Klassenkomponenten, dann werden sie nicht direkt in der nächsten Version eingeführt, während die vorangegangene Version nicht mehr weiter unterstützt wird. Stattdessen werden in einem solchen Fall die bisherigen Methoden umbenannt und stehen für eine Übergangszeit noch weiter zur Verfügung. Für solche Anpassungen bieten die Entwickler von React Ihnen mit *Codemods* ein Werkzeug zur automatisierten Anpassung des Quellcodes. Dieses Werkzeug stelle ich Ihnen in Kapitel 2, »Die ersten Schritte im Entwicklungsprozess«, detaillierter vor. Außerdem arbeitet das React-Team mit *Deprecations*. Das Team markiert also Teile der Bibliothek, die in Zukunft entfernt werden, sodass alle, die Applikationen entwickeln, gewarnt werden und entsprechende Refactorings einplanen können.

Bemerkenswert an der Entwicklung von React ist, dass selbst bahnbrechende Änderungen, wie die Einführung der Hook-API und die damit verbundene Abkehr von Klassenkomponenten, ohne Breaking Changes vonstattengingen. In diesem Fall hat das React-Team neue Features eingeführt, die von der Community angenommen wurden und sich schnell zum neuen Standard entwickelt haben.

1.3 Die wichtigsten Begriffe und Konzepte der React-Welt

Die Entwicklung einer React-Applikation entscheidet sich in einigen Aspekten gravierend von der traditioneller Webapplikationen. React folgt, wie die meisten SPA-Frameworks, einem komponentenorientierten Ansatz beim Aufbau einer Applikation. Das bedeutet, dass sich eine Applikation aus vielen kleinen Bausteinen zusammensetzt, die lose miteinander verbunden sind. Außerdem weist React einige Optimierungen auf, durch die Veränderungen in der Darstellung der Applikation sehr performant angezeigt werden können. Das Ziel von React ist es, auch mit sehr großen Datenmengen im Frontend umgehen zu können.

1.3.1 Komponenten und Elemente

Die wichtigsten Bausteine einer React-Applikation sind die *Komponenten*. Sie sorgen für die Struktur der Oberfläche. Eine Komponente sollte möglichst klein in ihrem Umfang und unabhängig von ihrer Umgebung sein. Dadurch können Sie sie an mehreren Stellen in der Applikation oder sogar in mehreren Applikationen verwenden. Den Kern einer Komponente bildet eine *Funktion*, die die Struktur der Komponente zurückgibt. Diese wird als `render()`-Funktion bezeichnet.

Die Struktur der Anzeige können Sie auf zwei verschiedene Arten definieren:

- Zum einen können Sie die Methode `React.createElement()` nutzen, um neue Elemente zu erzeugen, und mit diesen Elementen eine Hierarchie aufbauen.
- Zum anderen ist es möglich, JSX (*JavaScript XML*), eine Syntaxerweiterung für JavaScript, zu benutzen und damit direkt im JavaScript-Quellcode Ihrer Komponente eine HTML-ähnliche Schreibweise zu verwenden, um die Struktur zu beschreiben.

Die gebräuchlichere und hier im Buch durchgängig verwendete Variante ist die zweite mit JSX. Der Ansatz mit `createElement()` hat den entscheidenden Nachteil, dass der Code sehr umständlich und schlecht zu lesen ist.

Komponenten sind nicht die kleinste Einheit, die Ihnen in React zur Verfügung steht. Auf der untersten Ebene stehen die *React-Elemente*. Diese werden, je nach Umgebung, in native Elemente übersetzt. Für die Browserumgebung bedeutet das, dass das React-`div`-Element in ein HTML-`div`-Element übersetzt wird. In einer Komponente können Sie sowohl Komponenten als auch Elemente verwenden, um die Struktur Ihrer Applikation zu beschreiben. Zur Unterscheidung von Elementen und Komponenten hat React die Kon-

vention eingeführt, dass Elemente stets mit einem Kleinbuchstaben und Komponenten mit einem Großbuchstaben beginnen. In Listing 1.1 sehen Sie ein Beispiel für eine typische React-Komponente. Die `List`-Komponente nutzt das `ul`-Element zur Darstellung einer Liste und die `ListItem`-Komponente, um die einzelnen Listeneinträge zu rendern:

```
import ListItem from './ListItem';

const List = () => {
  return (
    <ul>
      <ListItem value="Cherry" />
      <ListItem value="Apple" />
      <ListItem value="Banana" />
    </ul>
  );
};

export default List;
```

Listing 1.1: Beispiel für eine React-Komponente (»List.jsx«)

Die `List`-Komponente wiederum können Sie in eine andere Komponente, wie beispielsweise in die `App`-Komponente einbinden. Dafür importieren Sie die `List`-Komponente und integrieren Sie als `<List>`. In Listing 1.2 sehen Sie den Code, der dafür notwendig ist:

```
import List from './List';

const App = () => {
  return (
    <div>
      <h1>Fruit List</h1>
      <List />
    </div>
  );
};

export default App;
```

Listing 1.2: Integration der »List«-Komponente (»App.jsx«)

Für eine verbesserte Wiederverwendbarkeit von Komponenten lassen sich diese parametrisieren. Verwenden Sie JSX, können Sie einer Komponente, die Sie als JSX-Tag schreiben, Attribute übergeben. Diese werden in React als *Props* bezeichnet und ermöglichen die Übergabe von Objekten im Komponentenbaum.

Generell verfügen React-Komponenten über einen definierten Lebenszyklus, in den Sie an verschiedenen Stellen eingreifen können, um das Verhalten einer Komponente zu beein-

flussen. Komponenten können außerdem über einen eigenen Zustand verfügen. Dieser Zustand, häufig auch als *State* bezeichnet, ist eine Datenstruktur, die Informationen für die Darstellung der Komponente beinhaltet. Ändert sich der Wert des States, sorgt React dafür, dass die Änderung im Browser dargestellt wird: Die Komponente wird neu gerendert.

Bei den Komponenten unterscheidet React wiederum zwei Kategorien: Klassen- und Funktionskomponenten.

Klassenkomponenten

Die Klassenkomponenten von React sind in gewisser Weise Relikte aus einer vergangenen Zeit. Sie waren vor der Einführung der Hook-API in Version 16.8 der Standard, wenn es um Komponenten ging, da nur sie über einen eigenen State und Lifecycle-Methoden verfügten. Die Einführung der Hook-API hat jedoch dafür gesorgt, dass die Klassenkomponenten an Bedeutung verlieren. In modernen React-Applikationen sind Klassenkomponenten kaum noch zu finden.

Der Grund für den Abstieg der Klassenkomponenten ist, dass die zweite Art der Komponenten, die Funktionskomponenten, eher dem Charakter von React entsprechen. Sie sind leichtgewichtiger und flexibler. Außerdem löst die Hook-API einige Probleme der Klassenkomponenten. Auf diese Aspekte gehen wir jedoch in Kapitel 8 noch im Detail ein.

Der Name »Klassenkomponente« rührt von der Tatsache her, dass es sich hierbei um eine JavaScript-Klasse handelt, die sich von der Basisklasse `React.Component` ableitet. In früheren Versionen von React wurden Klassenkomponenten durch die Methode `React.createClass()` erzeugt, wobei diese Methode mittlerweile nicht mehr zur Verfügung steht und Sie nur noch die klassenbasierte Variante verwenden sollten. Alternativ dazu können Sie mit dem Zusatzpaket `create-react-class` eine Schnittstelle verwenden, die sich sehr ähnlich wie `React.createClass()` verhält.

Das Herzstück der Klassenkomponente ist die `render()`-Methode. Sie sorgt dafür, dass die Komponente dargestellt werden kann. Die Methode gibt ein React-Element oder eine Komponente als Rückgabewert zurück, den React zur Darstellung nutzt. Die `List`-Komponente aus dem vorangegangenen Beispiel würde als Klassenkomponente wie in Listing 1.3 aussehen. Hierbei ändert sich lediglich die interne Struktur; die Einbindung der Komponente bleibt gleich:

```
import React from 'react';
import ListItem from './ListItem';

class List extends React.Component {
  render() {
    return (
      <ul>
        <ListItem value="Cherry" />
        <ListItem value="Apple" />
        <ListItem value="Banana" />
```

```
      </ul>
    );
  }
}

export default List;
```

Listing 1.3: Aufbau einer einfachen Klassenkomponente (»List.jsx«)

React verwaltet den State einer Klassenkomponente in Form einer Eigenschaft und bildet den Lebenszyklus über verschiedene Methoden ab, beispielsweise über `componentDidMount()`.

Falls Sie mit der Umsetzung einer neuen React-Applikation beginnen, sollten Sie den Einsatz von Klassenkomponenten vermeiden und stattdessen auf die moderneren Funktionskomponenten setzen.

Funktionskomponenten

Funktionskomponenten bestehen, wie der Name schon andeutet, aus einer einzelnen Funktion. Diese ist nichts weiter als die `render()`-Methode einer Klassenkomponente und sorgt für die Darstellung der Komponente. Standardmäßig verfügte eine solche Funktionskomponente vor der Einführung der Hooks-API in Version 17 über keinen eigenen State und keine Lifecycle-Methoden. Funktionskomponenten wurden in ihrer ursprünglichen Form als einfache Anzeige-Komponenten verwendet. Mit der Einführung der Hooks-API änderte sich dies jedoch, und Funktionskomponenten wurden zu vollwertigen React-Komponenten mit State und Lifecycle und haben mittlerweile Klassenkomponenten fast vollständig verdrängt. Dieses Buch setzt, bis auf wenige Ausnahmen, ausschließlich auf Funktionskomponenten zur Entwicklung von Applikationen.

1.3.2 Der Datenfluss

Eine React-Applikation lebt von ihrer Dynamik. In der Regel interagieren die Benutzer mit Ihrer Applikation und produzieren und konsumieren damit Informationen. Ein Kernelement bei der Erstellung einer React-Applikation ist die Modellierung von Datenströmen. Im Komponentenbaum der Applikation fließen die Informationen immer von den Elternelementen in Richtung ihrer Kinder.

Wenn Sie von einer typischen React-Applikation ausgehen, die zur Verwaltung von Informationen genutzt wird, dann ist typischerweise eine der ersten Anforderungen, eine Listendarstellung der Informationen zu implementieren. Im Komponentenbaum wird die Liste selbst durch eine Komponente abgebildet. Die einzelnen Einträge sind wiederum Komponenten, die in einer Eltern-Kind-Beziehung mit der Liste stehen. Die Listeneinträge sind die Kindelemente der Liste (siehe Abbildung 1.2). Um die Informationen darzustellen, lesen Sie die Daten an einem zentralen Punkt, beispielsweise in der Listenkompo-

nente, ein und verteilen die Informationen an die einzelnen Kindelemente. Diese Lösung hat den entscheidenden Vorteil, dass lediglich eine zusammengefasste Anfrage erforderlich ist und Sie sofort eine Übersicht erhalten.

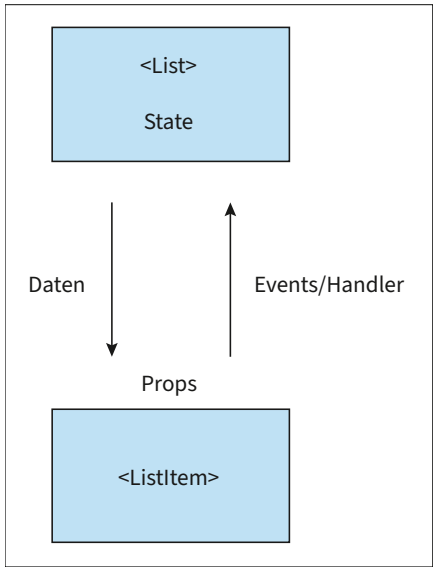

Abbildung 1.2: Datenfluss in einer React-Applikation

Die `List`-Komponente enthält den State, ist also für die Datenverwaltung zuständig. Sie lädt die Daten zur Anzeige vom Server herunter und kümmert sich darum, dass Änderungen zum Server gesendet werden. Für jeden Datensatz wird eine Kindkomponente erzeugt, die als Props die darzustellenden Informationen erhält. Mit dieser Methode wird ein gerichteter Datenfluss realisiert, durch den React in der Lage ist, die Darstellung der Oberfläche sehr performant zu gestalten. Ein Nebeneffekt dieser Komponentenaufteilung ist, dass Sie die Komponenten an mehreren Stellen in Ihrer Applikation verwenden können.

Mit dieser Art des Datenflusses haben jedoch die Kindkomponenten keine Möglichkeit, Informationen zurück an ihre Elternkomponenten zu geben. Dies wird jedoch notwendig, wenn Informationen modifiziert wurden und die Elternkomponente darüber benachrichtigt werden soll. Eine Lösung hierfür besteht darin, dass Sie neben den Informationen zur Darstellung auch Callback-Funktionen von der Eltern- an die Kindkomponente übergeben. Diese als *Event-Handler* bezeichneten Funktionen werden von der Kindkomponente in bestimmten Fällen ausgeführt, also beispielsweise bei einer Änderung der Daten. Die Funktion arbeitet im Gültigkeitsbereich der Elterkomponente, hat also Zugriff auf deren interne Strukturen und kann beispielsweise den State modifizieren. Dadurch haben Sie die Möglichkeit, die Informationen an die Elternkomponente zurückzugeben.

Die Performancevorteile, die durch den gerichteten Datenfluss entstehen, stammen daher, dass React genau die Stellen kennt, an denen die Anzeige der Applikation geändert werden muss.

1.3.3 Der Renderer

In der Standardkonfiguration einer React-Applikation werden die Pakete `react` und `react-dom` installiert. Das `react`-Paket enthält die Bibliothek selbst. Das zweite Paket, `react-dom`, enthält den Renderer. Dieser Bestandteil dient zur Übersetzung der React-Elemente in konkrete HTML-Elemente, die dann im Browser dargestellt werden können.

Der native Renderer – React Native

Ein weiterer Renderer für React ist *React Native*. Dieser sorgt dafür, dass die Elemente einer React-Applikation in ihre nativen Entsprechungen in einer iOS- oder Android-App übersetzt werden. Nutzen Sie unterschiedliche Renderer, sollten Sie beachten, dass in den verschiedenen Umgebungen eigene Elemente zum Einsatz kommen. So können Sie mit `react-dom` beispielsweise `div`-Elemente nutzen. In React Native nutzen Sie stattdessen das `View`-Element als Containerelement.

Eine React-Applikation kann nicht nur im Browser oder auf einem mobilen Gerät gerendert werden. Es existieren noch zahlreiche weitere Renderer, die als NPM-Pakete verfügbar sind. *Ink* ist beispielsweise ein Renderer für interaktive Kommandozeilen-Applikationen mit React.

1.3.4 Der Reconciler

Das Hauptpaket von React enthält unter anderem den Reconciler. Dieser Algorithmus ist für die Erkennung von Änderungen in einer Applikation verantwortlich. Eine Anpassung an einer Komponente kann entweder über eine Veränderung der Props oder des States erfolgen. Damit eine Änderung im Browser wirksam wird, muss im Fall der Browserumgebung der DOM-Baum zumindest teilweise neu aufgebaut werden. Das Rendern von HTML-Strukturen ist immer noch eine der größten Schwachstellen für die Performance von Webapplikationen. React bietet für diese Problematik eine Lösung in Form des Virtual DOMs. Das *Virtual DOM* ist ein Abbild der Struktur der Applikation in Form von JavaScript-Objekten.

Wird eine Anpassung der DOM-Struktur erforderlich, erzeugt React eine neue Version des Virtual DOMs. Diese Struktur dient als Bauplan für die neue Version der Applikation. React versucht anschließend, das bestehende DOM anhand einer Reihe von optimierten Aktionen anzupassen.

Für die Optimierung des Reconciler-Algorithmus werden zwei Annahmen getroffen, um die Komplexität zu reduzieren:

- Zwei Elemente mit unterschiedlichen Typen produzieren unterschiedliche Bäume.
- Mit der `key`-Prop kann angegeben werden, welche Kindelemente zwischen zwei Render-Schritten stabil bleiben.

Beim Vergleich des ursprünglichen Zustands und des Zielzustands geht React von der Wurzel in Richtung der Kindelemente vor und führt die Vergleiche durch.

Wie schon erwähnt, überprüft der Reconciler zunächst, ob die Typen zweier Elemente sich entsprechen. Handelt es sich um unterschiedliche Typen, greift die erste Optimierung des Algorithmus. Diese besagt, wie schon erwähnt, dass zwei verschiedene Typen zu zwei unterschiedlichen Bäumen führen (siehe Abbildung 1.3).

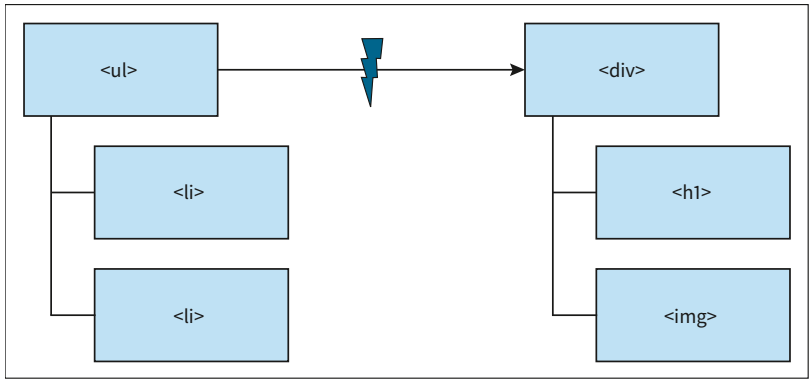

Abbildung 1.3: Unterschiedliche Baumstrukturen

React kann in diesem Fall die Überprüfung abbrechen und einen komplett neuen Unterbaum aufbauen. Das bedeutet, dass alle bisher aufgebauten Strukturen verworfen und alle Komponenten ausgehängt werden. Dadurch wird der Lebenszyklus der Komponenten beendet und, falls vorhanden, die entsprechende Unmount-Logik ausgeführt, die zum Aufräumen der Umgebung eingesetzt wird. Anschließend werden die neuen Komponenten aufgebaut und die passenden Lifecycle-Funktionen ausgeführt.

Gleichen sich die Typen zweier Elemente, werden die Attribute der Elemente überprüft und nur die Werte entsprechend angepasst. Ändert sich beispielsweise der Klassenname des Elements oder das `style`-Attribut, wird das zugrunde liegende DOM-Element entsprechend angepasst und der Unterbaum nicht verworfen (siehe Abbildung 1.4).

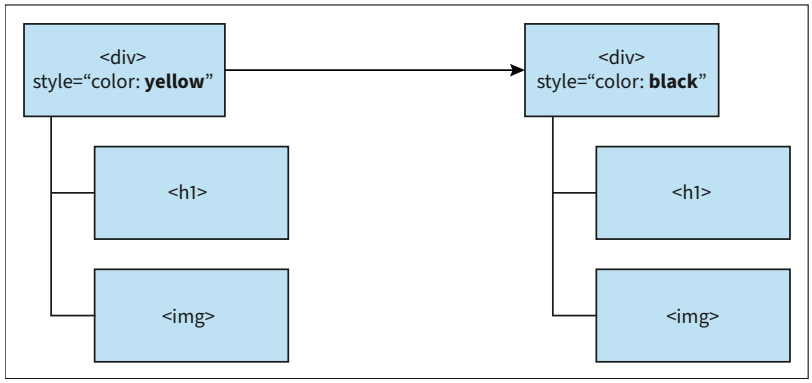

Abbildung 1.4: Änderung von Attributen in Baumstrukturen

Mit Komponenten verhält es sich ähnlich wie bei Elementen mit geänderten Attributen: Die Komponenteninstanz bleibt die gleiche, der State bleibt also erhalten, und die entsprechenden Lifecycle-Funktionen werden ausgeführt. Anschließend werden die Kindelemente und -komponenten verarbeitet.

Mehrere Kindelemente und die »key«-Prop

Bei einer Liste von mehreren Elementen kann es vorkommen, dass lediglich die Reihenfolge, nicht aber die Elemente selbst verändert werden. Relevant wird dies unter anderem, wenn an einer bestimmten Stelle der Liste ein neues Element eingefügt wird. Daraufhin ändert sich der Index aller nachfolgenden Elemente, was für React bedeuten würde, dass sie neu aufgebaut werden müssen. Mithilfe der `key`-Prop kann dies verhindert werden. Die `key`-Prop muss in einer Liste einen eindeutigen und stabilen Wert enthalten. Wird die `render()`-Methode der Elternkomponente erneut ausgeführt, wird der Wert der `key`-Props verwendet, um zu prüfen, ob sich die Elemente geändert haben.

Typischerweise kommen als Wert für die `key`-Prop die eindeutigen IDs von Datensätzen zum Einsatz. Der Wert muss dabei nicht numerisch sein. Verpflichtend ist allerdings, dass der Wert innerhalb der Liste – nicht über die gesamte Applikation – eindeutig ist und sich zwischen den verschiedenen `render()`-Aufrufen nicht ändert, da ansonsten die Zuordnung nicht korrekt erfolgen kann.

Warnmeldung ohne »key«-Prop

Als Hilfestellung während des Entwickelns gibt React bei jeder Iteration über eine Liste von Elementen ohne `key`-Prop eine Warnung aus. Diese Warnung besagt, dass jedes Kindelement einer Liste eine eindeutige `key`-Prop aufweisen sollte. Gerade bei umfangreichen Listen kann die Verwendung der `key`-Prop die Rendering-Performance erheblich verbessern.

Mit diesen Konzepten haben Sie einen ersten Einblick in die Welt von React erhalten. Die folgenden Kapitel dieses Buchs bauen auf diesen Begriffen auf und liefern zusätzliche detaillierte Erklärungen. Im nächsten Abschnitt lernen Sie einige weitere Bibliotheken kennen, die häufig im Zusammenhang mit React zum Einsatz kommen.

1.4 Ein Blick in das React-Universum

React ist eine spezialisierte Bibliothek zur Erstellung von grafischen Oberflächen in Web-Frontends. Im Gegensatz zu vollumfänglichen Frameworks (wie beispielsweise Angular) decken solche Bibliotheken nur einen bestimmten Teilaspekt einer Applikation ab. Je umfangreicher eine Applikation wird, desto mehr zusätzliche Bibliotheken müssen eingebunden werden, um die Entwicklung zu beschleunigen und den Quellcode übersichtlich zu halten. In den folgenden Abschnitten stelle ich Ihnen einige der wichtigsten Konzepte

und Bibliotheken des React-Universums kurz vor. Im Verlauf dieses Buchs lernen Sie diese Hilfsmittel noch im Detail kennen.

1.4.1 Das State-Management

Eine der populärsten Architekturformen großer React-Applikationen ist die *Flux*-Architektur. Diese sieht die Entkopplung der Präsentation vom Zustand und von der Geschäftslogik vor. Eine konkrete Implementierung der Flux-Architektur ist die *Redux*-Bibliothek. Im Kern stellt Redux einen zentralen Store zum Speichern der Informationen der Applikation zur Verfügung. Die Daten des Stores können gelesen, aber nicht direkt geschrieben werden. Hierzu müssen Sie den Umweg über die sogenannten *Actions* nehmen. Bei ihnen handelt es sich um einfache JavaScript-Objekte, die die Änderungen beschreiben. Diese werden von *Reducer*-Funktionen entgegengenommen, die wiederum den Store modifizieren können. Mehr zum zentralen State-Management und Redux erfahren Sie in Kapitel 14, »Zentrales State-Management mit Redux«.

1.4.2 Der Router

Verfügt die Applikation über mehrere Sichten, kann der Wechsel zwischen diesen recht aufwendig sein und zu unübersichtlichem Code in der Applikation führen. Eine Lösung, die auch in den meisten anderen Frontend-Frameworks zur Verfügung steht, ist *Routing*. Dieses Feature bezeichnet eine Erweiterung, mit deren Hilfe Komponentenbäume abhängig von der gewählten URL eingefügt werden können. Im Fall des React-Routers kann die History-API des Browsers verwendet werden.

Neben der reinen Navigation zwischen Komponentenbäumen unterstützt der Router weitere Features, wie Variablen in der URL, auf die Sie in den Komponenten zugreifen können, oder verschachtelte Routen.

1.4.3 Material UI

Im Web existieren zahlreiche Designempfehlungen. Eine der am weitesten verbreiteten ist das *Material Design* von Google. Damit Sie die einzelnen Elemente nicht selbst umsetzen müssen, existiert mit dem Material-UI-Paket eine Sammlung von Komponenten, die die Empfehlungen des Material Designs umsetzt. Die Komponentensammlung umfasst nicht nur Standardkomponenten wie Buttons oder Eingabefelder, sondern auch umfangreichere Komponenten wie Dialoge, Datentabellen oder Menüs.

1.4.4 Vitest

Vitest ist ein framework-unabhängiges Testframework, das hervorragend mit React harmoniert. Mit dem Testframework können Sie die Funktionalität Ihrer Komponenten, aber auch aller übrigen Applikationslogik automatisiert überprüfen. Diese Tests decken sowohl den Positivfall als auch Grenzwerte und Fehlerfälle ab und reduzieren so das Fehler-

potenzial in Ihrem Code. Vitest kommt im Standardfall ohne zusätzliche Konfiguration aus und kann die Tests sowohl im Browser als auch in einer simulierten Umgebung ausführen. Die Simulation sorgt auch dafür, dass die Tests im Vergleich zu anderen Frameworks (wie Jasmine in Kombination mit Karma) wesentlich schneller ausgeführt werden.

Ein weiteres bemerkenswertes Feature von Vitest ist das *Snapshot-Testing*, mit dessen Hilfe Sie eine statische Repräsentation einer Komponente erzeugen und diese als Vergleichsgrundlage nutzen können.

1.5 Thinking in React

In der Dokumentation von React finden Sie eine Sektion mit der Bezeichnung **Thinking in React** (*https://react.dev/learn/thinking-in-react*). Diese beschreibt, wie Sie bei der Erstellung einer Applikation vorgehen sollten. Hierbei spielt der komponentenorientierte Ansatz von React die Hauptrolle und sorgt dafür, dass eine Applikation aus einem Baum von Komponenten besteht. Diesen Umstand machen Sie sich beim Aufbau zunutze und folgen einem insgesamt fünfschrittigen Prozess.

Der Entwicklungsprozess beginnt stets mit einer konkreten Idee: Ein einfacher Mock reicht hier schon aus. Als konkretes Beispiel sehen Sie in Abbildung 1.5 ein Formular.

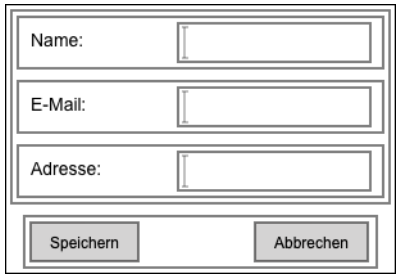

Abbildung 1.5: Entwurf eines Formulars

1.5.1 Die Oberfläche in eine Komponentenhierarchie zerlegen

Eine *Komponente* ist ein Baustein einer Applikation. Im einfachsten Fall können Sie die Abgrenzungen der Komponenten in Ihrem Mock durch Rechtecke darstellen. Dabei gehen Sie von den größeren Komponenten zu den kleineren vor.

Eine Komponente sollte dabei lediglich eine Aufgabe erfüllen. So können Sie das Formular beispielsweise in seine einzelnen Sektionen und diese wiederum in die Beschreibung und das Formularfeld unterteilen.

1.5.2 Eine statische Version in React implementieren

Die im vorherigen Schritt identifizierten Komponenten setzen Sie nun statisch um, also nur den aktuellen Zustand ohne Interaktivität oder Zustandslogik. Dadurch entsteht eine

Sammlung von Komponenten. Diese können Sie bei der weiteren Entwicklung Ihrer Applikation nutzen.

Die einzelnen Komponenten können ineinander geschachtelt werden, und die Informationen über Props können dann innerhalb der Komponentenhierarchie weitergegeben werden.

1.5.3 Den minimalen UI State bestimmen

Der State der Komponentenhierarchie enthält sowohl die Daten, die angezeigt werden sollen, als auch den Zustand einzelner sichtbarer Teile der Komponente. Bei der Modellierung des States sollten Sie darauf achten, dass Sie keine Duplikate produzieren. Diese müssen während der Laufzeit der Applikation synchronisiert werden und bergen das Risiko von Fehlern und Inkonsistenzen in sich.

1.5.4 Den Speicherort des States bestimmen

Nachdem Sie nun wissen, welche Daten Sie benötigen, müssen Sie noch den passenden Speicherort für sie festlegen. In der Regel platzieren Sie den State dort, wo er direkt benötigt wird. Eine Ausnahme bildet der Fall, dass Sie den State an einer übergeordneten Stelle platzieren, um die Informationen an mehrere Kindkomponenten weiterzugeben.

1.5.5 Den inversen Datenfluss modellieren

Die Informationen aus dem State werden über Props an die Kindkomponenten weitergegeben. Sollen Teile des States durch die Kindkomponenten modifiziert werden können, müssen Sie dies in Form von Funktionen implementieren. In diesem Fall übergeben Sie Funktionen von den Eltern- an die Kindkomponenten, in denen Sie den State der Elternkomponenten modifizieren.

Nach diesem Schema können Sie Ihre komplette Applikation aufbauen. Im Laufe der Zeit bauen Sie eine eigene Bibliothek aus Komponenten auf, die Sie wiederverwenden können, was die Entwicklung beschleunigt.

1.6 Codebeispiele

Im Verlauf dieses Buches werden Sie die einzelnen Bereiche von React und des Ökosystems, das sich um die Bibliothek gebildet hat, näher kennenlernen. Die Theorie zu lesen, ist das eine, das Gelesene praktisch anzuwenden, ist aber etwas ganz anderes. Nur wenn Sie selbst mit React und seinen zahlreichen Erweiterungen arbeiten, lernen Sie die Bibliothek wirklich kennen. Die Beispiele in den einzelnen Kapiteln sind in sich geschlossen, sodass sie nicht aufeinander aufbauen und unabhängig voneinander funktionieren.

Sie können in React nahezu jedes Thema umsetzen – angefangen von klassischen *CRUD-Applikationen* (Create-, Read-, Update- und Delete-Applikationen) zum Verwalten von Datensätzen über komplexe interaktive Applikationen, mit denen Sie Geschäftsprozesse abbilden, bis hin zu Browserspielen. Die einzelnen Kapitel sind zwar unabhängig, teilen sich jedoch ein Thema, das sich durch das gesamte Buch ziehen wird. Die Beispiele bilden die Teile einer Bibliothek ab, also einer Verwaltungsoberfläche für Bücher, mit der Sie Datensätze anzeigen, erstellen, modifizieren und wieder löschen können.

1.7 Zusammenfassung

Dieses Kapitel bildet den Einstieg in die Welt von React. Die wichtigsten Themen, die Sie in diesem Kapitel gelernt haben, sind:

- React wird in der Regel im Rahmen von Single-Page-Applikationen eingesetzt.
- Die Entwicklung von React reicht von der initialen Einführung in Facebook bis zur aktuellen Entwicklung und der Integration des *Concurrent Renderers* und der *Server Components*.
- React folgt dem Semantic-Versioning-Ansatz, bei dem es nur bei Major-Versionen zu Breaking Changes kommt. Das Entwicklungsteam versucht jedoch, solche Breaking Changes möglichst zu vermeiden.
- Komponenten sind die Bausteine einer React-Applikation. Sie kommen in React in zwei Ausprägungen vor: als die alten *Klassenkomponenten* und die moderneren *Funktionskomponenten* in Kombination mit der *Hook-API*.
- Den Datenfluss modellieren Sie mit *Props*, indem Sie Objekte und Funktionen an Kindkomponenten übergeben. Mit den Funktionen können Kindkomponenten die Elternkomponenten benachrichtigen.
- Der *Renderer* kümmert sich in React um die Darstellung in der jeweiligen Zielplattform.
- Der *Reconciler* ist der Algorithmus, der für die Berechnung der Unterschiede zwischen dem aktuellen Komponentenbaum und dem zukünftigen Komponentenbaum verwendet wird.
- Außerdem haben Sie einige Bibliotheken kennengelernt, die gemeinsam mit React in einer Applikation zum Einsatz kommen.
- Schließlich haben Sie noch erfahren, wie Sie beim Aufbau einer Applikation vorgehen.

In diesem Kapitel haben Sie sich vor allem mit den eher theoretischen Konzepten hinter React beschäftigt. Im nächsten Kapitel erfahren Sie, wie Sie mit der Entwicklung einer React-Applikation beginnen können.

Kapitel 2

Die ersten Schritte im Entwicklungsprozess

In diesem Kapitel legen Sie den Grundstein für Ihre React-Applikation: Sie richten die Entwicklungsumgebung ein, starten Ihren ersten Entwicklungsserver und lernen, wie Sie Ihre Anwendung erfolgreich bauen. Damit schaffen Sie die Basis, auf der alle weiteren Schritte Ihres Projekts aufbauen.

Initialisieren Sie Ihre Single-Page-Applikation selbst, ist dies meist mit großem Aufwand verbunden: Sie müssen Abhängigkeiten herunterladen und installieren, Strukturen schaffen, also Dateien und Verzeichnisse anlegen, und die Applikation entweder direkt vom Dateisystem starten oder über einen Webserver ausliefern. Doch keine Sorge: Für React gibt es, wie für die meisten anderen Frameworks und Bibliotheken auch, Hilfsmittel, die Ihnen diese Aufgaben abnehmen. In der Regel ist die Initialisierung damit nur noch ein Befehl auf der Kommandozeile, und schon können Sie mit der Entwicklung beginnen.

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.

Für die Initialisierung einer React-Applikation gab es in der Vergangenheit ein Standardwerkzeug mit dem Namen *Create React App*, das direkt von Facebook zur Verfügung gestellt wurde. Mittlerweile wird dieses Werkzeug jedoch nicht mehr weiterentwickelt. An seine Stelle sind eine Reihe von Community-Ansätzen getreten, von denen die Initialisierung mit *Vite* die am weitesten verbreitete Variante ist.

2.1 Schnellstart

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

Technische Voraussetzungen

Für die Initialisierung Ihrer Applikation benötigen Sie eine aktuelle Version von *Node.js* und *NPM*. Beides erhalten Sie über die Installation der Node.js-Plattform auf Ihrem System. Weitere Informationen dazu finden Sie auf der offiziellen Webseite von Node.js unter *https://nodejs.org/*.

2.1.1 Die Initialisierung

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

```
npm create vite@latest my-app -- --template react -no-interactive
```

Listing 2.1: Initialisierung einer Applikation mit NPM

Das `npm create`-Kommando erzeugt mithilfe von Vite und dem React-Template ein neues Verzeichnis mit dem Namen *my-app*, in dem sich Ihre Applikation befindet. Die Angabe von `--` trennt die NPM-Argumente von den Vite-Argumenten. Alles, was nach den `--` folgt, geht direkt an das Vite-Create-Skript. Die Option `--no-interactive` unterdrückt alle weiteren Rückfragen und sorgt dafür, dass Ihre Applikation ohne Kommandozeilen-Prompts erstellt wird.

Bei der Initialisierung erzeugt Vite alle erforderlichen Datei- und Verzeichnisstrukturen. Außerdem legt das Werkzeug eine *package.json*-Datei an, in der alle erforderlichen Paket-Abhängigkeiten aufgelistet sind. Mit diesen Schritten ist Ihre Applikation so weit vorbereitet, dass Sie mit den Kommandos aus Listing 2.2 die notwendigen Pakete herunterladen, Ihre Applikation starten und mit der Entwicklung beginnen können:

```
cd my-app
npm install
npm start
```

Listing 2.2: Starten der Applikation

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

TypeScript-Unterstützung

Ich empfehle Ihnen, Ihre React-Applikation, egal wie klein sie auch sein mag, immer mit *TypeScript* zu initialisieren. Für diesen Zweck gebt es ein weiteres Vite-Template für React-TypeScript-Applikationen. Das Kommando

```
npm create vite@latest my-app -- --template react-ts --no-interactive
```

Listing 2.3: Initialisierung mit TypeScript

nutzt das `react-ts`-Template und integriert TypeScript und alle erforderlichen Werkzeuge bei der Initialisierung in die Applikation. Anschließend führen Sie wieder die Kommandos aus Listing 2.2 aus und können mit der Entwicklung beginnen.

2.2 Playgrounds für React

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

Achtung!

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

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

2.2.1 CodePen – ein Playground für die Webentwicklung

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

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

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.

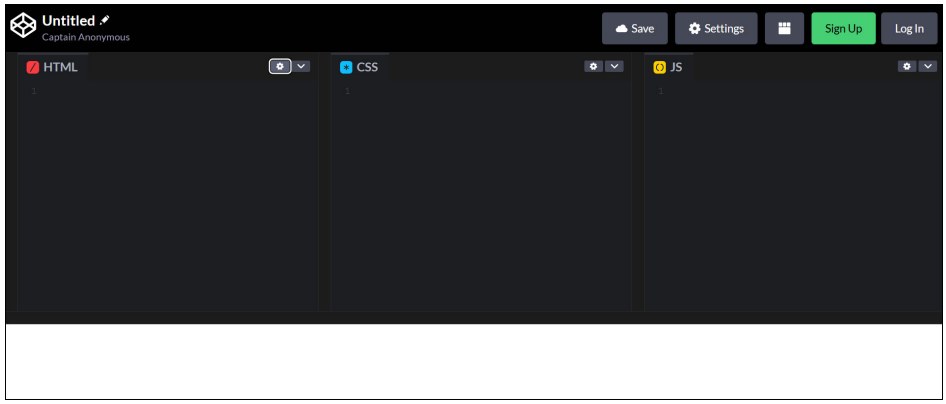

Abbildung 2.1: Ansicht von CodePen

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 **Einstellungen**-Icon im JavaScript-Editor klicken und dort unter dem Punkt **JS** zunächst **Babel** als **JavaScript Präprozessor** wählen. Mit dieser Konfiguration können Sie mit der Entwicklung Ihrer React-Applikation beginnen.

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

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

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

Listing 2.4: Container für die React-Applikation

In Listing 2.5 sehen Sie den JavaScript-Quellcode der Beispielapplikation. Dieser besteht aus der Definition einer Komponente mit dem Namen `Greet` und dem Rendern der React-Applikation:

```
import React from "https://esm.sh/react@19.2.3";
import {createRoot} from "https://esm.sh/react-dom@19.2.3/client";

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

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

Listing 2.5: Quellcode der React-Applikation

Im JavaScript-Editor importieren Sie zunächst die Pakete `react` und `react-dom` von esm.sh. *esm.sh* ist ein CDN-Dienst, der NPM-Pakete in ECMAScript-Module umwandelt, sodass Sie diese direkt in Ihrer Applikation verwenden können. Beim Import können Sie auch konkrete Versionsnummern angeben, was den Code deutlich stabiler macht. Anschließend erzeugen Sie eine einfache Komponente mit dem Namen `Greet`. Achten Sie darauf, dass der Name der Komponente mit einem Großbuchstaben beginnen muss.

Die Komponente ist als Arrow-Funktion implementiert, die ein JSX-Element zurückgibt. Als Argument erhält die Funktion ein Props-Objekt, über das Sie auf übergebene Werte zugreifen können. Auf den Wert `name`, den Sie über ein Destructuring-Statement extrahieren, können Sie in der JSX-Struktur mit geschweiften Klammern zugreifen. Die Details zum Aufbau einer Komponente erfahren Sie in den folgenden Kapiteln.

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

Die CodePen-Plattform ist einfach gehalten, es ist also nicht sinnvoll, größere Applikationen auf dieser Plattform umzusetzen. Die Entwicklung von Applikationen auf Ihrem eigenen System ist deutlich komfortabler und flexibler. Im folgenden Abschnitt erfahren Sie, wie Sie hierbei vorgehen können.

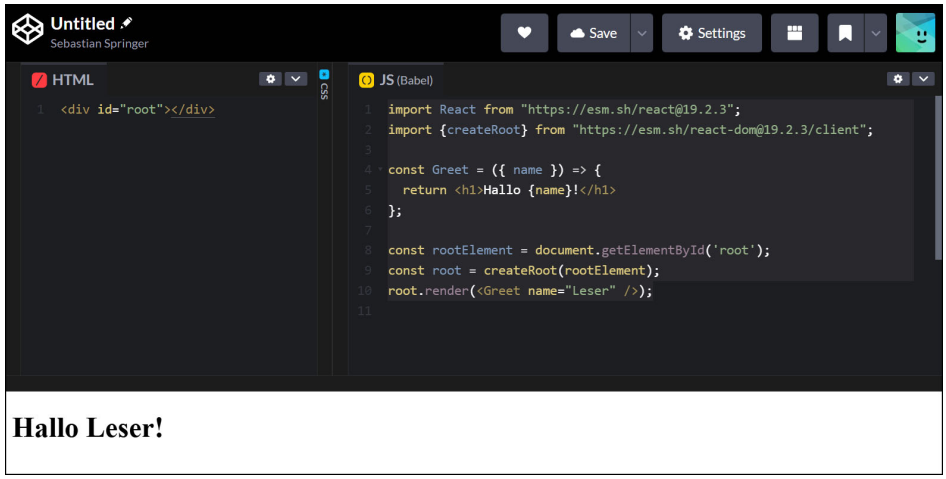

Abbildung 2.2: Ansicht der React-Applikation in CodePen

2.3 Lokale Entwicklung

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

2.3.1 React in eine HTML-Seite einbinden

Normalerweise steht eine React-Applikation für sich und umfasst das gesamte HTML-Dokument. Hinzu kommen der Build-Prozess und die Infrastruktur. Es ist jedoch auch möglich, React in eine bestehende HTML-Seite zu integrieren. In diesem Fall ist die React-Applikation lediglich ein Teil des umgebenden Dokuments, das noch weitere Inhalte zur Verfügung stellt.

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

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

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Hello React</title>

     <script type="importmap">
      {
        "imports": {
          "react": "https://esm.sh/react@19",
          "react-dom/client": "https://esm.sh/react-dom@19/client",
          "react/jsx-runtime": "https://esm.sh/react@19/jsx-runtime"
        }
      }
    </script>

    <script src="https://unpkg.com/@babel/standalone/babel.min.js"/>
  </head>

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

    <script type="text/babel" data-type="module" src="./index.jsx"/>
  </body>
</html>
```

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

In Listing 2.6 sehen Sie die Struktur dieser Einstiegsdatei. Zunächst bauen Sie eine *Importmap* auf. Diese sorgt dafür, dass der Browser weiß, wie er den Import der Pakete auflösen soll. Die Pakete kommen auch in diesem Beispiel von esm.sh und liegen dadurch im ECMAScript-Modulsystem vor. Damit Sie die komfortablere JSX-Syntax verwenden können, um Ihre Komponenten zu erzeugen, benötigen Sie außerdem Babel, das Sie von *unpkg.sh* beziehen können. Dieses Content Delivery Network stellt Ihnen Babel als Standalone-Paket zur Verfügung, das Sie direkt in Ihren Browser integrieren können. Mit diesem Setup können Sie schließlich die *index.jsx*-Datei einbinden, die Ihre eigene Applikation enthält.

Wichtig hierbei ist, dass Sie das `type`-Attribut mit dem Wert `text/babel` versehen und im `data-type`-Attribut den Wert `module` 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 ein `div`-Element als Container, in den Ihre Applikation eingehängt wird. Die Dateiendung *.jsx* signalisiert, dass es sich bei dieser Datei nicht um eine gewöhnliche JavaScript-Datei handelt, sondern dass JSX verwendet wird und deshalb ein Werkzeug wie Babel zur Übersetzung benötigt wird.

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

```
import React from 'react';
import { createRoot } from 'react-dom/client';

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

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

Listing 2.7: JavaScript-Quellcode der lokalen React-Applikation (»index.jsx«)

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

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

Einen lokalen Webserver nutzen

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

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

- **Lokale Installation**

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

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

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

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

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

- **Globale Installation**

  Die globale Installation erreichen Sie auf der Kommandozeile mit dem Befehl

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

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

  ```
  http-server .
  ```

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

- **On-Demand-Ausführung**

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

  ```
  npx http-server .
  ```

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

Welche Möglichkeit für welchen Zweck?

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

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

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

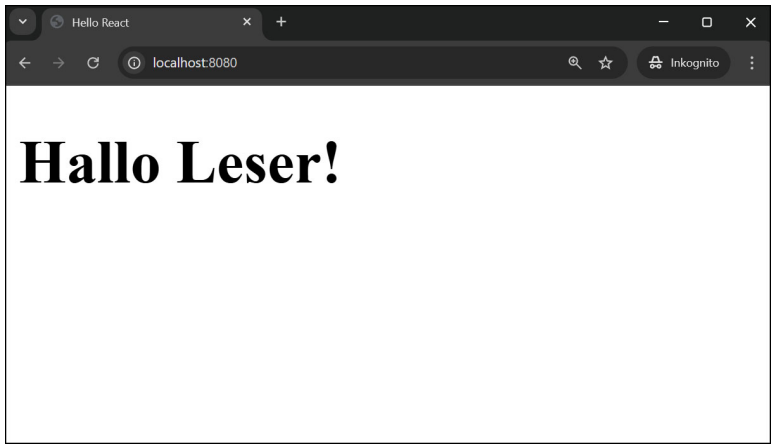

Abbildung 2.3: Lokale Ausführung der React-Applikation

2.4 Der Einstieg in die Entwicklung mit React

Setup von React

Bei den ersten Schritten mit einem JavaScript-Framework oder einer Bibliothek sollten Sie für Standardaufgaben wie das Setup immer auf Werkzeuge zurückgreifen, die Ihnen diese Aufgaben abnehmen. Im Fall von React ist eines der populärsten Hilfsmittel für diesen Zweck das Build-Werkzeug *Vite*.

Nahezu alle großen JavaScript-Frameworks verfügen über Kommandozeilenwerkzeuge, die Ihnen Routinearbeiten abnehmen, die im Zuge der Entwicklung anfallen. Die Aufgabe, die im Entwicklungsprozess am meisten Zeit beansprucht, ist das Einrichten der Umgebung. Das liegt vor allem daran, dass zahlreiche Werkzeuge für den Entwicklungs- und Build-Prozess zusammengefügt werden müssen.

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

2.4.1 Technische Voraussetzungen

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

Node.js

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

Node.js

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

Neben der eigentlichen JavaScript-Plattform enthält das Node.js-Paket außerdem *NPM*, den *Node Package Manager*, mit dem Sie die Abhängigkeiten Ihrer JavaScript- und TypeScript-Applikation verwalten können, auch im Frontend.

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
v25.1.0
$ npm -v
11.6.2
```

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 Komfort-Features wie beispielsweise Syntax-Highlighting für JavaScript, TypeScript und im besten Fall auch für React und JSX verfügt und Ihnen *Code Completion* bietet, also die korrekte Vervollständigung von begonnenen Variablen- und Methodennamen.

Browser

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

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

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

Listing 2.8: Polyfills für den Internet Explorer 11 laden

Polyfill

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

2.4.2 Das Build-Werkzeug Vite

Die bisher vorgestellten Varianten, um mit der Entwicklung mit React zu starten, haben den Nachteil, dass sie zwar einen schnellen Start in die Entwicklung erlauben, der Entwicklungskomfort jedoch auf der Strecke bleibt. Dieses Problem löst *Vite*, ein modernes Build- und Entwicklungswerkzeug. Das vorrangige Ziel von Vite ist es, die Entwicklung von Webapplikationen schneller und effizienter zu machen. Es bietet eine flexible Architektur, die eine Vielzahl von Bibliotheken und Frameworks unterstützt.

Im Kern besteht Vite aus zwei Komponenten:

- **Entwicklungsserver:** Der Entwicklungsserver liefert die Applikation während der Entwicklung lokal aus. Ein wichtiges Feature dieser Komponente ist das *Hot Module Replacement*, mit dessen Hilfe Vite die Ansicht des Browsers automatisch aktualisiert, ohne dass Sie Ihre Applikation manuell neu laden müssen.

- **Build-Prozess:** Für den Produktivbetrieb einer Applikation wird der Quellcode der Applikation zu einigen wenigen Dateien zusammengefasst und optimiert. Diese Aufgabe übernimmt der Build-Prozess aktuell noch mit dem in JavaScript geschriebenen Bundler *Rollup*. In Zukunft soll diese Aufgabe das in Rust geschriebene *Rolldown* übernehmen.

Die offizielle Website von Vite finden Sie unter *https://vite.dev*. Vite stammt ursprünglich aus der Vue.js-Welt, ist aber so unabhängig vom Framework gehalten, dass es auch hervorragend für React funktioniert. Im Gegensatz zu anderen CLI-Werkzeugen (wie beispielsweise dem Angular-CLI), die Sie über den gesamten Lebenszyklus einer Applikation begleiten, unterstützt Vite Sie vor allem beim Erstellungsprozess der Applikation.

Vite nimmt Ihnen nahezu alle Arbeiten ab, die bei der Initialisierung einer Applikation anfallen. Das Werkzeug erzeugt die Basisstruktur der Applikation und bereitet alle benötigten Abhängigkeiten vor, sodass Sie diese nur noch installieren müssen. Damit können Sie nach dem initialen Setup direkt mit der Entwicklung beginnen.

Das React-Template von Vite installiert lediglich die React-Basispakete `react` und `react-dom`. Außerdem werden einige zusätzliche Abhängigkeiten für die Entwicklung hinzugefügt, wie beispielsweise das `vite`-Paket, `eslint`, ein Werkzeug zur statischen Codeanalyse, oder verschiedene Pakete mit Typdefinitionen.

Doch nun genug der einleitenden Worte, beginnen wir mit der Entwicklung unserer Applikation!

Initialisierung mit »npm create«

Für JavaScript existiert zwar eine ganze Reihe von Paketmanagern, NPM ist jedoch mit Abstand der am meisten verwendete. Die einfachste Variante, um Ihre React-Applikation zu initialisieren, besteht aus der Verwendung des `npm create`-Kommandos. Das Kommando aus Listing 2.9 startet den interaktiven Setup-Prozess, an dessen Ende Sie über eine voll funktionsfähige React-Applikation verfügen:

```
$ npm create vite@latest

> npx
> create-vite

|
o  Project name:
|  my-app
|
o  Select a framework:
|    Vanilla
|    Vue
|  > React
…
```

```
o  Select a variant:
|    TypeScript
|    TypeScript + React Compiler
|    TypeScript + SWC
|  > JavaScript
…
o  Use rolldown-vite (Experimental)?:
|    Yes
|  > No
o  Install with npm and start now?
|  > Yes /   No
o  Scaffolding project in D:\temp\my-app...
|
o  Installing dependencies with npm...
  VITE v7.3.0  ready in 539 ms

  →  Local:   http://localhost:5173/
  →  Network: use --host to expose
  →  press h + enter to show help
```

Listing 2.9: Initialisierung einer React-Applikation mit »npm create«

In Listing 2.9 Sehen Sie die verkürzte Ausgabe des interaktiven Erstellungsprozesses. Im ersten Schritt werden Sie nach dem Projektnamen gefragt. Dieser legt den Namen des Verzeichnisses für die Applikation fest und wird in dem Verzeichnis angelegt, in dem Sie das Kommando ausführen. Im zweiten Schritt wählen Sie das Template aus, das Vite für die Erzeugung der Applikation nutzen soll. In unserem Fall ist das das React-Template. Für die verschiedenen Frameworks gibt es mehrere Varianten, so auch für React. Hier können Sie zwischen TypeScript und JavaScript und weiteren Konfigurationen wählen. Aktuell bietet Ihnen Vite an, zwischen dem Standard-Bundler *Rollup* und dem neuen und momentan noch experimentellen *Rolldown* zu wählen. Im letzten Schritt weisen Sie Vite noch an, dass es die Abhängigkeiten installieren und die Applikation im Entwicklungsmodus starten soll.

Der Node Package Manager (NPM)

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

Neben dem `npm`-Befehl gibt es weitere wichtige Bestandteile von NPM in Ihrem Projekt: Die *package.json*-Datei, die sich normalerweise im Wurzelverzeichnis Ihrer Ap-

plikation 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 zu installieren, die in der *package.json*-Datei aufgeführt sind.

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, ihre exakte Versionsnummer, ihren Speicherort in der NPM-Registry und einen Integritäts-Hash, der vor Manipulationen des Pakets schützt. Sie sollten sowohl die *package.json* als auch die *package-lock.json* Ihres Projekts versionieren.

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

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

Hinweis

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

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

Yarn

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

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

Yarn ist weitestgehend API-kompatibel mit NPM und verfügt über einen annähernd gleichen Funktionsumfang. An dieser Stelle werden Sie sich berechtigterweise fragen, warum Sie sich außer mit dem etablierten NPM noch mit einem weiteren Paketmanager beschäftigen sollten. Der Grund liegt in der Entwicklungsgeschichte von Yarn. Wenn Sie einen Blick auf die Website von Yarn unter *https://yarnpkg.com* werfen, springen 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 lösen, schufen einige Entwickler von Facebook den Paketmanager Yarn.

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

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

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

Initialisierung mit »yarn create«

Yarn bietet Ihnen eine Alternative zu NPM. Mit dem folgenden Kommando starten Sie den gleichen interaktiven Prozess wie bei NPM:

```
yarn create vite
```

Listing 2.10: Initialisierung einer React-Applikation mit Yarn

Egal für welche Variante der Installation Sie sich entschieden haben, nach der Ausführung des jeweiligen Kommandos können Sie direkt mit der Entwicklung Ihrer React-Applikation beginnen. Bevor wir im nächsten Schritt in die Struktur der Applikation einsteigen, folgen nun noch einige weiterführende Informationen zur Verwendung von Vite.

Kommandozeilenoptionen von Vite

Sie können den Erstellungsprozess einer React-Applikation mit Vite entweder, wie vorgestellt, interaktiv durchlaufen oder Sie geben die entsprechende Auswahl bereits über Kommandozeilenoptionen an. In diesem Fall erhalten Sie keine Rückfragen.

Vite bietet Ihnen die folgenden Optionen:

- `-h`, `--help`: Diese Option zeigt Ihnen eine Liste der verfügbaren Optionen an.
- `-t`, `--template` *NAME*: Mit der Template-Option legen Sie die Grundstruktur Ihrer Applikation fest. Für React sind folgende Templates relevant: `react`, `react-compiler`, `react-swc`, `react-ts`, `react-compiler-ts` und `react-swc-ts`.
- `-i`, `--immediate`: Diese Option sorgt dafür, dass die Abhängigkeiten während des Erstellungsprozesses installiert werden und der Entwicklungsserver nach Abschluss der Initialisierung und Installation gestartet wird.
- `--rolldown` / `--no-rolldown`: Mit dieser Option können Sie festlegen, ob Sie das aktuell noch experimentelle *Rolldown* für Ihre Applikation nutzen möchten. Es ersetzt *Rollup* als Standard-Bundler in Vite.
- `--interactive` / `--no-interactive`: Mit der `--no-interactive`-Option stellen Sie sicher, dass Vite keine Rückfragen stellt. In diesem Fall werden für alle nicht festgelegten Optionen die Standardwerte verwendet.

React Compiler

Der *React Compiler* ist ein Hilfsmittel, der Ihren Code zur Build-Zeit Ihrer Applikation optimiert. Hier geht es vor allem um die Memoisierung von Komponenten und Werten, sodass diese zwischen zwei Renderläufen nicht unnötig erneut erzeugt werden müssen. Mehr über den React Compiler erfahren Sie in Kapitel 18.

SWC (Speedy Web Compiler)

Der SWC ist ein Werkzeug, das für das Kompilieren und Bundeln von JavaScript und TypeScript eingesetzt wird. Es stellt eine Alternative zu Babel dar und ist nach Angaben seiner Entwickler zwischen 20-mal (Single Thread) und 70-mal (vier Kerne) schneller als Babel. Mehr Informationen zu SWC finden Sie auf der offiziellen Webseite des Werkzeugs unter *https://swc.rs/*. Nutzen Sie eines der `swc`-Templates von Vite, ist alles für den Einsatz des Werkzeugs vorbereitet, sodass Sie nichts weiter unternehmen müssen und ohne größeren Aufwand in den Genuss der Performanceverbesserungen kommen.

Aktualisierung einer bestehenden Applikation

Wenn Sie über längere Zeit an der Entwicklung einer Applikation arbeiten, kommen Sie früher oder später in die Situation, dass neue Versionen von React und Vite erscheinen. Damit Sie von den Verbesserungen, aber auch von Sicherheitsupdates profitieren, sollten Sie regelmäßig den Softwarestand Ihrer Applikation prüfen und die Pakete bei Bedarf aktualisieren. Ob eine solche Aktualisierung ansteht, finden Sie heraus, indem Sie das Kommando `npm outdated` im Verzeichnis Ihrer Applikation ausführen. In Listing 2.11 sehen Sie die Ausgabe des Kommandos:

```
$ npm outdated
Package    Current  Wanted  Latest  Location                Depended by
react       18.3.1  18.3.1  19.2.3  node_modules/react      my-app
react-dom   18.3.1  18.3.1  19.2.3  node_modules/react-dom  my-app
vite         5.1.6  5.4.21   7.3.0  node_modules/vite       my-app
```

Listing 2.11: Ausgabe von »npm outdated«

Sind veraltete Versionen von Paketen installiert, führt `npm outdated` sie auf. Die Informationen, die Sie hier erhalten, sind:

- `package`: der Name des veralteten Pakets
- `current`: die Version, die aktuell in der Applikation installiert ist
- `wanted`: die höchste Version, die mit der Angabe in der *package.json*-Datei kompatibel ist
- `latest`: die neueste stabile Version des Pakets
- `location`: der Pfad, in dem das Paket installiert ist
- `depended by`: Gibt an, welches Paket das veraltete Paket benötigt. Im Beispiel ist das die Applikation selbst.

Um nun die aktuellste Version der Pakete zu installieren, können Sie den folgenden Befehl auf der Kommandozeile verwenden:

```
npm install react@latest react-dom@latest vite@latest
```

Listing 2.12: Aktualisieren des »react-scripts«-Pakets

Ein solches Update sollten Sie jedoch nicht leichtfertig durchführen – gerade bei Major-Releases, also einer Aktualisierung der Hauptversionsnummer, kann es zu Breaking Changes kommen, die die Funktionsfähigkeit Ihrer Applikation einschränken können. Das React Team veröffentlicht üblicherweise für jedes größere Release einen ausführlichen Blogartikel, der die neuen Features erklärt. Außerdem gibt es ein Changelog, in dem alle Änderungen festgehalten sind, die eine neue Version einführt. Die jeweiligen Informationen finden Sie unter:

- React Blog: *https://react.dev/blog*
- React Changelog: *https://github.com/facebook/react/blob/main/CHANGELOG.md*
- Vite Changelog: *https://github.com/vitejs/vite/blob/main/packages/vite/CHANGELOG.md*

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

Automatische Aktualisierung des Quellcodes mit »react-codemod«

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

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

2.4.3 Alternativen zu Vite

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

- **Parcel:** Parcel reiht sich ebenfalls in die Reihe der Web-Buildtools ein. Seine Features sind ähnlich umfangreich wie die von Webpack und Vite. Nur beim Setup ist etwas mehr Arbeit notwendig als mit den eleganten Template-Lösungen von Create React

App und Vite. Die Dokumentation von Parcel beinhaltet eine Schritt-für-Schritt-Anleitung für den Aufbau einer React-Applikation. Diese finden Sie unter *https://parceljs.org/recipes/react/*.

- **Rsbuild:** Hierbei handelt es sich um ein hochperformantes, auf Rust basierendes Build-Werkzeug für die Webentwicklung. Rsbuild basiert im Kern auf *Rspack*, einer ebenfalls in Rust geschriebenen Alternative zu Webpack. Rsbuild funktioniert problemlos mit React, Vue oder Svelte und stellt für die Entwicklung, ähnlich wie Vite, Templates für die verschiedenen Umgebungen zur Verfügung. Für die Initialisierung Ihrer Applikation nutzen Sie das Kommando `npm create rsbuild@latest` und starten damit einen interaktiven Assistenten, der Sie durch den Setup-Prozess leitet. Mehr Informationen zu Rsbuild finden Sie auf der offiziellen Webseite des Projekts unter *https://rsbuild.rs/*.

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

2.4.4 Vite Scripts

Neben den Abhängigkeiten, die Sie für die Entwicklung benötigen, erzeugt Vite für Sie einige hilfreiche Scripts in der *package.json*-Datei Ihrer Applikation. Insgesamt gibt es vier verschiedene Scripts mit unterschiedlichen Bedeutungen:

- `dev`: Das Start-Script startet den *Dev-Server* von Vite und führt Ihre Applikation aus. In der Standardkonfiguration erreichen Sie Ihre Applikation unter der URL *http://localhost:5173*. Ist der Port bereits von einer anderen Applikation belegt, wählt Vite automatisch den nächsten freien Port aus. Mit der `--port`-Option können Sie den Port auch selbst festlegen, indem Sie einen anderen Port auswählen.

  In Listing 2.13 sehen Sie, wie Sie mit der `--port`-Option statt Port `5173` alternativ Port `8181` nutzen können:

  ```
  npm run dev -- --port 8181
  ```

  Listing 2.13: Die Applikation auf einem alternativen Port ausführen

  Alternativ dazu können Sie die Portnummer auch in der Konfigurationsdatei von Vite, der *vite.config.js*, festlegen. In Listing 2.14 sehen Sie, wie Sie das Konfigurationsobjekt, das Sie der `defineConfig()`-Funktion übergeben, erweitern können, um die Portnummer zu definieren:

  ```
  import { defineConfig } from 'vite'
  import react from '@vitejs/plugin-react'

  // https://vite.dev/config/
  export default defineConfig({
    server: {
  ```

```
    port: 8181,
  },
  plugins: [react()],
})
```

Listing 2.14: Angabe der Portnummer in der Vite-Konfiguration (»vite.config.js«)

In einer Applikation, die Sie mit Vite und dem React-Template erzeugen, können Sie beispielsweise das ECMAScript-Modulsystem mit den Schlüsselwörtern `import` und `export` verwenden. Damit das in allen Browsern funktioniert, kommt *Rolldown* zum Einsatz. Ein positiver Seiteneffekt der verwendeten Werkzeuge ist, dass die Applikation bei Änderungen automatisch neu geladen wird.

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

- `build`: Bei der Entwicklung sollten Sie Ihre Applikation in möglichst viele kleine Komponenten unterteilen. Jede dieser Komponenten hat genau einen Zweck und liegt in einer separaten Datei. Das `build`-Script dient dazu, die vielen Komponenten- und Hilfsdateien zu einem Applikationsbundle zusammenzupacken und dieses so zu optimieren, dass es möglichst wenig Speicherplatz benötigt, was sich positiv auf die benötigte Bandbreite auswirkt, wenn die Applikation zum Client ausgeliefert wird. Führen Sie dieses Script aus, finden Sie das Ergebnis im *dist*-Verzeichnis Ihrer Applikation.
- `lint`: Mit dem `lint`-Script können Sie Ihren Code mit *ESLint* überprüfen lassen. Das React-Template enthält zu diesem Zweck eine Standardkonfiguration, mit der die Einhaltung der gängigen Best Practices für React-Applikationen sichergestellt wird.
- `preview`: Das `preview`-Script gibt Ihnen eine Vorschau auf die gebaute Produktiv-Applikation. Damit das funktionieren kann, muss das *dist*-Verzeichnis existieren und die gebaute Applikation beinhalten. Sie müssen zuvor also das `build`-Script ausführen. Anschließend erhalten Sie durch den Aufruf des `preview`-Scripts Zugriff auf die Applikation mit dem Produktiv-Stand, ohne dass Sie sie auf das Zielsystem deployen müssen.

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

```
npm run dev
```

Listing 2.15: Starten der Applikation im Entwicklungsmodus

Das Script bereitet die Applikation für den Einsatz vor und startet den Dev-Server. Als Ausgabe erhalten Sie eine entsprechende Erfolgsmeldung (siehe Listing 2.16):

```
$ npm run dev

> my-app@0.0.0 dev
> vite


  VITE v7.3.0  ready in 130 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

Listing 2.16: Ausgabe von »npm run dev«

Ist Ihre Applikation gestartet, können Sie sie im Browser öffnen. Als Ergebnis erhalten Sie die Ansicht aus Abbildung 2.4.

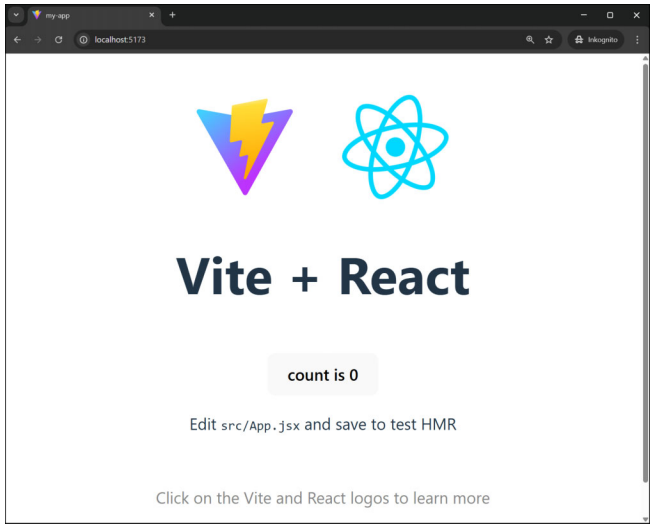

Abbildung 2.4: Die initiale React-Applikation

Achtung

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

2.4.5 Serverkommunikation im Entwicklungsbetrieb

Bei Single-Page-Applikationen ist die Trennung zwischen Frontend und Backend in der Regel sehr strikt. Auf der einen Seite steht die React-Applikation im Browser und auf der anderen Seite eine Serverschnittstelle, die mit einer beliebigen Programmiersprache umgesetzt sein kann. Während der Entwicklung nutzen Sie den Dev-Server, um Ihre Frontend-Applikation auszuliefern. Das serverseitige Backend kümmert sich in einer solchen Applikation um das Speichern von Daten und um weitere Aspekte, beispielsweise um die Authentifizierung von Benutzern. Der Dev-Server liefert das Backend nicht aus. Es läuft als separater Serverprozess. Das bedeutet, dass das Frontend und das Backend über zwei verschiedene URLs erreichbar sind. Sicherheitsmechanismen (in diesem Fall *CORS*, das *Cross Origin Resource Sharing*) im Browser verhindern, dass Sie ohne Weiteres zwischen verschiedenen Servern kommunizieren können.

In Listing 2.17 sehen Sie den Quellcode einer Komponente, die mit dem Backend kommuniziert. (Mehr Informationen zu den Implementierungsdetails finden Sie in den nächsten Kapiteln. Hier geht es lediglich darum, dass Sie das Proxy-Feature von Vite kennenlernen.)

```
import { useEffect, useState } from 'react';

function App() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    fetch('http://localhost:3000/users')
      .then((response) => response.json())
      .then((data) => setUsers(data));
  }, []);

  return (
    <ul>
      {users.map((user) => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}

export default App;
```

Listing 2.17: Serverkommunikation mit einem Backend (»src/App.jsx«)

Ausschlaggebend ist hier der Aufruf der `fetch()`-Funktion in Kombination mit einem Backend, das den Access-Control-Allow-Origin-Header nicht korrekt gesetzt hat. Ist dies der Fall, erhalten Sie die Fehlermeldung aus Listing 2.18:

```
Access to fetch at 'http://localhost:3000/users' from origin 'http://
localhost:5173' has been blocked by CORS policy: No 'Access-Control-
Allow-Origin' header is present on the requested resource.
```

Listing 2.18: CORS-Fehlermeldung

Diesem Fehler können Sie auf zwei Arten begegnen: Entweder sorgen Sie dafür, dass der Server den korrekten Header setzt, oder Sie verwenden den Vite-Proxy. Der Proxy hat einen weiteren Vorteil: Wenn Ihr Frontend in Ihrer Produktivumgebung die Rolle eines Proxys einnimmt, stimmt Ihr Entwicklungs-Setup mit dem Produktiv-Setup überein.

Um den Proxy zu aktivieren, passen Sie die *vite.config.js* so an wie in Listing 2.19:

```
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';

// https://vite.dev/config/
export default defineConfig({
  plugins: [react()],
  server: {
    proxy: {
      '/api': {
        target: 'http://localhost:3000',
        changeOrigin: true,
        rewrite: (path) => path.replace(/^\/api/, ''),
      },
    },
  },
});
```

Listing 2.19: Aktivieren der Proxy-Funktionalität in der Vite-Konfiguration (»vite.config.js«)

Diese Erweiterung der Konfiguration sorgt dafür, dass der Dev-Server alle Anfragen an die URL *http://localhost:5173/api/* an *http://localhost:3000* umleitet. Wenn Sie den `fetch()`-Aufruf wie folgt anpassen

```
fetch('/api/users')
```

Listing 2.20: Angepasster »fetch«-Aufruf (»src/App.jsx«)

dann wird die Anfrage an *http://localhost:3000/users* weitergeleitet und Sie sehen die Benutzerliste ohne Fehlermeldung im Browser.

Mehr zum Thema Serverkommunikation erfahren Sie in Kapitel 3, »Die Grundlagen von React«.

2.5 Die Struktur der Applikation

Das React-Template von Vite erzeugt eine Reihe von Dateien und Verzeichnissen für Sie. Damit Sie sich in der erzeugten Struktur besser zurechtfinden, erhalten Sie in diesem Abschnitt eine kurze Erklärung zu den verschiedenen Dateien und Verzeichnissen.

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

- *.gitignore*: Diese Datei listet die Dateien und Verzeichnisse auf, die nicht ins Git-Repository eingecheckt werden sollen. Hierunter fällt beispielsweise das *node_modules*-Verzeichnis.
- *eslint.config.js*: Vite liefert ESLint für die statische Codeanalyse mit aus und stellt in der *eslint.config.js* die Basiskonfiguration zur Verfügung. Hier können Sie die verwendeten Regeln erweitern und sie an Ihre Bedürfnisse anpassen. Weitere Informationen zu ESLint finden Sie unter *https://eslint.org/*.
- *index.html*: Hierbei handelt es sich um die Einstiegsdatei in Ihre Applikation. In dieser Datei finden Sie ein `div`-Element mit der `id` `root`, in das React Ihre Applikation einfügt. Diese Datei bindet außerdem die *main.jsx*-Datei und damit den JavaScript-Quellcode Ihrer Applikation ein.
- *package-lock.json*: In der *package-lock.json*-Datei werden die exakten Versionen und die Integritäts-Hashes der installierten Abhängigkeiten und deren Unterabhängigkeiten festgehalten. Dies stellt sicher, dass die Installationen Ihrer Applikation auf allen Systemen zu jedem Zeitpunkt gleich sind. Außerdem stellt der Paketmanager durch die Prüfsummen der Pakete sicher, dass keine manipulierten Pakete eingeschleust werden können.
- *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 im Markdown-Format. In der initialen Version dieser Datei finden Sie eine grundlegende Beschreibung von Vite und den React-Plugins sowie den React Compiler und eine Erklärung, wie Sie die ESLint-Konfiguration erweitern können. In Ihrer Applikation sollten Sie versuchen, in dieser Datei alles zu dokumentieren, was Entwickler benötigen, um an der Entwicklung der Applikation zu arbeiten. Dazu gehören das Setup der Entwicklungsumgebung, der Build- und Releaseprozess und Besonderheiten der Applikation. Das Ziel sollte sein, dass neue Kollegen nur die Readme durchlesen müssen und danach produktiv mitarbeiten können.
- *vite.config.js*: Diese Datei enthält die Konfiguration für Vite, also Ihr Build-Werkzeug. Mit dieser Datei sind Sie bereits bei der Konfiguration des Proxy-Servers in Berührung gekommen. In den meisten Fällen reicht jedoch die Standardkonfiguration von Vite aus, sodass Sie mit dieser Datei wenig zu tun haben werden.

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

- *node_modules*: In diesem Verzeichnis werden alle installierten NPM-Pakete in einer flachen Hierarchie vorgehalten. Das *node_modules*-Verzeichnis ist dem Paketmanager vorbehalten, sodass Sie die Inhalte nicht von Hand manipulieren sollten, damit diese Änderungen nicht an andere Entwickler verteilt werden und so verloren gehen. Dieses Verzeichnis wird normalerweise von der Versionskontrolle ausgeschlossen.
- *public*: 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 Vite so die Möglichkeit hat, die Inhalte zu optimieren.

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

  Das React-Template von Vite gibt hier eine leichtgewichtige Struktur vor, die aus einer Reihe von Dateien besteht. Die *main.jsx*-Datei bildet den Einstieg in den JavaScript-Teil Ihrer Applikation und rendert die Wurzelkomponente, die sich wiederum in der Datei *App.jsx* befindet. *App.css* enthält Style-Definitionen, die in der *App.jsx* geladen werden. In der *index.css*-Datei finden Sie allgemeine Style-Definitionen, beispielsweise für das `body`-Element Ihrer Applikation. Das *assets*-Verzeichnis kann weitere statische Assets enthalten, die Sie im Quellcode Ihrer Applikation importieren können.

2.6 Fehlersuche in einer React-Applikation

Ein in der Entwicklung häufig unterschätztes Hilfsmittel ist der Webbrowser. Ihn können Sie nicht nur zur Anzeige Ihrer Applikation verwenden, sondern auch aktiv in den Entwicklungsprozess einbinden. Moderne Webbrowser verfügen über eine Reihe hilfreicher Werkzeuge zur Analyse und Fehlersuche in einer Applikation. Allen voran ist der Debugger zu nennen. Mit der Tastenkombination Cmd+Alt+i auf dem Mac beziehungsweise Strg+Alt+i oder F12 auf einem Windows- oder Linux-System öffnen Sie die Entwicklerwerkzeuge des Browsers. In Abbildung 2.5 sehen Sie die Ansicht eines Browsers, der an einem Breakpoint in der Applikation angehalten hat.

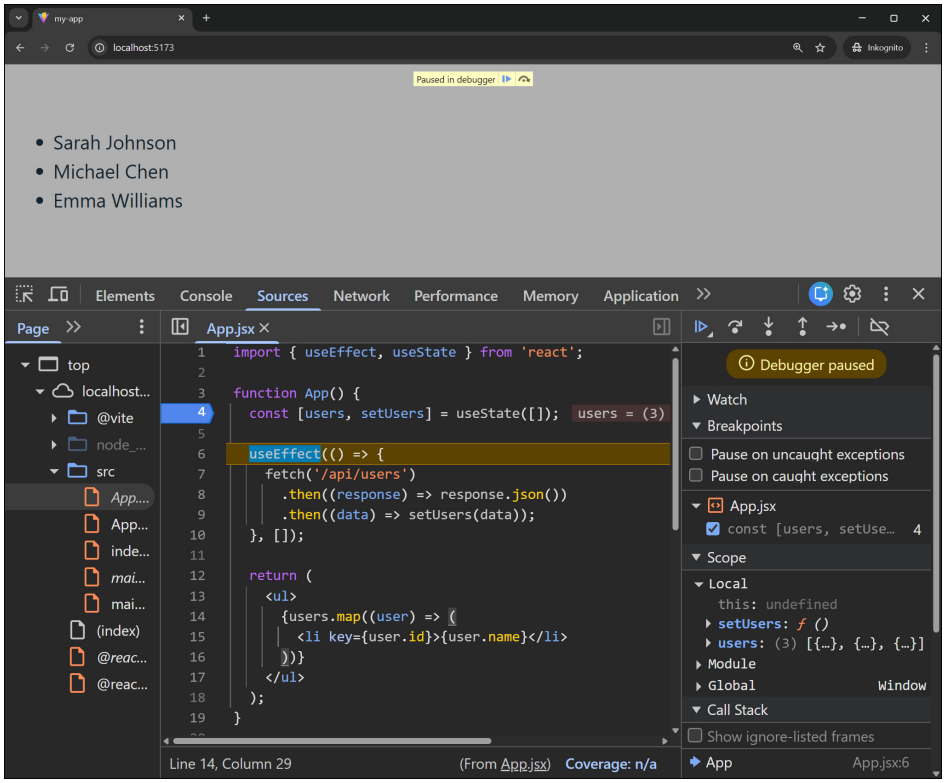

Abbildung 2.5: Angehaltener Debugger in einer React-Applikation

Um Ihre Applikation zu debuggen, öffnen Sie die Entwicklerwerkzeuge, wechseln auf den **Sources**-Tab und setzen einen Breakpoint, indem Sie auf die Zeilennummer klicken. Ein Rechtsklick auf die Zeilennummer bietet Ihnen außerdem die Möglichkeit, bedingte Breakpoints zu setzen, bei denen der Browser nur anhält, wenn eine zuvor definierte Bedingung erfüllt ist. Eine weitere Möglichkeit, einen Breakpoint zu setzen, ist die Verwendung des `debugger`-Statements. Hierfür schreiben Sie im Quellcode Ihrer Applikation an der gewünschten Stelle das Schlüsselwort `debugger`. Sind die Entwicklerwerkzeuge Ihres Browsers geöffnet, wird die Ausführung an dieser Stelle angehalten.

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

- **Resume**: Pausiert der Debugger an einem Breakpoint, kann der Ablauf des Scripts mit dieser Schaltfläche fortgesetzt werden.
- **Step over**: Überspringt den nächsten Funktionsaufruf.
- **Step into**: Springt in die aufzurufende Funktion hinein. Diese Aktion fügt einen Eintrag zum Callstack hinzu.

- **Step out**: Springt aus der aktuellen Funktion heraus. Mit dieser Aktion wird der aktuelle Eintrag vom Callstack entfernt.
- **Step**: Diese Aktion springt in die nächste Zeile.
- **Deactivate Breakpoints**: Deaktiviert alle aktuell gesetzten Breakpoints, sodass die Ausführung ohne Unterbrechung fortgesetzt wird.
- **Pause on Exception**: Mit dieser Schaltfläche sorgen Sie dafür, dass die Ausführung pausiert wird, sobald eine Exception geworfen wird.

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

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

2.6.1 Arbeiten mit den React Developer Tools

Der Debugger ist allerdings nicht das einzige Hilfsmittel, auf das Sie bei der Entwicklung einer React-Applikation zugreifen können. Sowohl für Firefox als auch für Chrome existiert eine Browsererweiterung, die Ihnen die Struktur und die dynamischen Daten Ihrer Applikation anzeigen kann. Für Chrome finden Sie die *React Developer Tools* im Chrome Web Store unter der URL *https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi*. Für Firefox ist das Werkzeug unter *https://addons.mozilla.org/en-US/firefox/addon/react-devtools/* verfügbar.

Nach der Installation finden Sie in den Entwicklerwerkzeugen Ihres Browsers zwei Tabs mit den Bezeichnungen **Profiler** und **Components**. Mit dem **Profiler** können Sie Leistungsdaten zur Laufzeit Ihrer Applikation aufnehmen und anschließend auswerten. Im **Components**-Tab sehen Sie die aktuelle Komponentenstruktur Ihrer Applikation und können die einzelnen Komponenten inspizieren sowie deren Props und State überprüfen. Wie die React Dev Tools im Browser aussehen, zeigt Abbildung 2.6.

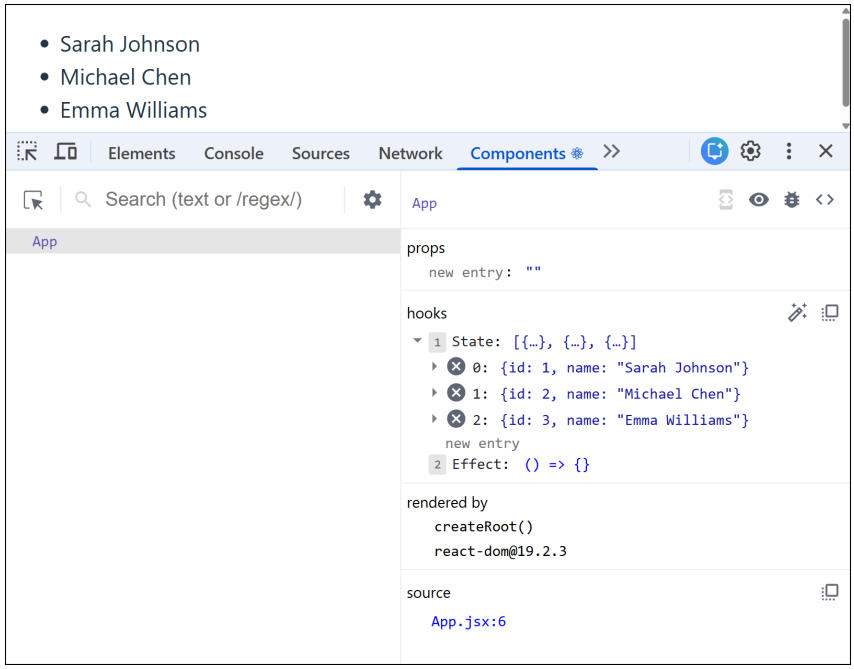

Abbildung 2.6: Die »React Dev Tools« in Chrome

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

2.7 Die Applikation bauen

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

Als Ergebnis wird ein neues Verzeichnis mit dem Namen *dist* erstellt. In diesem Verzeichnis befinden sich sämtliche Ressourcen, die Sie benötigen, um die Applikation an Ihre Benutzer auszuliefern. Den Inhalt dieses Verzeichnisses können Sie in das Document-Root-Verzeichnis eines Webservers kopieren und so Ihre Applikation deployen.

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

2.8 Zusammenfassung

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

- Sie können React sowohl in bestehende Webseiten einbinden als auch eine React-Applikation von Grund auf neu implementieren.
- Mit Playgrounds wie *CodePen* können Sie einfache Experimente mit React durchführen, indem Sie die Bibliothek direkt einbinden. Diese Variante eignet sich allerdings nicht für umfangreiche Projekte.
- React lässt sich sehr leichtgewichtig durch direkte Einbindung der Bibliotheksdateien in eine statische HTML-Seite integrieren. Hierbei verzichten Sie jedoch auf den Komfort einer vorgefertigten Struktur und konfigurierter Werkzeuge.
- *Vite* ist das Werkzeug der Wahl, wenn es um das Setup größerer Anwendungen geht. Auf der generierten Struktur können Sie auch umfangreiche Applikationen umsetzen.
- Das React-Template von Vite bereitet vier Scripts für Sie vor, mit denen Sie den Dev-Server starten, Ihren Code statisch überprüfen, Ihre Applikation bauen und das Ergebnis des Build-Prozesses prüfen können. Diese Scripts sind in der *package.json*-Datei Ihrer Applikation gespeichert.
- Das Verhalten von Vite können Sie über Kommandozeilenoptionen, Umgebungsvariablen und die Vite-Konfiguration in Form der *vite.config.js*-Datei beeinflussen.
- Alle modernen Browser verfügen über leistungsstarke Entwicklerwerkzeuge, die Ihnen bei der Fehlersuche helfen. Außerdem existieren mit den *React Developer Tools* Erweiterungen, die Ihnen bei der Analyse einer React-Applikation helfen.

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

Kapitel 3
Die Grundlagen von React

In diesem Kapitel lernen Sie die wichtigsten Bausteine von React kennen: Komponenten, den State, Datenflüsse zwischen Komponenten und das Reagieren auf Events.

Alle populären Single-Page-Frameworks – wie Angular, Vue, aber auch React – verfolgen beim Aufbau einer Applikation ein ähnliches Konzept: Die Oberfläche wird in kleinere Einheiten, die *Komponenten*, heruntergebrochen. React bietet im Vergleich zu den beiden anderen Frameworks den Vorteil, dass Komponenten in React sehr leichtgewichtig sind und es Ihnen damit einfach gemacht wird, viele kleine und in sich geschlossene Komponenten zu erzeugen.

In diesem Kapitel beschäftigen wir uns mit den wesentlichen Aspekten von Komponenten in React. Sie erfahren, wie Sie den State Ihrer Applikation organisieren und wie die Informationen durch den Komponentenbaum fließen.

3.1 Vorbereitung

Falls Sie an dieser Stelle noch nicht über eine React-Applikation verfügen, sollten Sie sich jetzt um die Initialisierung kümmern, damit Sie die Konzepte selbst ausprobieren können. Wechseln Sie hierfür auf die Konsole Ihres Systems und führen Sie die folgenden Kommandos aus. Sie erzeugen damit eine neue Applikation mit dem Namen `my-app`, wechseln in das Verzeichnis und starten die Applikation:

```
npm create vite@latest my-app -- --template react --no-interactive
cd my-app
npm install
npm run dev
```

Listing 3.1: Erzeugen und Starten der Applikation

Nach dem Start können Sie Ihre Applikation im Browser über die URL *http://localhost:5173* öffnen und sehen die initiale Komponente. Mehr zum Thema »Setup und Initialisierung« finden Sie in Kapitel 2, »Die ersten Schritte im Entwicklungsprozess«.

Am Ende dieses Kapitels haben Sie eine Applikation mit einer einfachen Komponentenhierarchie implementiert. Als Vorlage dient uns das Mockup aus Abbildung 3.1.

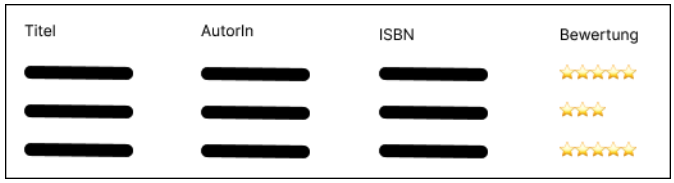

Abbildung 3.1: Mockup der Applikation

Die Applikation aufräumen

Einen Teil der Strukturen, die Vite für Sie erzeugt hat, benötigen Sie nicht für Ihre Applikation. Sie sollten stets darauf achten, keine unnötigen Strukturen in Ihrer Applikation zu haben. Der erste Schritt besteht also darin, in der Applikation aufzuräumen.

Folgende Dateien können Sie löschen:

- *src/assets/react.svg*: Das React-Icon wird lediglich in der App-Komponente angezeigt.
- *public/vite.svg*: Das Vite-Icon kommt an zwei Stellen zum Einsatz. Zum einen wird es von der App-Komponente gerendert, und zum anderen dient es als Favicon. Um es sauber zu entfernen, müssen Sie in der *index.html*-Datei das `link`-Element entfernen.

Die Datei *public/vite.svg* sollte erhalten bleiben, da sie als Favicon verwendet wird, das der Browser im Titel des Tabs anzeigt.

Die Styles der Demo-Anwendung in den Dateien *src/App.css* und *src/index.css* können Sie löschen, indem Sie die Dateien leeren.

Als Nächstes werfen wir einen Blick auf die bestehenden Strukturen. Die beiden wichtigsten Dateien im *src*-Verzeichnis sind *main.jsx* und *App.jsx*. Eine in Projekten häufig verwendete Konvention sieht vor, dass Dateien, die JSX-Strukturen enthalten, mit der Endung *.jsx* statt mit *.js* versehen werden.

3.2 Einstieg in die Applikation

Die *main.jsx* ist der Startpunkt der Applikation und für das Rendering zuständig. In der *App.jsx* finden Sie die Wurzelkomponente der Applikation, also die Basis des Komponentenbaums.

3.2.1 »main.jsx« – das Rendering der Applikation

Der Quellcode der *main.jsx*-Datei sorgt dafür, dass die Applikation dargestellt wird.

Schnellstart

Die *main.jsx*-Datei ist der Einstieg in eine React-Applikation. Sie erfüllt die folgenden Aufgaben: