Für Studium und Beruf

HTML & CSS, JavaScript, Frameworks, Security, Web Performance

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>HTML-Event-Handler</title>
  <link rel="stylesheet" href="styles/main.css">
  <script src="scripts/main.js"></script>
</head>
<body>
<div>
<label for="age"></label>
<input id="age" type="number" value="0" onblur="che
</div>
  <div id="output">
</body>
```

Philip Ackermann

Fullstack-Entwicklung

Das Handbuch für Webentwickler

- Grundlagen, Anforderungen, Best Practices
- HTML, CSS, JavaScript, PHP, Web-APIs, Webarchitekturen
- Webservices, Datenbanken, DevOps, Testing, Deployment und Hosting

2., aktualisierte und erweiterte Auflage

Rheinwerk Computing

Vorwort

Wenn man heute Webanwendungen entwickelt, gibt es zwar mehr Möglichkeiten und bessere Tools als vor 20 Jahren. Die Anforderungen an Entwickler sind allerdings ebenfalls gestiegen. Kaum eine Stellenanzeige für Webentwickler, in der nicht das Wort »Fullstack« verwendet wird und in der nicht zahlreiche Kenntnisse in verschiedensten Bereichen gefordert werden. Kurz: »Fullstack-Entwickler« sind gefragt wie nie zuvor.

Die Anforderungen an solche Fullstack-Entwickler sind jedoch enorm. Kein Wunder, bezeichnet »Fullstack« doch den gesamten Technologiestack, den es zu beherrschen gilt, um eine Webanwendung »von vorne bis hinten« – also vom Frontend bis zum Backend – zu entwickeln und dann auch noch für den Produktiveinsatz vorzubereiten.

Ich erinnere mich noch gut daran, als ich selbst mit der Webentwicklung das erste Mal in Berührung kam. Das war um die Jahrtausendwende – also vor etwa 20 Jahren –, noch vor meinem Informatikstudium. Während meiner Ausbildung zum Mediengestalter verschlang ich ganze Bücher zu den Themen HTML, CSS und JavaScript, arbeitete Onlinetutorials durch und entwickelte, was das Zeug hielt. Diese drei – noch heute – wichtigen Sprachen für das Web reichten mir damals als Webentwickler aus.

Im Studium und während meiner Tätigkeit als studentische Hilfskraft beim Fraunhofer-Institut für Angewandte Informationstechnik kam ich dann das erste Mal mit einer »richtigen« Programmiersprache – Java – in Berührung und lernte nach und nach hierüber die weiteren Ebenen einer Webanwendung kennen: die Ebenen auf der Serverseite. Meine Aufgabe: der Umbau einer Java-basierten Desktopsoftware hin zu einer auf Webservices basierenden Webanwendung.

Später dann – Node.js sei Dank – implementierten wir die Anwendung fast komplett neu mit JavaScript, das sich mittlerweile zu einer ernst zu nehmenden Programmiersprache für die Serverseite gewandelt hatte. Außerdem tauschten wir die vormals relationale Datenbank aufgrund der neuen Anforderungen gegen eine nicht-relationale Datenbank.

Seit nunmehr knapp sechs Jahren bin ich – mittlerweile in der Rolle des CTO – bei der Cedalo GmbH beschäftigt, bei der wir mehrere Softwareprodukte entwickeln. Auch hier beschäftige ich mich nach wie vor mit allen Ebenen, sprich mit dem gesamten »Stack« der Software. Ob es jetzt darum geht, die richtige Datenbank für den jeweiligen Anwendungsfall auszuwählen oder die richtige Schnittstelle für einen Webservice zu definieren. All das sind Fragestellungen, mit denen ich mich tagtäglich konfrontiert sehe. Und damit die fertigen Softwareprodukte auch anständig für das

Produktivsystem vorbereitet werden, sind mittlerweile auch DevOps-Themen zu meinem Werkzeugkasten hinzugekommen.

Warum ich Ihnen das alles erzähle? Sicher nicht, damit ich Ihnen meinen Lebenslauf vorstelle. Sondern weil genau die genannten Themen – HTML, CSS, JavaScript, Webservices, Datenbanken, DevOps – mittlerweile Anforderungen an einen Fullstack-Entwickler sind. Klar, dass es da schwerfällt zu entscheiden, wo man anfangen und womit genau man sich beschäftigen sollte.

Dieses Buch soll Ihnen eine Roadmap und ein Ratgeber gleichermaßen sein. Mein Wissen aus mehr als 20 Jahren in der Web- und Softwareentwicklung möchte ich dazu nutzen, Ihnen die wichtigsten Grundlagen für die Fullstack-Entwicklung kompakt zu vermitteln; Ihnen zum einen die Angst vor der Anzahl an Themen zu nehmen und zum anderen zu vermitteln, welche Themen wirklich wichtig sind.

Sei es bezogen auf die Frontend-Entwicklung oder auf die Backend-Entwicklung, sei es bei der Implementierung von Webservices oder bei der Auswahl der richtigen Datenbank, sei es bei Themen wie der Sicherheit, dem automatisierten Testen oder dem Deployment von Webanwendungen. In diesem Buch finden Sie alles, was Sie als Fullstack-Entwickler oder -Entwicklerin benötigen.

Für wen ist dieses Buch?

Das Buch richtet sich vor allem an Einsteiger in die Webentwicklung, die einen umfassenden Überblick über die Fullstack-Entwicklung gewinnen möchten. Hierzu benötigen Sie keine Vorkenntnisse in HTML, keine Vorkenntnisse in CSS und auch keine Programmierkenntnisse. Dieses Buch dient dazu, Sie auf dem Weg zum Fullstack-Entwickler zu begleiten und Ihnen die wichtigen Technologien effektiv zu vermitteln.

Wie ist dieses Buch aufgebaut?

Insgesamt besteht das Buch aus 23 Kapiteln zu Themen, die ich als besonders wichtig empfinde, wenn es um die Webentwicklung und Fullstack-Entwicklung geht. Eine etwas genauere Beschreibung zu den einzelnen Kapiteln finden Sie in Kapitel 1, »Die Grundlagen verstehen«. Wenn Sie also noch vor der Kaufentscheidung stehen und ich Sie bis hierhin noch nicht überzeugen konnte, wäre jetzt eine gute Gelegenheit, dort kurz nachzuschauen. ☺

Den Quelltext zum Buch finden Sie übrigens auf der offiziellen Website zum Buch unter *https://www.rheinwerk-verlag.de/webentwicklung_5559*. Alternativ dazu steht

der Quelltext auch in einem GitHub-Repository unter *https://github.com/cleancode-rocker/webhandbuch* zur Verfügung.

Wie sollte ich das Buch durchlesen?

Ich rate Ihnen dazu, das Buch von vorne bis hinten, Kapitel für Kapitel, durchzuarbeiten. Auf diese Weise können Sie am besten den Zusammenhang der einzelnen Themen nachvollziehen, und es ist sichergestellt, dass Ihnen nichts Wichtiges entgeht.

Natürlich soll das Buch auch als Ratgeber und Nachschlagewerk dienen, das Sie immer wieder gerne aus dem Regal ziehen, um Wissen bei Bedarf aufzufrischen. Mit den sprechenden Überschriften und dem Index sollten Sie das jeweils gesuchte Thema schnell wiederfinden.

Danksagung

Das Schreiben eines Buches ist immer mit allerhand Arbeit verbunden. An dieser Stelle möchte ich all denen danken, die dieses Buchprojekt möglich gemacht und dabei geholfen haben.

Am allermeisten möchte ich meiner Frau und meinen Kindern danken, für ihre Geduld und Unterstützung während der Zeit, die ich an diesem Buch gearbeitet habe.

Außerdem bedanke ich mich bei meinem Lektor Stephan Mattescheck für die wie immer sehr professionelle und freundliche Zusammenarbeit sowie beim gesamten beteiligten Team im Rheinwerk Verlag. Sascha Kersken und Sebastian Springer gilt mein Dank für ihr wie immer äußerst nützliches Fachgutachten und die vielen hilfreichen Hinweise und Anregungen.

Auch Ihnen möchte ich danken, zum einen natürlich für den Kauf dieses Buches, zum anderen aber auch für die Zeit, die Sie mit dem Lesen und Durcharbeiten verbringen werden. Ich hoffe, Sie haben viel Freude dabei und lernen viel Neues.

Über Feedback zum Buch würde ich mich natürlich auch sehr freuen. Unter *info@philipackermann.de* stehe ich Ihnen gerne für Fragen und Anregungen zur Verfügung. Unter *https://www.rheinwerk-verlag.de/webentwicklung_5559* oder *www.webdevhandbuch.de* finden Sie zudem weitere Informationen und Updates zum Buch.

Philip Ackermann
Rheinbach

Kapitel 1
Die Grundlagen verstehen

In diesem Kapitel geht es zunächst darum, Ihnen einen Überblick über die Webentwicklung zu geben und die wichtigsten Begrifflichkeiten zu erläutern.

Jeder spricht von Fullstack-Entwicklung, doch um zu verstehen, was das eigentlich ist, müssen wir uns erst mal mit den Grundlagen beschäftigen. Im ersten und im zweiten Teil dieses Kapitels geht es genau um diese Grundlagen, bevor wir uns im dritten Teil des Kapitels dem Begriff *Fullstack-Entwicklung* widmen.

1.1 Begrifflichkeiten

In diesem Teil möchte ich Ihnen zunächst eine Übersicht über wichtige Begrifflichkeiten geben, die in der Webentwicklung relevant sind. Anschließend schauen wir uns dann den grundsätzlichen Aufbau von Webanwendungen an.

1.1.1 Client und Server

Webseiten und *Webanwendungen* (Definition und Unterschied siehe Kasten) bestehen aus einem Teil, der auf der *Clientseite* ausgeführt wird (dem *Frontend*), und einem Teil, der auf der *Serverseite* ausgeführt wird (dem *Backend*). Auf Serverseite sorgt ein *Webserver* dafür, dass die Webseite bereitgestellt wird. Auf Clientseite greift man über einen *Webclient* (auch nur *Client* oder auch *User Agent* genannt) auf die Webanwendung zu. In der Regel handelt es sich dabei um einen *Webbrowser* (kurz: *Browser*), aber es gibt auch noch andere Arten von Clients wie beispielsweise *Screenreader*, kommandozeilenbasierte bzw. programmatisch gesteuerte *HTTP-Clients* oder sogenannte *Headless Browser*, die keine grafische Oberfläche haben.

Ruft man im Browser eine Webseite auf, ist der Ablauf dabei folgender: Auf Clientseite gibt der Nutzer im Browser die Adresse ein (auch *URL* für *Uniform Resource Locator*, siehe auch nächster Abschnitt) und bestätigt mit entsprechender Taste bzw. entsprechendem Button im Browser das »Laden« der Webseite. Der Browser generiert daraufhin im Hintergrund eine *Anfrage*, die über das *HTTP-Protokoll* (Kapitel 5, »Webprotokolle verwenden«) an den Server gesendet wird. Diese Anfrage nennt man

auch *HTTP-Anfrage* oder *HTTP-Request*. Auf Serverseite nimmt der Webserver die Anfrage entgegen und generiert eine passende *Antwort* (*HTTP-Antwort* bzw. *HTTP-Response*), die er dann an den Client zurückschickt. Der Browser wiederum nimmt die Antwort entgegen und *rendert*, sprich, visualisiert die Webseite. Eventuell benötigte Ressourcen wie Bilder etc. lädt der Browser dabei automatisch nach, damit sie dargestellt werden können.

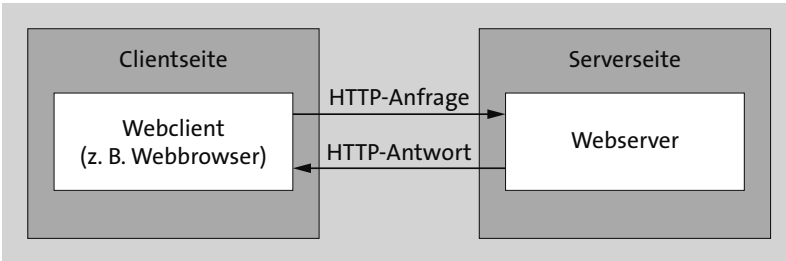


Abbildung 1.1 Das Prinzip von Client und Server

Begriffsdefinition

Ich verwende in diesem Buch immer wieder die Begriffe *Webseite*, *Website* und *Webanwendung*, die oftmals fälschlicherweise synonym verwendet werden. Daher an dieser Stelle eine kurze Klarstellung und Definition: Eine *Webseite* bezeichnet ein einzelnes HTML-Dokument, das unter einer bestimmten URL abgerufen werden kann (*HTML* für *Hypertext Markup Language*, dazu gleich mehr), eine *Website* dagegen ist eine Zusammenfassung verschiedener solcher einzelner Webseiten, beispielsweise die Website zum Buch *https://www.webdevhandbuch.de/* oder die Verlagswebsite *https://www.rheinwerk-verlag.de/*. Bei einer *Webanwendung* dagegen handelt es sich um eine Website, die sich eher wie eine Desktopanwendung anfühlt. Beispiele hierfür sind Google Documents und Google Sheets. Ein synonymer Begriff für solche (vor allem interaktiven und mitunter komplexen) Webanwendungen ist auch *Rich Internet Application*.

1.1.2 Zusammenhang von URLs, Domains und IP-Adressen

Die Adresse, die man in die Adressleiste des Browsers eingibt, wird wie erwähnt als *URL* (*Uniform Resource Locator*) bezeichnet. Beispiele für URLs sind:

- *https://www.philipackermann.de/*
- *https://www.philipackermann.de/static/img/profile.jpg*
- *https://www.webdevhandbuch.de/static/styles/styles.css*
- *https://www.webdevhandbuch.de/service/api/users?search=max*

Eine URL besteht dabei aus verschiedenen Teilen:

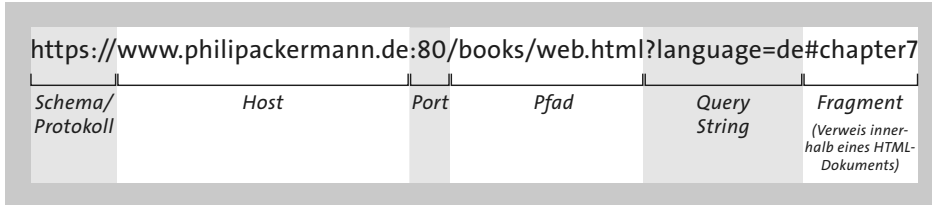


Abbildung 1.2 Aufbau von URLs

- **Protokoll/Schema**: Definiert das zu verwendende Protokoll. Mögliche Protokolle sind beispielsweise:
  - **HTTP (Hypertext Transfer Protocol)**: Für das Übertragen von Webseiten wird das Protokoll HTTP verwendet oder dessen sichere Variante *HTTPS* (*Hypertext Transfer Protocol Secure*).
  - **FTP (File Transfer Protocol)**: Dieses Protokoll dient dem Übertragen von Dateien zu oder von einem FTP-Server.
  - **SMTP (Simple Mail Transfer Protocol)**: Dieses Protokoll kommt für das Übertragen von E-Mails zum Einsatz.
- **Host** (auch *Hostname*): Identifiziert den Webserver eindeutig. Der Host besteht dabei aus *Subdomain*, *Domain* und *Top-Level-Domain*. Der Host »www.philipackermann.de« beispielsweise besteht aus der Subdomain »www«, der Domain »philipackermann« und der Top-Level-Domain »de«.
- **Port**: Gibt an, über welchen »Kanal« auf den Webserver zugegriffen werden soll. In der Regel werden Sie diesen Teil beim »normalen Browsen« nicht sehen, da die *Standardports* (beispielsweise 80 für HTTP oder 443 für HTTPS, siehe auch *https://de.wikipedia.org/wiki/Liste_der_standardisierten_Ports*) von den Browsern in der Adressleiste nicht angezeigt werden. Für die lokale Entwicklung und insbesondere für die Entwicklung von Webservices (Kapitel 13, »Programmiersprachen auf der Serverseite verwenden«) werden Sie es allerdings häufiger mit individuellen Ports zu tun haben. Beispielsweise könnte eine URL eines lokal auf Ihrem Rechner laufenden Webservices so aussehen: »*http://localhost:8080/myservice/api/users*«.
- **Pfad** (auch *Path*): Gibt den Pfad auf dem Webserver zu der angefragten Datei an (bzw. allgemeiner zu der angefragten *Ressource*, weil es sich nicht immer um eine Datei im physischen Sinne handeln muss). In der URL »*https://www.philipackermann.de/static/img/profile.jpg*« beispielsweise ist »/static/img/profile.jpg« der Pfad. Der Pfadtrenner ist dabei immer ein vorwärts gerichteter Schrägstrich (*Slash*).
- **Query String**: Hierüber können in Form von Schlüssel-Wert-Paaren zusätzliche Informationen übergeben werden, anhand derer der Webserver die HTTP-Ant-

wort generieren kann. Der Query String wird dabei über ein Fragezeichen eingeleitet, die einzelnen *Query-String-Parameter* sind durch ein kaufmännisches »Und« verbunden und die Schlüssel und Werte jeweils durch ein Gleichheitszeichen getrennt, also zum Beispiel »*https://www.philipackermann.de/example?search=javascript&display=list*«.

- **Fragment**: Hierüber können Sie gezielt eine bestimmte Stelle innerhalb der jeweiligen Webseite »ansteuern«, sodass der Browser beim Laden der Webseite direkt dorthin »springt«. Eingeleitet wird dieser Teil durch das #-Zeichen, also beispielsweise »*https://www.philipackermann.de/example#chapter5*«.

IP-Adressen und DNS

Jeder Webserver hat eine eindeutige Adresse, über die er erreichbar ist: die sogenannte *IP-Adresse*, wobei mittlerweile zwischen *IPv4* und *IPv6* unterschieden wird. Technisch gesehen handelt es sich im Falle von IPv4 um 32-stellige und im Falle von IPv6 um 128-stellige Binärzahlen, beispielsweise 192.0.2.45 (IPv4) oder 0:0:0:0:0:ffff:c000:22d (IPv6). Da sich aber niemand diese kryptisch anmutenden Adressen merken kann, gibt es sogenannte *DNS Server* (*Domain Name System*), die ein Mapping von Hostnamen auf IP-Adressen enthalten und für einen Hostnamen die IP-Adresse des entsprechenden Webservers zurückgeben.

Wenn Sie also im Browser eine URL (und als Teil davon den Hostnamen) eingeben, fragt der Browser – bevor er die Anfrage an den eigentlichen Webserver stellen kann – zunächst einen DNS-Server an, um auf Basis des Hostnamens an die IP-Adresse des Webservers zu gelangen. Erst wenn er diese hat, stellt er die eigentliche Anfrage an den Webserver (Abbildung 1.3).

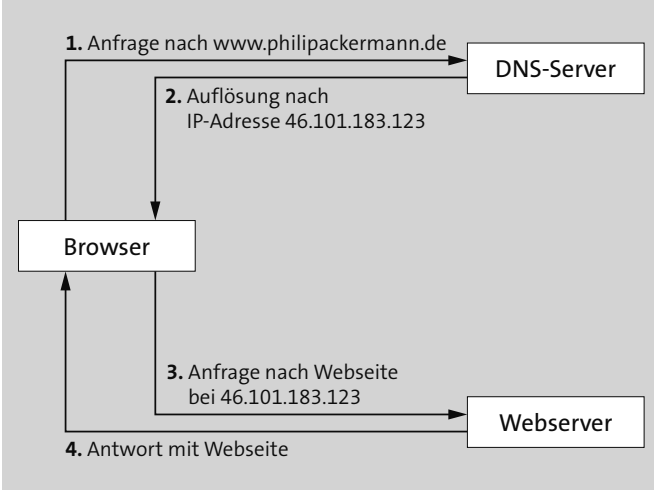


Abbildung 1.3 Das Prinzip von DNS

1.2 Aufbau von Webapplikationen

Für die Entwicklung von Webapplikationen sind insbesondere für die Entwicklung im Frontend drei Sprachen ausschlaggebend und für Webentwickler unverzichtbar. Konkret sind dies die *Auszeichnungssprache* HTML, die *Stilsprache* CSS und die *Programmiersprache* JavaScript. In diesem Abschnitt möchte ich Ihnen den Zusammenhang zwischen diesen drei Sprachen erläutern. Im Detail gehe ich dann auf jede dieser drei Sprachen in den folgenden Kapiteln ein.

1.2.1 Webseiten erstellen mit HTML (Hypertext Markup Language)

Mithilfe von *HTML*, der *Hypertext Markup Language*, definieren Sie über *HTML-Elemente* und *HTML-Attribute* die *Struktur* einer Webseite und legen die *Bedeutung* (die *Semantik*) einzelner Komponenten auf einer Webseite fest. Beispielsweise beschreiben Sie mit HTML, welcher Bereich auf der Webseite den Hauptinhalt darstellt, welcher die Navigation und welcher den Fußbereich, und definieren dabei Komponenten wie Formulare, Listen, Schaltflächen, Eingabefelder oder, wie in Abbildung 1.4 zu sehen, Tabellen.

Artist	Album	Release Date	Genre
Monster Magnet	Powertrip	1998	Spacerock
Kyuss	Welcome to Sky Valley	1994	Stonerrock
Ben Harper	The Will to Live	1997	Singer/Songwriter
Tool	Lateralus	2001	Progrock
Beastie Boys	Ill Communication	1994	Hip Hop

Abbildung 1.4 Mithilfe von HTML definieren Sie die Struktur einer Webseite.

Hinweis

Mit HTML beschäftigen wir uns im Detail in Kapitel 2, »Webseiten strukturieren mit HTML«.

1.2.2 Webseiten gestalten mit CSS (Cascading Style Sheets)

Nur HTML alleine macht eine Webseite visuell noch nicht sonderlich ansprechend. Für diesen Teil ist *CSS* (*Cascading Style Sheets*) zuständig. Hierüber gestalten Sie mithilfe von sogenannten *CSS-Regeln*, wie die einzelnen Komponenten, die Sie zuvor in HTML definiert haben, visuell dargestellt werden. Sie legen also über CSS das *Design* und *Layout* einer Webseite fest. Beispielsweise definieren Sie Textfarbe, Textgröße, Umrandungen, Hintergrundfarben, Farbverläufe etc.

In Abbildung 1.5 ist zum Beispiel zu sehen, wie CSS dazu genutzt wurde, die Schriftart und Schriftgröße der Tabellenüberschriften sowie der Tabellenzellen anzupassen, Rahmen zwischen den einzelnen Spalten und Zeilen der Tabelle hinzuzufügen und die Hintergrundfarbe der einzelnen Zeilen im Wechsel mit einer jeweils anderen Hintergrundfarbe einzufärben. Das sieht schon um einiges moderner und ansprechender aus als die Standardvisualisierung des reinen HTML aus Abbildung 1.4.

Artist	Album	Release Date	Genre
Monster Magnet	Powertrip	1998	Spacerock
Kyuss	Welcome to Sky Valley	1994	Stonerrock
Ben Harper	The Will to Live	1997	Singer/Songwriter
Tool	Lateralus	2001	Progrock
Beastie Boys	Ill Communication	1994	Hip Hop

Abbildung 1.5 Mit CSS definieren Sie das Layout und das Aussehen einzelner Elemente der Webseite.

Hinweis

In Kapitel 3, »Webseiten gestalten mit CSS«, lernen Sie, wie CSS genau funktioniert.

1.2.3 Webseiten interaktiv machen mit JavaScript

HTML definiert die Struktur und CSS das Design und Layout. Beides sind – anders als mitunter fälschlicherweise zu lesen – **keine Programmiersprachen** und ermöglichen es daher auch nicht, Anwendungslogik innerhalb einer Webseite zu implementieren. Das ist genau der Punkt, an dem *JavaScript* ins Spiel kommt. Anders als die anderen beiden Sprachen **ist JavaScript eine Programmiersprache** und dient dazu, der Webseite (bzw. den Komponenten auf einer Webseite) dynamisches Verhalten hinzuzufügen und so die *Interaktivität* auf der Webseite zu erhöhen.

Beispiele hierfür sind die Sortierung und Filterung von Tabellendaten (siehe Abbildung 1.6 und Abbildung 1.7), dynamisches (clientseitiges) Generieren von Inhalten, clientseitige Validierung von Formulareingaben (wobei dies teilweise auch mit HTML möglich ist) und vieles mehr.

Hinweis

Eine ausführlichere Einführung in JavaScript gebe ich Ihnen in Kapitel 4, »Webseiten interaktiv machen mit JavaScript«.

Eine einzelne Webseite besteht also (in den meisten Fällen) aus einer Kombination von HTML-, CSS- und JavaScript-Code (siehe Abbildung 1.8).

Search artist

Artist	Album	Release Date	Genre
Beastie Boys	Ill Communication	1994	Hip Hop
Ben Harper	The Will to Live	1997	Singer/Songwriter
Kyuss	Welcome to Sky Valley	1994	Stonerrock
Monster Magnet	Powertrip	1998	Spacerock
Tool	Lateralus	2001	Progrock

Abbildung 1.6 JavaScript ermöglicht es, eine Webseite nutzerfreundlicher und interaktiver zu gestalten, zum Beispiel, um die Daten in einer Tabelle sortierbar zu machen ...

Be

Artist	Album	Release Date	Genre
Beastie Boys	Ill Communication	1994	Hip Hop
Ben Harper	The Will to Live	1997	Singer/Songwriter

Abbildung 1.7 ... oder die Daten filterbar zu machen.

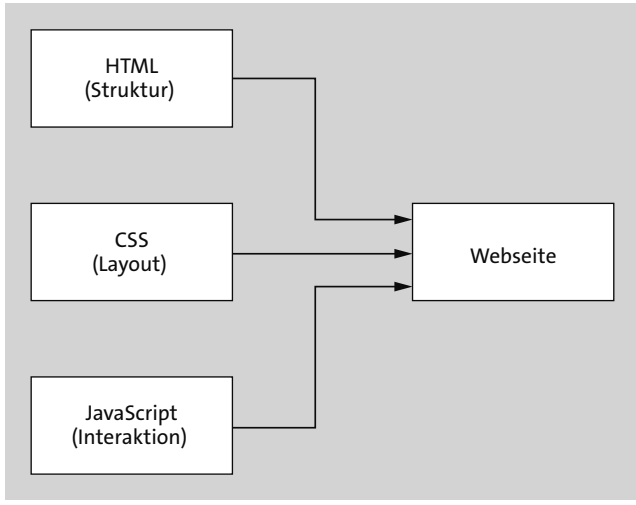


Abbildung 1.8 In der Regel wird innerhalb einer Webseite eine Kombination aus HTML, CSS und JavaScript verwendet.

Merke

HTML dient der Struktur einer Webseite, CSS dem Layout und dem Design, JavaScript dem Verhalten und der Interaktivität. HTML und CSS sind keine Programmiersprachen – HTML ist eine *Auszeichnungssprache* und CSS eine *Stilsprache*. Von den drei genannten ist nur JavaScript eine *Programmiersprache*.

1.2.4 Webseiten dynamisch machen mit serverseitiger Logik

Webseiten bestehen wie erwähnt aus einem Teil, der auf dem Client ausgeführt wird, und einem Teil, der auf dem Server ausgeführt wird. Prinzipiell muss man diesbezüglich allerdings unterscheiden: Bei *statischen Webseiten* liefert der Webserver lediglich den **statischen Inhalt** in Form von Dateien (HTML, CSS, JavaScript, Bilder etc.) aus dem Dateisystem aus (Abbildung 1.9).

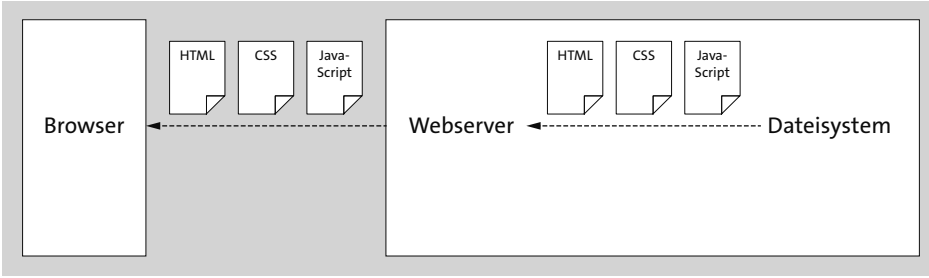


Abbildung 1.9 Das Prinzip statischer Webseiten

Bei *dynamischen Webseiten* hingegen generiert der Webserver **zur Laufzeit dynamisch** den Inhalt (in Form von HTML-Code), der an den Client zurückgeschickt wird. Die Daten, auf deren Basis der Inhalt generiert wird, holt sich der Webserver dazu in der Regel aus einer *Datenbank* (Abbildung 1.10).

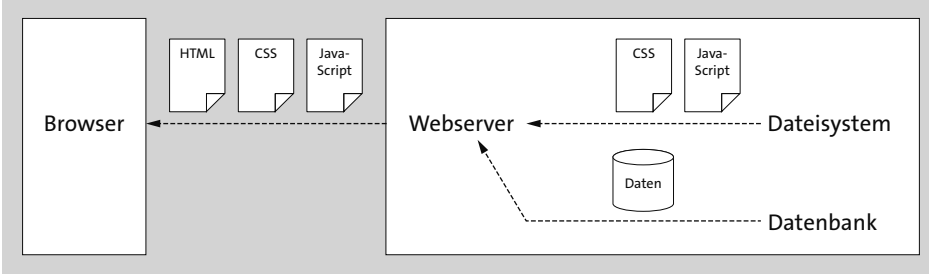


Abbildung 1.10 Das Prinzip dynamischer Webseiten

Statische Webseiten sind vom Implementierungsaufwand in der Regel nicht so aufwendig wie dynamische Webseiten, weil das Backend lediglich aus einem einfachen Webserver besteht, der die Dateien (ohne eigens zu implementierende Backend-Logik) einfach an den Client sendet. Dynamische Webseiten hingegen können je nach Anwendungsfall beliebig komplex werden.

1.3 Fullstack-Entwicklung

Nachdem Sie nun einen ersten Einblick und Überblick über die wichtigsten Begriffe in der Webentwicklung bekommen haben, möchte ich Ihnen zeigen, was es mit dem Begriff Fullstack auf sich hat, woher er kommt und was ausgehend davon die Aufgaben eines Fullstack-Entwicklers sind.

1.3.1 Was sind Software-Stacks?

Bei einem *Software-Stack* (auch *Stack*, *Solution-Stack* oder *Lösungsstack*) handelt es sich um eine konkrete Auswahl bestimmter Komponenten, die gemeinsam eine *Plattform* bilden, mit der eine Applikation entwickelt und betrieben wird. Einzelne Komponenten des Stacks können dabei folgende sein:

- Das **Betriebssystem**, auf dem die Software bzw. Applikation läuft (sowohl das Betriebssystem der Clientseite als auch das der Serverseite). Hierzu zählen die klassischen Betriebssysteme Linux, macOS und Windows, aber auch mobile Betriebssysteme wie Android oder iOS.
- Der **Webserver**, der die Applikation *hostet*, beispielsweise nginx oder der Apache HTTP Server.
- Die **Programmiersprache**, die verwendet wird, um die Applikationslogik zu implementieren, beispielsweise JavaScript, Java, C++, PHP (siehe auch Kapitel 13, »Programmiersprachen auf der Serverseite verwenden«).
- Die **Programmierwerkzeuge** (*Tools*), die während der Entwicklung zum Einsatz kommen, wie beispielsweise bestimmte *Compiler*, die (für Menschen lesbaren) *Quelltext* in (für Computer lesbaren) *Maschinencode* übersetzen.
- Die **Bibliotheken**, die zum Einsatz kommen, beispielsweise Frontend-Bibliotheken wie Lodash (*https://lodash.com/*) oder jQuery (*https://jquery.com/*).
- Die **Frameworks**, die zum Einsatz kommen, beispielsweise Web-Frameworks wie Express (*https://expressjs.com/*).
- Die **Laufzeitumgebung**, unter der die Applikation ausgeführt wird, beispielsweise die JavaScript-Laufzeitumgebung Node.js, die es ermöglicht, JavaScript-Code auch auf Serverseite auszuführen (siehe Kapitel 14, »JavaScript auf der Serverseite verwenden«).

- Die verwendeten **Datenbanken**, etwa *relationale Datenbanken* wie MySQL und PostgreSQL oder *nicht-relationale Datenbanken* (auch: *NoSQL*-Datenbanken) wie MongoDB (siehe Kapitel 14, »JavaScript auf der Serverseite verwenden«).

Der Begriff *Stack* rührt nun daher, dass die einzelnen Komponenten in der Plattform hierarchisch angeordnet und quasi wie »übereinandergestapelt« sind (Abbildung 1.11).

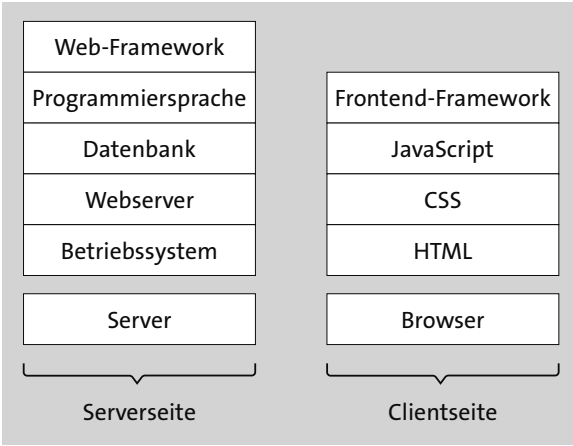


Abbildung 1.11 Software-Stacks sind eine Auswahl von bestimmten Komponenten für die Entwicklung einer Applikation.

Bei Webapplikationen unterscheidet man darüber hinaus oft auch zwischen einem *serverseitigen Stack* (*Server-Stack*) und einem *clientseitigen Stack* (*Client-Stack*), die grundsätzlich verschieden sein können. Beispielsweise ist es nicht unüblich, für die Implementierung des Backends auf eine Programmiersprache wie Java zurückzugreifen und das Frontend mit HTML, JavaScript und CSS umzusetzen. Umgekehrt ist es auch denkbar, das Backend in JavaScript zu programmieren und das Frontend in Java (beispielsweise bei der Entwicklung einer Android-basierten App für Smartphones).

1.3.2 Welche Stacks gibt es?

Eines der bekanntesten (und ältesten) Beispiele für einen Backend-Stack ist der sogenannte *LAMP-Stack*, also eine Kombination aus Linux (Betriebssystem), Apache (Webserver), MySQL (Datenbank) und PHP (Programmiersprache). Alternative Varianten dieses Stacks verwenden statt Linux eines der anderen Betriebssysteme und nennen sich dann *WAMP* (Windows-Variante) bzw. *MAMP* (macOS-Variante). Neuere Varianten dieses Stacks verwenden auch Ruby oder Python als Programmiersprache anstelle von PHP.

Nicht ganz so alt, aber dafür in den letzten Jahren umso mehr bekannt geworden ist der *MEAN-Stack* (der allerdings nicht ganz sauber zwischen Backend und Frontend unterscheidet). MEAN steht für MongoDB (eine nicht-relationale Dokumentendatenbank), express (ein Web-Framework und ein Webserver), Angular (ein Frontend-Framework) sowie die serverseitige JavaScript-Laufzeitumgebung Node.js. Diese Kombination hat für Entwickler insofern den charmanten Vorteil, dass sowohl für die Clientseite als auch für die Serverseite dieselbe Programmiersprache zum Einsatz kommt.

Der ebenfalls gern verwendete *MERN-Stack* ist eine leichte Abwandlung des MEAN-Stacks, bei dem lediglich für das Frontend statt Angular das Framework React zum Einsatz kommt. Die anderen Komponenten des Stacks, also MongoDB, Express und Node.js, sind dabei die gleichen.

Hinweis

Software-Stacks müssen nicht zwangsweise alle der weiter oben genannten Komponenten enthalten. Die beiden Stacks MEAN und MERN sind beispielsweise nicht auf ein einziges Betriebssystem beschränkt, sondern können dank der Plattformunabhängigkeit von Node.js auf allen drei großen Betriebssystemen (Linux, macOS, Windows) betrieben werden.

HTML, CSS und JavaScript

Der Frontend-Stack – bestehend aus den grundlegenden Webtechnologien HTML, CSS und JavaScript – ist für die Entwicklung von Frontends so wichtig, dass hier schon gar nicht mehr explizit von einem Stack gesprochen wird.

1.3.3 Was ist ein Fullstack-Entwickler?

Nachdem wir nun den Begriff *Software-Stack* geklärt haben, stellt sich die Frage, was denn genau *Fullstack* bedeutet, was ein *Fullstack-Entwickler* ist und welches seine Aufgaben sind.

Fullstack

Der Begriff *Fullstack* umfasst den kompletten Stack, also sowohl den Backend-Stack als auch den Frontend-Stack. Ein Fullstack-Entwickler ist also jemand, der sowohl das Backend entwickelt als auch das Frontend, bzw. jemand, der sich gleichermaßen mit Backend-Technologien wie mit Frontend-Technologien auskennt.

Fullstack-Entwickler

Ein Fullstack-Entwickler ist also ein echter Allrounder: Er kann sowohl eine Weboberfläche mit HTML, CSS und JavaScript als auch einen Webservice implementieren, der auf Serverseite läuft und Anfragen vom Client verarbeitet. Er ist außerdem in der Lage, je nach Anforderungen die richtige Datenbank auszuwählen, und weiß auch, wie die gesamte Applikation für den Produktiveinsatz vorbereitet werden kann (Abbildung 1.12).

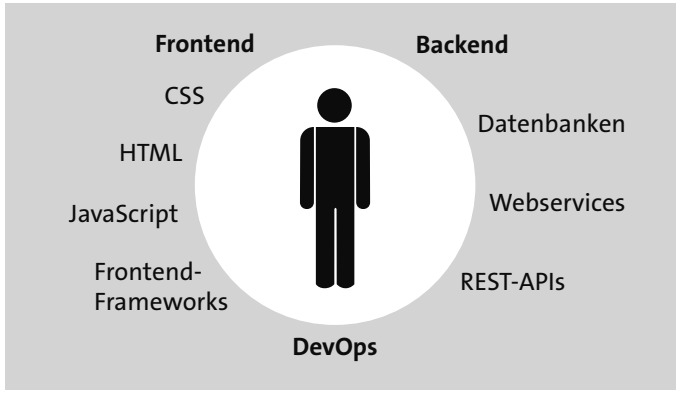


Abbildung 1.12 Fullstack-Entwickler müssen mit einer ganzen Menge Anforderungen zurechtkommen.

Anforderungen an Fullstack-Entwickler

Schauen wir uns kurz an, welche Rollen es überhaupt in einem Webprojekt (ausschließlich bezüglich der Entwicklung) geben kann:

- **Frontend-Entwickler**: Kann mit HTML, CSS und JavaScript komplexe Weboberflächen bauen und hat im besten Fall auch Erfahrung im Design. Kennt sich außerdem mit Frontend-Bibliotheken und -Frameworks wie Angular, React und Vue aus, kennt die entsprechenden Tools wie WebPack, Babel und weiß mit CSS-Präprozessorsprachen umzugehen.
- **Backend-Entwickler**: Kann komplexe Applikationslogik für die Serverseite entwickeln und diese beispielsweise über Webservices zur Verfügung stellen. Kennt sich mit dem Design von Webservice-APIs aus und weiß, welche Datenbank für welchen Zweck geeignet ist.
- **DevOps-Spezialisten**: Den Begriff *DevOps* muss man zunächst genauer betrachten, um ihn besser zu verstehen. Er setzt sich wiederum aus den Begriffen *Development* (also »Entwicklung«) und *Operations* (also »Vorgänge«) zusammen und vereint diese beiden Tätigkeitsfelder in einer Rolle, den sogenannten *DevOps-Spe-*

zialisten (bzw. kurz *DevOps*). Zu den Aufgaben von DevOps-Spezialisten zählen also beispielsweise, Anwendungen für das Produktivsystem zusammenzustellen (»deployen«), Build-Systeme zu konfigurieren und Webserver zu administrieren. Darüber hinaus kennen sich DevOps-Spezialisten auch oft mit Themen wie Sicherheit und Performance von Webanwendungen aus.

Als Fullstack-Entwickler erfüllt man im besten Fall alle der genannten Anforderungen und vereint die drei Rollen bis zu einem gewissen Grad in einer Person. Keine leichte Aufgabe und zudem eine, für die man einige Jahre lang Erfahrung sammeln muss.

Hinweis

Es sei darauf hingewiesen, dass die oben genannten Anforderungen an Frontend-Entwickler, Backend-Entwickler und DevOps keineswegs vollständig sind und dass es auch bei diesen drei Gruppen weitere Unterteilungen in speziellere Untergruppen gibt, beispielsweise Datenbankspezialisten, Frontend-Designer, UX-Designer, Tester u. v. m.

Fullstack-Entwickler vs. Spezialisten

Aufgrund der breit gestreuten Anforderungen läuft man als Fullstack-Entwickler allerdings auch schnell Gefahr, »alles nur ein bisschen, dafür aber nichts ganz richtig zu können«. Dem stehen dann die Spezialisten gegenüber, die sich genau auf ein bestimmtes Gebiet spezialisiert haben. Ein Datenbankspezialist kennt sich mit Datenbanken in der Regel besser aus als ein Fullstack-Entwickler, ein Frontend-Designer kennt im Zweifelsfall besser die verschiedenen CSS-Kniffe, die notwendig sind, um ein Layout genau nach Vorlage umsetzen zu können. Und ein DevOp-Spezialist ist bei der Arbeit auf der Kommandozeile und der Verwaltung von Servern normalerweise auch effektiver als ein Fullstack-Entwickler.

Große Unternehmen vs. Start-ups

In großen Unternehmen bzw. großen Softwareprojekten ist es zudem häufig so, dass es für verschiedene Bereiche des verwendeten Stacks ganze Teams gibt. Ein Team kümmert sich um die Datenbank, ein weiteres um das Frontend-Design, ein weiteres ist für die Umsetzung des Designs in eine konkrete Weboberfläche zuständig, und so weiter. Fullstack-Entwickler werden daher oftmals insbesondere in kleineren Firmen oder Start-ups benötigt, die oft agiler und flexibler auf Anforderungen reagieren können (oder müssen).

Fokus in diesem Buch

Fullstack-Entwickler wird man nicht von heute auf morgen, und auch das Durcharbeiten eines dicken Buches wie des vorliegenden wird aus Ihnen noch keinen Fullstack-Entwickler machen. Ziel dieses Buches ist es vielmehr, Ihnen eine Übersicht über das breite Spektrum von Themengebieten zu geben, die für Fullstack-Entwickler wichtig sind. Das Buch soll Sie auf Ihrem Weg zu einem Entwickler begleiten, der eine Übersicht über viele Technologien hat, sowohl im Backend als auch im Frontend und zu einem gewissen Grad auch im Bereich DevOps. Ob Sie sich dann Fullstack-Entwickler nennen möchten oder nicht, das sei Ihnen überlassen.

1.3.4 Aufbau des Buches

Die insgesamt 23 Kapitel lassen sich grob in drei Abschnitte unterteilen: Die Kapitel 1 bis 11 beschäftigen sich vor allem mit allgemeinen Grundlagen und Themen aus dem Frontend-Bereich, in den Kapiteln 12 bis 17 geht es um Themen, die vor allem dem Backend-Bereich zuzuordnen sind, und die Kapitel 18 bis 23 befassen sich mit sogenannten Querschnittsthemen wie dem Testen und der Sicherheit (also Themen, die sowohl im Frontend als auch im Backend relevant sind), mit dem Deployment sowie der Verwaltung von Webprojekten.

Frontend

Dieser Teil des Buches ist wie folgt aufgebaut:

- In Kapitel 1, »Die Grundlagen verstehen«, lernen Sie die wichtigsten Grundlagen kennen (bzw. haben sie schon kennengelernt).
- In Kapitel 2, »Webseiten strukturieren mit HTML«, Kapitel 3, »Webseiten gestalten mit CSS«, und Kapitel 4, »Webseiten interaktiv machen mit JavaScript«, werde ich auf die drei wichtigsten Sprachen des Webs im Detail eingehen.
- In Kapitel 5, »Webprotokolle verwenden«, lernen Sie, welche Protokolle im Rahmen von Webapplikationen wichtig sind und wann diese jeweils zum Einsatz kommen.
- In Kapitel 6, »Webformate verwenden«, lernen Sie verschiedene Formate kennen, die für die Implementierung von Webapplikationen wichtig sind.
- In Kapitel 7, »Web-APIs verwenden«, stelle ich Ihnen eine Auswahl von Web-APIs für JavaScript vor, das heißt programmatische Schnittstellen, die über JavaScript gesteuert werden können.
- In Kapitel 8, »Webseiten für Barrierefreiheit optimieren«, zeige ich Ihnen, was Sie im Hinblick auf die Barrierefreiheit von Webseiten beachten sollten.

- In Kapitel 9, »CSS vereinfachen mit CSS-Präprozessoren«, lernen Sie Tools kennen, die Ihnen dabei helfen, CSS-Code übersichtlicher zu gestalten.
- In Kapitel 10, »Single-Page-Applikationen implementieren«, zeigt Ihnen der Gastautor dieses Buches, Sebastian Springer, was Single-Page-Applikationen (kurz: SPAs) sind und wie Sie mithilfe der JavaScript-Bibliothek React eine solche implementieren.
- In Kapitel 11, »Mobile Anwendungen implementieren«, lernen Sie die verschiedenen Arten mobiler Webanwendungen kennen.

Backend

Dieser Teil des Buches gliedert sich wie folgt:

- In Kapitel 12, »Webarchitekturen verstehen und einsetzen«, stelle ich Ihnen verschiedene Webarchitekturen vor, die die Grundlage für die meisten Webapplikationen bilden.
- In Kapitel 13, »Programmiersprachen auf der Serverseite verwenden«, wenden wir uns den verschiedenen Programmiersprachen zu, die auf Serverseite zum Einsatz kommen. Ich zeige Ihnen, welche Arten von Programmiersprachen es gibt und welche für die Implementierung von Webapplikationen relevant sind.
- Aufbauend auf Kapitel 13 zeige ich Ihnen in Kapitel 14, »JavaScript auf der Serverseite verwenden«, wie Sie mithilfe der Laufzeitumgebung Node.js JavaScript auf Serverseite ausführen können.
- Für Kapitel 15, »Die Sprache PHP verwenden«, konnte ich einen weiteren Gastautor für dieses Buch gewinnen: Florian Simeth gibt Ihnen hier eine Einführung in die Sprache PHP.
- In Kapitel 16, »Webservices implementieren«, lernen Sie, was Webservices sind und wie Sie diese implementieren.
- Kapitel 17, »Daten in Datenbanken speichern«, gibt Ihnen alle wichtigen Informationen, die Sie als Fullstack-Entwickler rund um Datenbanken wissen müssen.

Querschnittsthemen und DevOps

Dieser Teil des Buches widmet sich folgenden Themen:

- In Kapitel 18, »Webanwendungen testen«, lernen Sie, wie sich Webapplikationen automatisiert testen lassen.
- In Kapitel 19, »Webanwendungen deployen und hosten«, zeige ich Ihnen, wie Sie Webapplikationen für den Produktiveinsatz vorbereiten, sprich, wie Sie sie »deployen«.
- In Kapitel 20, »Webanwendungen absichern«, zeige ich Ihnen, was Sie bezüglich der Sicherheit von Webapplikationen beachten müssen und wie Sie Schwachstellen und Sicherheitslücken vorbeugen.

- In Kapitel 21, »Die Performance von Webanwendungen optimieren«, beschäftigen wir uns damit, wie sich Webanwendungen bezüglich ihrer Geschwindigkeit und Performance optimieren lassen.
- In Kapitel 22, »Webprojekte organisieren und verwalten«, lernen Sie, wie Sie Webprojekte am besten verwalten und ihre Versionen organisieren.
- In Kapitel 23, »Webprojekte managen«, zeige ich Ihnen, welche Arten des Projektmanagements es gibt, und gehe etwas detaillierter auf das Vorgehensmodell Scrum ein.

Anhang

Abgerundet wird das Buch durch einen Anhang, in dem Sie Übersichten zu HTTP und HTML finden (Anhang A und B) sowie Informationen und Befehlsreferenzen zu den in diesem Buch verwendeten Tools (Anhang C).

Übersicht

Damit Sie immer einen Überblick haben, wo das jeweils aktuelle Kapitel thematisch einzuordnen ist, finden Sie am Beginn jedes Kapitels jeweils folgende Abbildung, wobei das jeweils besprochene Thema hervorgehoben ist. Einige der Themen sind dabei auf Clientseite einzuordnen (beispielsweise HTML oder Web-APIs), andere auf Serverseite (beispielsweise Webservices und Datenbanken); ein Thema befindet sich dazwischen (Webprotokolle) und andere – die Querschnittsthemen – auf beiden Seiten.

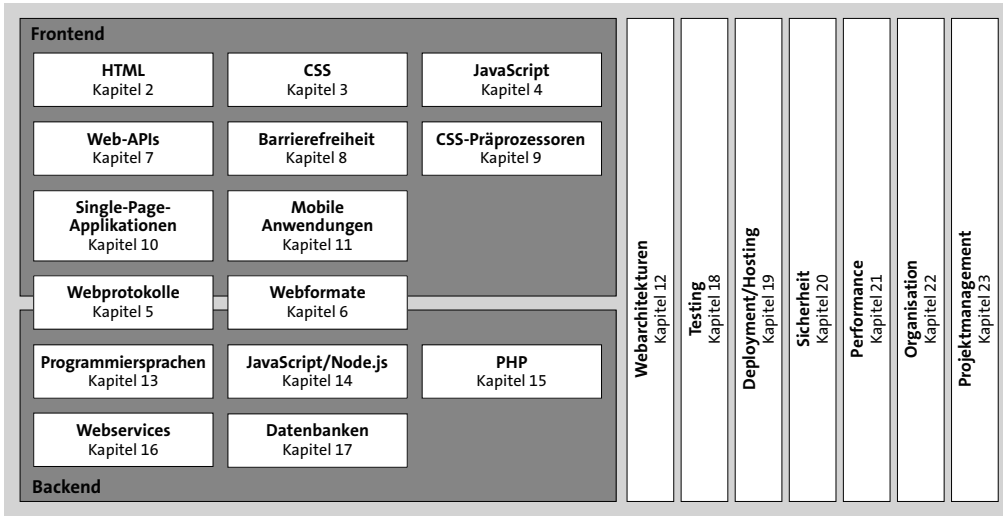


Abbildung 1.13 Einordnung der verschiedenen Themen

1.4 Tools für Fullstack-Entwickler

Für die Entwicklung von Webapplikationen benötigen Sie das richtige Werkzeug: Tools, die Ihnen die Arbeit erleichtern und ohne die die Entwicklung um einiges aufwendiger wäre. Im Laufe dieses Buches werde ich Ihnen diesbezüglich immer wieder Tipps geben. An dieser Stelle möchte ich Ihnen aber schon einmal die grundlegenden Tools vorstellen, die Sie für die Entwicklung unbedingt benötigen.

- **Editoren**: Auch wenn für die Entwicklung von Webapplikationen bzw. die Implementierung des dazugehörigen Quelltextes prinzipiell ein einfacher Texteditor reicht, empfehle ich Ihnen, sich einen guten Editor zu installieren, der Sie beim Schreiben von Quelltext unterstützt und der speziell für die Entwicklung ausgelegt ist. Code-Editoren unterstützen Sie beispielsweise dahingehend, dass sie den Quelltext farblich hervorheben (*Syntax-Highlighting*), Ihnen Schreibarbeit bei wiederkehrenden Quelltextbausteinen abnehmen, *Syntaxfehler* im Quelltext erkennen und vieles mehr.
- **Entwicklungsumgebungen**: Hierbei handelt es sich vereinfacht gesagt um sehr mächtige Code-Editoren, die neben Features wie Syntax-Highlighting auch fortgeschrittene Features wie integrierte Debugging-Tools, Versionskontrolle etc. zur Verfügung stellen.
- **Browser**: Natürlich darf für das Testen von Webapplikationen ein Browser nicht fehlen. Insbesondere die Entwicklertools (*Developer Tools*), die über die meisten Browser zur Verfügung stehen, sind aus dem Werkzeugkasten eines Webentwicklers nicht mehr wegzudenken.

1.4.1 Editoren

Mittlerweile stehen eine ganze Reihe sehr guter Code-Editoren für die Entwicklung von Webapplikationen zur Verfügung. Besonders beliebt sind beispielsweise Sublime Text (*www.sublimetext.com*, Abbildung 1.14), Atom (*https://atom.io*, Abbildung 1.15) sowie Coda 2 (*https://panic.com/coda*) und Brackets (*https://brackets.io*). Mit Ausnahme von Coda 2 stehen alle der genannten Editoren für Linux, Windows und macOS zur Verfügung (Coda 2 nur für Letzteres). Prinzipiell ähneln sich die genannten Editoren sehr stark, sodass es hier vor allem auf den persönlichen Geschmack ankommt. Probieren Sie am besten aus, welcher Ihnen am meisten zusagt.

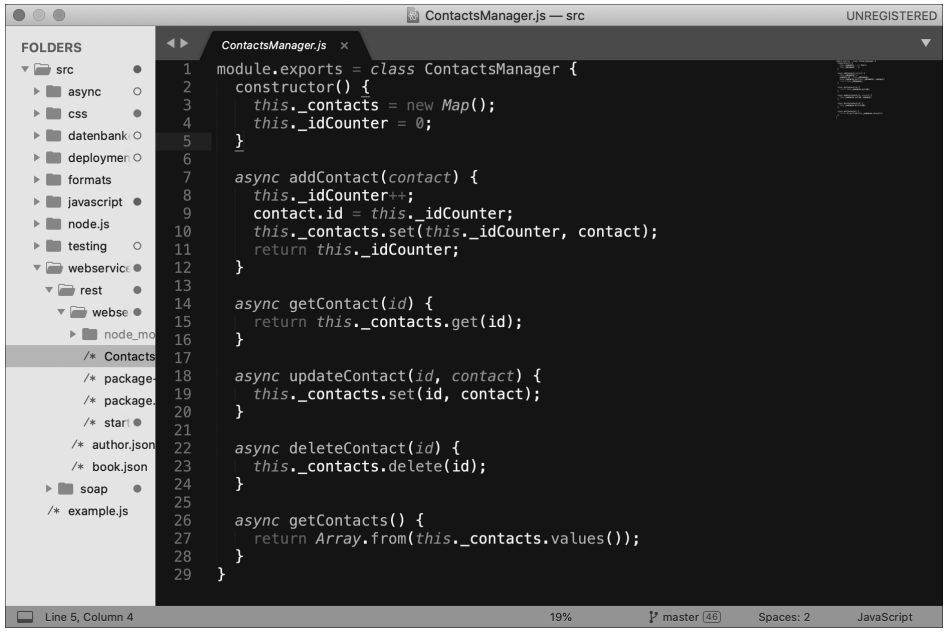


Abbildung 1.14 Der Editor Sublime Text

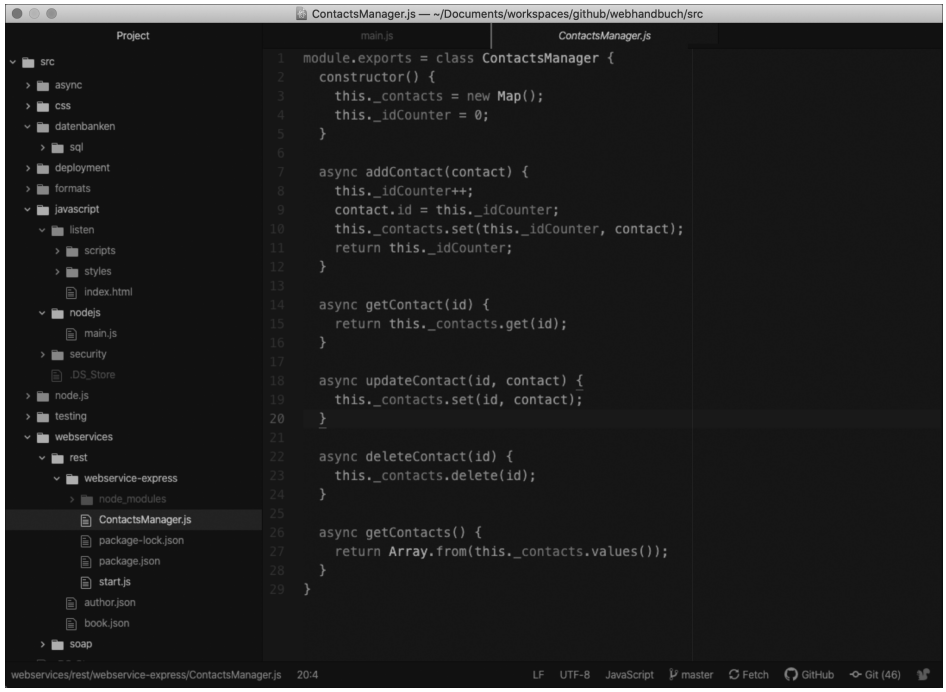


Abbildung 1.15 Der Editor Atom

1.4.2 Entwicklungsumgebungen

Mächtiger als Editoren sind *Entwicklungsumgebungen* (im Englischen kurz: *IDE* für *Integrated Development Environment*). Gegenüber einem »normalen« Editor verfügen Entwicklungsumgebungen über spezielle Features für die Entwicklung von Software, wie beispielsweise die Synchronisation mit einem *Versionskontrollsystem*, das Ausführen von *automatischen Builds* oder die Integration von *Test-Frameworks*. Und ist ein Feature standardmäßig mal nicht installiert, lässt es sich meistens nachträglich über ein entsprechendes Plugin nachinstallieren.

Bekannte Beispiele für Entwicklungsumgebungen speziell für die Entwicklung von Webanwendungen sind Microsoft Visual Studio Code (*https://code.visualstudio.com*, Abbildung 1.16) und WebStorm von IntelliJ (*www.jetbrains.com/webstorm*, Abbildung 1.17).

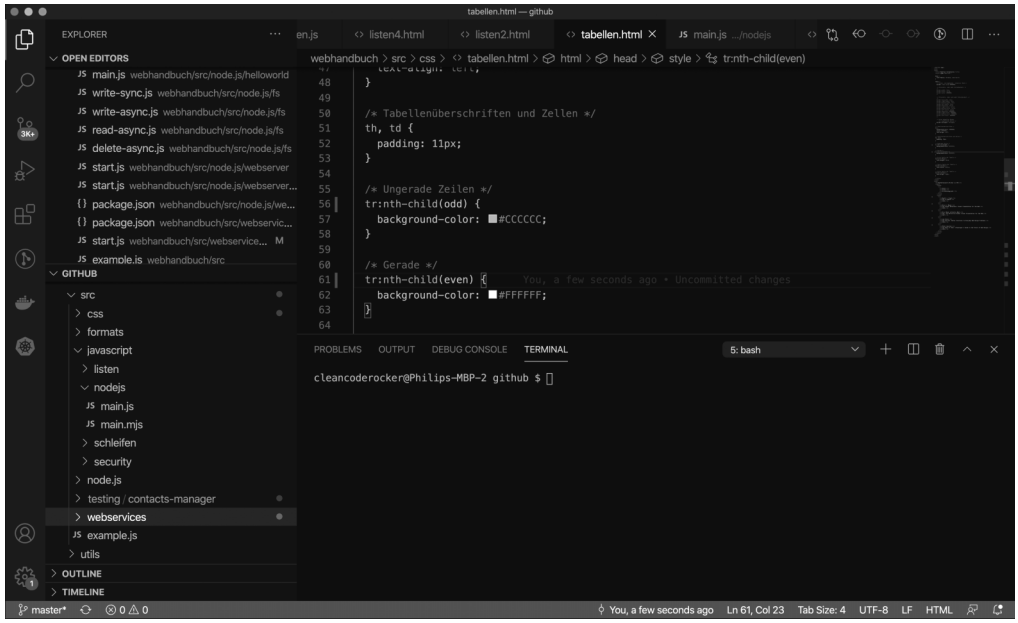


Abbildung 1.16 Die Entwicklungsumgebung Visual Studio Code

> **Hinweis**
>
> Eine gute Übersicht darüber, welche Plugins für Visual Studio Code zur Verfügung stehen, bietet die offizielle Website unter *https://marketplace.visualstudio.com/vscode*.

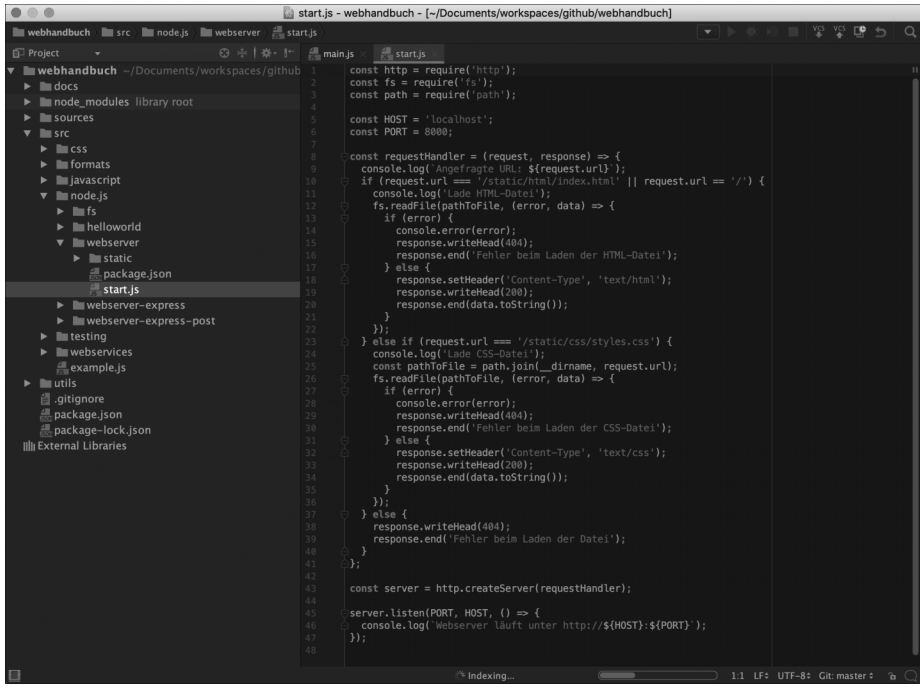


Abbildung 1.17 Die Entwicklungsumgebung WebStorm

1.4.3 Browser

Um eine Webseite bzw. ein HTML-Dokument, das Sie mithilfe eines Editors oder einer Entwicklungsumgebung erstellt haben, zu visualisieren, benötigen Sie einen *Webbrowser*. Wesentlicher Bestandteil eines Webbrowsers ist die sogenannte *Rendering Engine*, mit deren Hilfe die Visualisierung von HTML, CSS und JavaScript möglich ist.

> **Tipp**
>
> Da sich die Rendering Engines der einzelnen Browser im Detail unterscheiden (also teils zu unterschiedlichen Visualisierungen kommen), sollten Sie eine Webseite immer in mehreren Browsern testen.

Die fünf bekanntesten Browser sind:

- Google Chrome (*https://www.google.com/intl/de_de/chrome*, Abbildung 1.18)
- Mozilla Firefox (*https://www.mozilla.org/de/firefox*, Abbildung 1.19)
- Safari (*https://support.apple.com/de_DE/downloads/safari*, Abbildung 1.20)
- Opera (*https://www.opera.com/de*, Abbildung 1.21)
- Microsoft Edge (*https://www.microsoft.com/de-de/edge*, Abbildung 1.22)

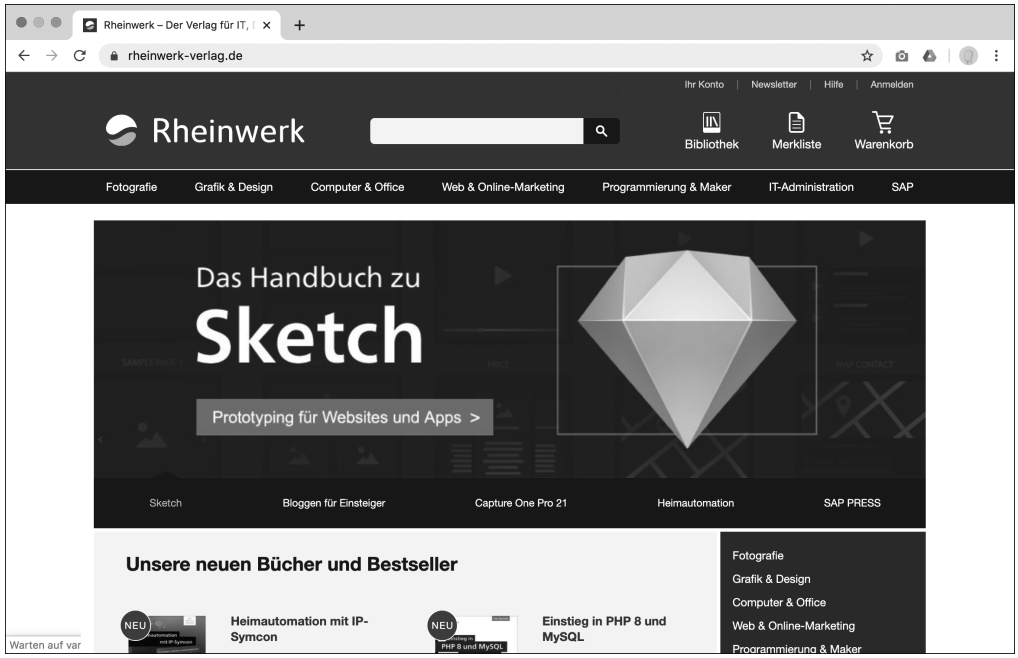


Abbildung 1.18 Google Chrome (macOS)

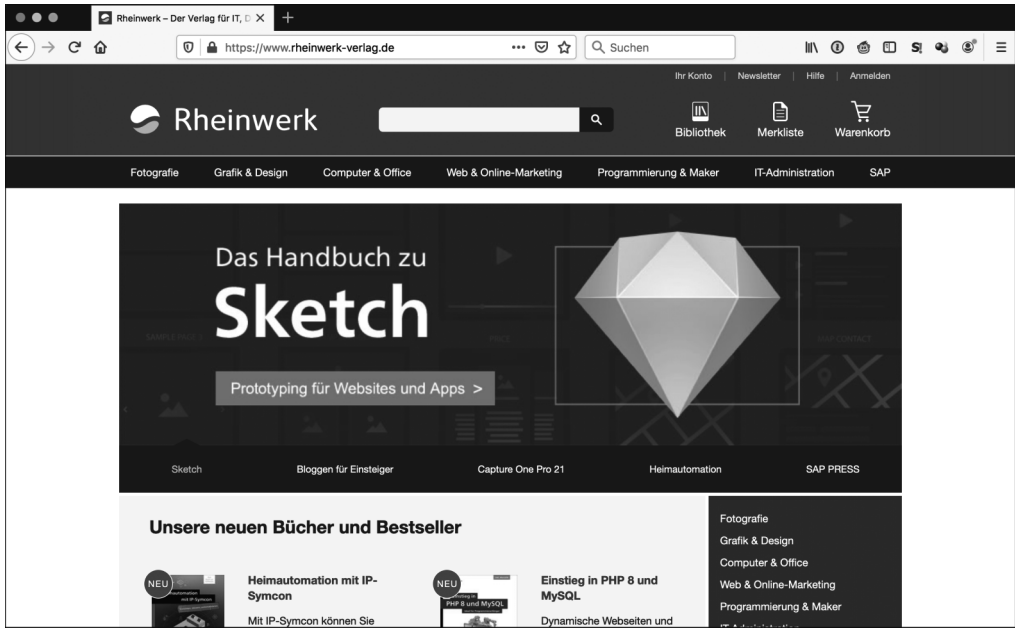


Abbildung 1.19 Mozilla Firefox (macOS)

Abbildung 1.20 Safari (macOS)

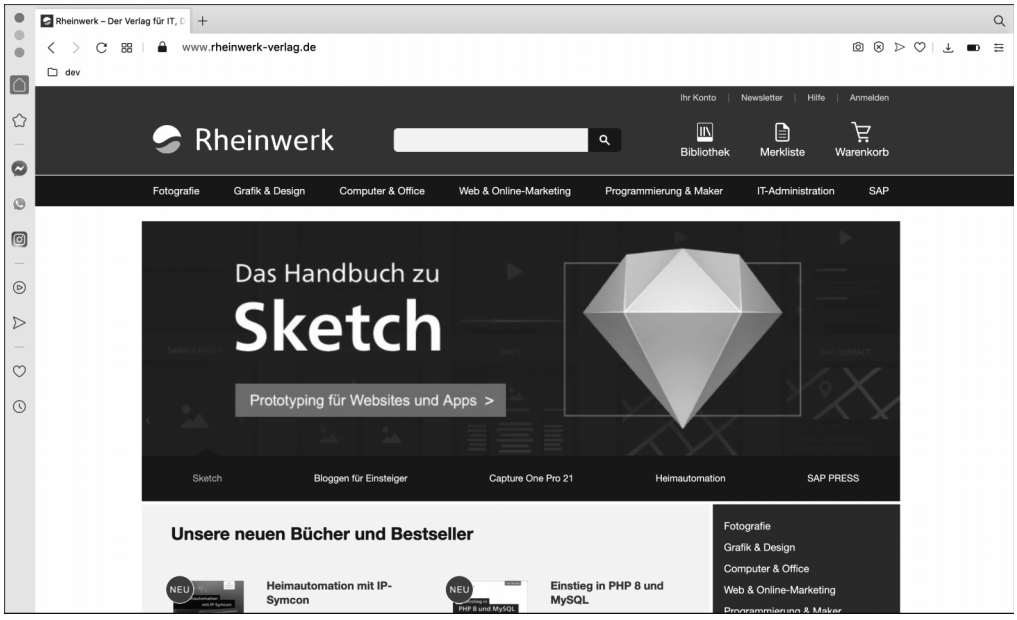


Abbildung 1.21 Opera (macOS)

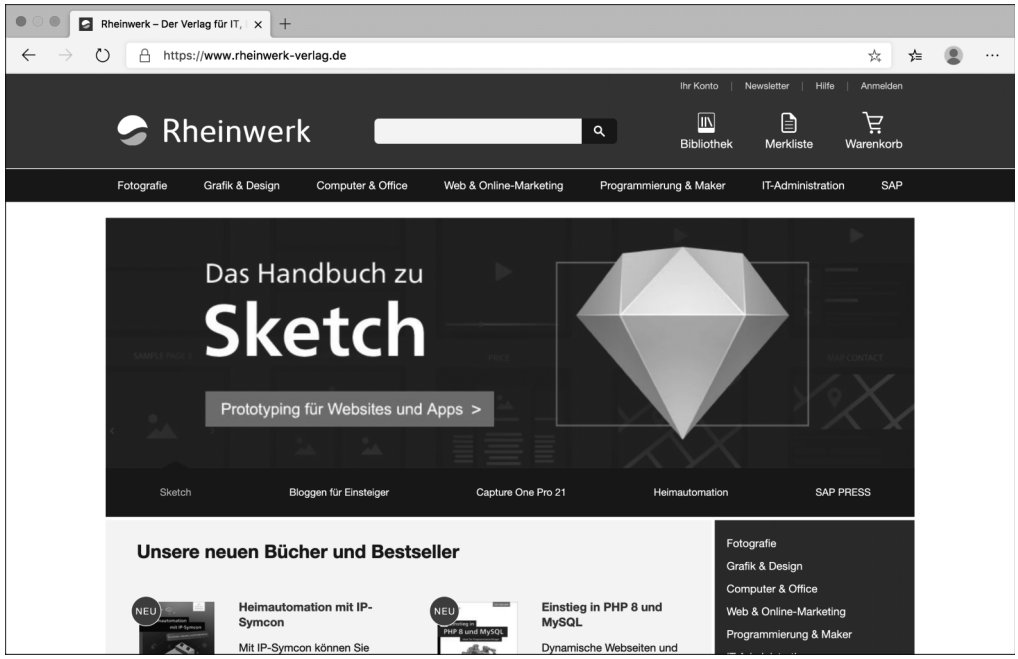


Abbildung 1.22 Microsoft Edge (macOS)

Neben diesen »großen« fünf Browsern gibt es noch eine ganze Reihe weiterer Browser. Nennenswert sind hier vor allem der (in die Jahre gekommene) Internet Explorer von Microsoft (quasi der Vorgänger des schnelleren Browsers Microsoft Edge, der – zum Leidwesen vieler Entwickler – in einigen Firmen bis heute verwendet wird) und Brave (*https://brave.com/de*), der von Brendan Eich, dem Erfinder der Sprache JavaScript, initiiert wurde. Bei Interesse finden Sie eine umfangreiche Liste von weiteren Alternativen unter den weiter unten genannten Links zu den Wikipedia-Seiten (siehe Kasten).

Hinweis

Eine Übersicht über die Verbreitung der Browser finden Sie in der Wikipedia unter *https://en.wikipedia.org/wiki/Usage_share_of_web_browsers*. Stand heute (Februar 2023, mit StatCounter-Statistiken von Dezember 2022) ist Chrome der mit Abstand (71,2 %) am meisten verwendete Browser, gefolgt von Safari (15,1 %), Microsoft Edge (3,7 %), Firefox (3,0 %) und Opera (1,3 %). Einen gelungenen Vergleich der verschiedenen Browser hinsichtlich verschiedener Kriterien wie etwa Funktionsumfang, Unterstützung von Webtechnologien etc. finden Sie zudem unter *https://en.wikipedia.org/wiki/Comparison_of_web_browsers*.

Weiterführende Informationen

Eine Übersicht darüber, welcher Browser welche Rendering Engine verwendet, finden Sie auf der englischsprachigen Wikipedia-Seite unter *https://en.wikipedia.org/wiki/Comparison_of_browser_engines*. Sehr lesenswert ist in diesem Zusammenhang auch der Artikel »Funktionsweise von Browsern: Hinter den Kulissen moderner Web-Browser« unter *https://www.html5rocks.com/de/tutorials/internals/howbrowserswork*, in dem erläutert wird, wie Browser und Rendering Engines im Detail funktionieren.

Welcher Browser ist der richtige?

Welcher Browser »der richtige« ist bzw. welchen Sie verwenden sollten, hängt von verschiedenen Faktoren ab:

- **Kundenanforderungen**: Gibt es spezielle Anforderungen vom Kunden? Oft ist es so, dass Kunden sehr spezielle Vorgaben haben, für welche Browser eine zu entwickelnde Webapplikation funktionieren muss. Hin und wieder kann dies – nicht erschrecken – auch noch der Internet Explorer sein.
- **Features**: Müssen bestimmte Features unterstützt werden? Die einzelnen Browser (bzw. genauer: die verschiedenen Rendering Engines) unterstützen je nach Version unterschiedliche Features. Insbesondere hinsichtlich der Fülle an Web-APIs, die es mittlerweile gibt, unterscheiden sich die einzelnen Engines. Eine gute Übersicht darüber, ob ein Feature unterstützt wird oder nicht, gibt die Website *https://caniuse.com*.
- **Funktionsumfang**: Mittlerweile bieten alle großen Browser einen sehr ähnlichen Funktionsumfang. Für Entwickler interessant sind hier insbesondere die genannten Entwicklertools, über die sich beispielsweise für eine bereits gerenderte Webseite HTML und CSS anpassen oder JavaScript ausführen lässt.

1.5 Zusammenfassung und Ausblick

In diesem Kapitel haben Sie die wichtigsten Grundlagen kennengelernt, die für den weiteren Verlauf des Buches wichtig sind. Zum Abschluss des Kapitels – und im Übrigen analog auch in den folgenden Kapiteln – gebe ich Ihnen eine Übersicht über die wichtigsten Punkte und einen kurzen Ausblick darauf, was Sie in den folgenden Kapiteln erwartet.

1.5.1 Die wichtigsten Punkte

Die wichtigsten Punkte aus diesem Kapitel sind:

- *Webseiten*, *Websites* und *Webanwendungen* bestehen aus einem Teil, der auf der *Clientseite* ausgeführt wird (dem *Frontend*), und einem Teil, der auf der *Serverseite* ausgeführt wird (dem *Backend*).
- Die Adresse, die Sie in die Adressleiste des Browsers eingeben, wird als *URL* (*Uniform Resource Locator*) bezeichnet.
- Jeder Webserver hat eine eindeutige Adresse, über die er erreichbar ist: die sogenannte *IP-Adresse*, wobei zwischen *IPv4* und *IPv6* unterschieden wird.
- *DNS Server* (*Domain Name System*) mappen Hostnamen auf IP-Adressen und geben für einen Hostnamen die IP-Adresse des entsprechenden Webservers zurück.
- Für die Webentwicklung sind vor allem folgende Sprachen wichtig:
  - Mithilfe der *Auszeichnungssprache HTML*, der *Hypertext Markup Language*, definieren Sie über *HTML-Elemente* und *HTML-Attribute* die *Struktur* einer Webseite und legen die *Bedeutung* (die *Semantik*) einzelner Komponenten auf einer Webseite fest.
  - Über die *Stilsprache CSS* (*Cascading Style Sheets*) gestalten Sie mithilfe von sogenannten *CSS-Regeln*, wie die einzelnen Komponenten, die Sie zuvor in HTML definiert haben, visuell dargestellt werden.
  - Über die *Programmiersprache JavaScript* fügen Sie einer Webseite *dynamisches Verhalten* hinzu.
- Bei einem *Software-Stack* bzw. kurz *Stack* handelt es sich um eine konkrete Auswahl bestimmter Komponenten, die gemeinsam eine *Plattform* bilden, mit der eine Applikation entwickelt und betrieben wird.
- Für Webapplikationen gibt es verschiedene bekannte Stacks wie *LAMP*, *WAMP*, *MAMP*, *MEAN* und *MERN*.
- *Fullstack-Entwickler* kennen sich mit allen Bereichen eines Stacks aus und vereinen die Rollen des *Frontend-Entwicklers*, des *Backend-Entwicklers* und des *DevOps*.
- Der *Werkzeugkasten* eines Fullstack-Entwicklers besteht aus einem *Editor* oder einer *Entwicklungsumgebung*, einem oder mehreren *Browsern* (in verschiedenen Versionen), den *Browser-Entwicklertools* und weiteren Tools, die ich Ihnen im weiteren Verlauf des Buches vorstellen werde.

1.5.2 Ausblick

Wie oben erwähnt besteht das Buch grob aus drei Teilen, wobei ich in den nächsten Kapiteln auf die wichtigsten Sprachen des Webs eingehen möchte: Im folgenden Kapitel beginnen wir mit HTML.

Kapitel 10
Single-Page-Applikationen implementieren

In diesem Kapitel erklärt Sebastian Springer Ihnen, wie Sie mithilfe von React eine Single-Page-Applikation umsetzen können und wie dort die Grundlagen von HTML, CSS und JavaScript zum Einsatz kommen.

Als Fullstack-Entwickler kommen Sie früher oder später mit einem der großen Frontend-Frameworks in JavaScript in Berührung. In diesem Kapitel erfahren Sie, wie Sie mithilfe von React eine Single-Page-Applikation erstellen und die Informationen Ihrer Applikation verwalten können.

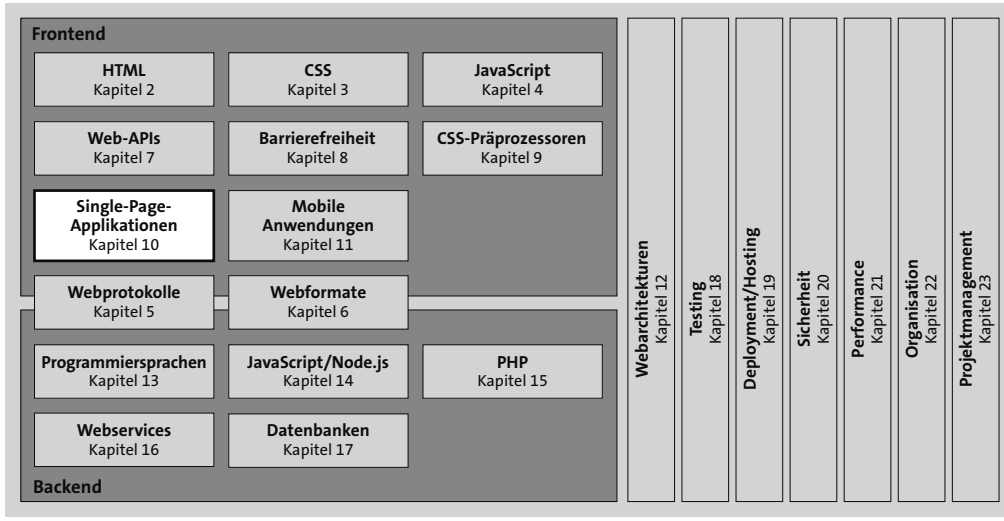


Abbildung 10.1 Single-Page-Applikationen sind Anwendungen, die aus einer einzelnen HTML-Seite bestehen, deren Inhalte dynamisch per JavaScript geändert werden.

10.1 Einführung

Mit der Kombination aus HTML, CSS und JavaScript können Sie Webapplikationen erstellen, die nativen Desktop- oder Mobile-Applikationen in nichts nachstehen. Eines der wichtigsten Merkmale ist an dieser Stelle die Tatsache, dass solche Applika-

tionen nicht mit einzelnen HTML-Seiten umgesetzt werden, zwischen denen ein Benutzer über Hyperlinks hin und her navigiert, sondern als sogenannte *Single-Page-Applikationen* oder kurz *SPA* implementiert werden.

Single-Page-Applikationen

Bei einer Single-Page-Applikation handelt es sich um eine Anwendung, die im Browser ausgeführt wird und die aus HTML, CSS und JavaScript besteht. Grundsätzlich besteht eine SPA, wie der Name andeutet, aus nur einer HTML-Seite. Diese wird mithilfe der zugrunde liegenden Webtechnologien gesteuert, und dem Benutzer werden nur bestimmte Sichten präsentiert.

Die Datenhaltung erfolgt im Speicher des Browsers. Sobald die Seite manuell vom Benutzer neu geladen wird, wird dieser Speicher zurückgesetzt und muss von der Applikation neu aufgebaut werden, was einen entscheidenden Geschwindigkeitsnachteil mit sich bringt. So ist es das Ziel eines jeden Entwicklers, diese vollständigen Page Loads zu verhindern.

React unterstützt Sie bei der Implementierung einer solchen SPA und stellt Ihnen eine Reihe von Hilfsmitteln zur Verfügung. Was React selbst nicht leistet, können Sie über externe Bibliotheken einbinden.

Eine solche Applikation können Sie in reinem HTML, CSS und JavaScript umsetzen. Das hat jedoch den Nachteil, dass Sie eine ganze Reihe von Standardproblemen lösen und so quasi das Rad noch mal neu erfinden müssen. Besser ist an dieser Stelle, wenn Sie auf eine etablierte Lösung setzen, die Ihnen diese Aufgaben abnimmt. Wie in der JavaScript-Welt üblich, gibt es auch hier nicht die eine richtige Standardlösung, sondern zahlreiche verschiedene Frameworks und Bibliotheken, die Sie einsetzen können. Über die letzten Jahre haben sich jedoch drei große Lösungen herauskristallisiert und sich als Quasistandard etabliert: Angular, Vue und React.

Allen drei gemeinsam ist, dass es sich um Open-Source-Projekte handelt, die auf GitHub verwaltet werden. Außerdem folgen sie einem komponentenorientierten Ansatz, doch dazu später mehr. An dieser Stelle aber enden die Gemeinsamkeiten auch schon. Jede der angesprochenen Lösungen hat ihre Daseinsberechtigung und kann auf eine starke und sehr aktive Community zurückgreifen. Alle drei spielen ihre Stärke auf einem anderen Gebiet aus und verfügen über eine eigene Charakteristik:

- **Angular**: Struktur und Ordnung, diese beiden Begriffe beschreiben Angular meiner Meinung nach am besten im Vergleich zu den anderen Frameworks. Angular wurde von Google als Frontend-Framework entwickelt und bringt für die meisten Problemstellungen bereits eine Standardlösung mit. Das Framework setzt auf der Programmiersprache TypeScript (*https://www.typescriptlang.org*) auf und macht einige Vorgaben hinsichtlich des Aufbaus und der Struktur von Applikationen. Die Richtlinien des Frameworks geben klare Empfehlungen für den Aufbau der Ver-

zeichnis- und Dateihierarchie. Außerdem ist ein Rahmen für die Implementierung der Bestandteile einer Applikation festgelegt, der auch beschreibt, wie die einzelnen Teile der Applikation miteinander interagieren und wo sie im Dateisystem platziert werden sollten.

- **Vue**: Ein großer Vorteil von Vue gegenüber Angular ist, dass die Einstiegshürde deutlich niedriger ist. Mit Vue können Sie schneller loslegen und auch schneller Ergebnisse erzielen. Das hat allerdings den Nachteil, dass etwas auf Strukturen verzichtet wird. Vue ist das einzige der drei großen Frameworks, hinter dem kein großes Unternehmen, sondern nur ein einzelner Hauptentwickler steht.
- **React**: Wenn Angular Struktur und Ordnung ist, ist React Freiheit und Chaos. Der Schwerpunkt von React ist im Gegensatz zu Angular nicht, eine Komplettlösung zu bieten, sondern sich stattdessen auf die Erstellung von User Interfaces zu konzentrieren. Genau genommen handelt es sich bei React also nicht um ein vollwertiges Framework, sondern um eine Bibliothek. Mit diesem Ansatz bietet React das Grundgerüst für eine Applikation, macht allerdings keinerlei Vorgaben, wenn es um die Struktur geht. Alle weiteren Elemente, die Sie für eine Applikation benötigen, müssen Sie als zusätzliche Bibliotheken oder Eigenimplementierung einbinden.

Geht es um die Umsetzung einer Single-Page-Applikation, schränkt Sie keine der drei Lösungen ein, sodass Sie sich frei für eine der drei entscheiden können. Als kleine Entscheidungshilfe können Sie einen Blick auf verschiedene Statistiken werfen. Die populärsten sind hier die Stars auf GitHub, die einen Hinweis auf die Größe der Community geben, aber auch die wöchentlichen Downloads des jeweiligen Pakets auf npm. Abbildung 10.2 zeigt einen Screenshot der Website *http://npmtrends.com* mit dem Verlauf der wöchentlichen npm-Downloads der drei Lösungen über die letzten zwei Jahre.

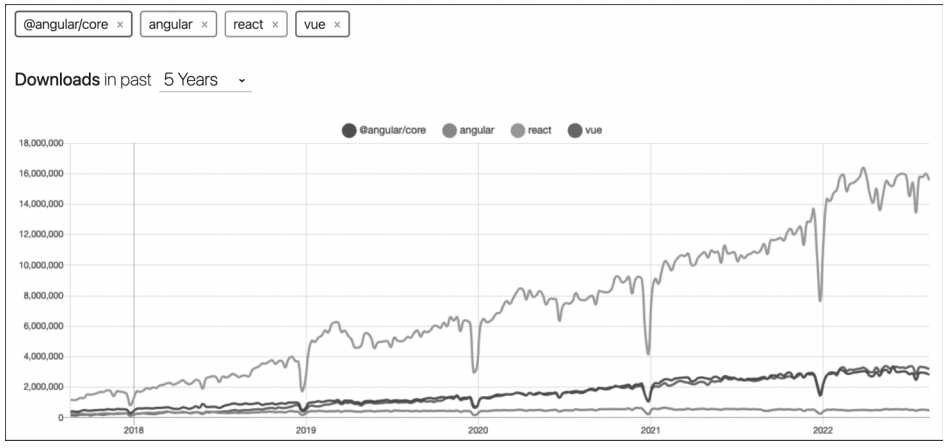


Abbildung 10.2 Downloadzahlen von Angular, React und Vue (https://www.npmtrends.com)

Wie Sie sehen können, liegen Angular und Vue relativ gleichauf, nur React wird deutlich häufiger heruntergeladen. Diese Statistik ist, wie so ziemlich jede Statistik über Open-Source-Projekte, mit Vorsicht zu genießen, da sie beispielsweise durch Bots leicht manipuliert werden kann und lokale npm-Repositorys nicht einbezieht.

Warum React?

In diesem Kapitel werfen wir einen Blick auf die Implementierung einer Applikation mit React. Da stellt sich natürlich die Frage: Warum React und nicht eines der anderen Frameworks? Die Frage ist einfach beantwortet: persönliche Vorliebe. Ich arbeite tagtäglich mit React und habe seine Flexibilität und den leichtgewichtigen Charakter der Bibliothek schätzen gelernt. Sie werden auf den folgenden Seiten sehen, dass die so gefürchtete Einstiegshürde von React in Wirklichkeit gar nicht so groß ist und Sie sehr schnell produktiv arbeiten können.

10.2 Setup

Eine React-Applikation besteht in der Regel aus einer Vielzahl kleiner Dateien, die gegenseitige Abhängigkeiten aufweisen, die Sie über das JavaScript-Modulsystem auflösen. Außerdem können Sie weitere Werkzeuge wie beispielsweise TypeScript oder CSS-Präprozessoren einsetzen. Der Buildprozess einer Applikation wird dadurch sehr schnell aufwendig und komplex. Aus diesem Grund gibt es mit dem Create-React-App-Projekt ein Kommandozeilenwerkzeug, das Ihnen das initiale Setup Ihrer Applikation erleichtert und alle erforderlichen Bibliotheken so miteinander kombiniert, dass Sie gleich mit der eigentlichen Arbeit an Ihrer Applikation starten können. Create React App wird, wie auch React selbst, von Facebook entwickelt und der Quellcode auf GitHub verwaltet. Create React App steht als npm-Paket zur Verfügung und kann entweder als globales Paket auf dem System installiert und verwendet werden, was allerdings mittlerweile nicht mehr empfohlen wird, oder Sie nutzen ein Feature Ihres Paketmanagers, das Ihnen die temporäre Verwendung eines solchen Pakets erlaubt. Je nachdem, ob Sie npm oder Yarn als Paketmanager nutzen, können Sie eine der folgenden Varianten nutzen:

- `npm create react-app <app-name>`
- `npx create-react-app <app-name>`
- `yarn create react-app <app-name>`

Mit dem Kommando `npm create react-app contact` erzeugen Sie also ein neues React-Projekt mit dem Namen *contact*. In diesem binden wir die REST-Schnittstelle aus Kapitel 16, »Webservices implementieren«, an, um eine einfache grafische Oberfläche zur Verwaltung von Kontakten zu implementieren.

Create React App akzeptiert zur Steuerung des Erstellungsprozesses einer neuen Applikation einige Optionen. Die wichtigsten sind:

- `--use-npm`: Standardmäßig nutzt Create React App Yarn als Paketmanager. Falls Sie das nicht wünschen, können Sie die Verwendung von npm mit dieser Option erzwingen.
- `--template <template>`: Create React App unterstützt den Einsatz von sogenannten Templates. Dabei handelt es sich um Vorlagen für den Aufbau und die Konfiguration einer React-Applikation. Ein sehr populäres Beispiel für ein solches Template ist das TypeScript-Template. Dieses bereitet Ihre Applikation für den Einsatz von TypeScript vor, sodass Sie React nicht mit JavaScript, sondern mit TypeScript entwickeln.

Weitere Informationen über die Verwendungsmöglichkeiten von Create React App finden Sie in der Dokumentation des Projekts unter *https://create-react-app.dev/docs/getting-started*.

Struktur der Applikation

Die von Create React App erzeugte Struktur ist in drei Teile unterteilt: das Basis-, das *public*- sowie das *src*-Verzeichnis. Die verschiedenen Sektionen erfüllen bei der Entwicklung unterschiedliche Zwecke:

- Im Wurzelverzeichnis finden Sie die wichtigsten Konfigurationsdateien Ihres Projekts. Hier liegen mit *package.json* und *package-lock.json* die Dateien, die npm zur Verwaltung des Projekts benötigt. Außerdem finden Sie hier die *.gitignore*-Datei, die bereits über einen Satz von Standardeinträgen von Dateien und Verzeichnissen verfügt, die nicht in die Versionskontrolle aufgenommen werden sollen. Die Datei *README.md* enthält die Dokumentation des Projekts. Hier können Sie alles festhalten, was für Entwickler, die am Projekt teilnehmen, wichtig ist.
- Das *public*-Verzeichnis enthält statische Inhalte, die von einem Webserver ausgeliefert werden. Hierunter fallen beispielsweise Mediendateien, aber auch die Datei *index.html*, die zentrale Einstiegsdatei in die Applikation.
- Die meiste Zeit werden Sie während der Entwicklung im *src*-Verzeichnis verbringen. Hier speichern Sie den Quellcode Ihrer Komponenten und aller anderen JavaScript-Dateien. Create React App legt in diesem Verzeichnis eine *index.js*-Datei an, die den Einstieg in die Applikation darstellt, und eine erste Komponente mit dem Dateinamen *App.js*.

Ihre Applikation ist mit dem von Create React App generierten Stand bereits lauffähig, sodass Sie sie auf der Kommandozeile im Wurzelverzeichnis der Applikation mit dem Kommando `npm start` starten können. Mit der Bestätigung des Befehls passiert Folgendes: Die Applikation wird im Watch-Modus gestartet, der Standardbrowser Ihres Systems wird geöffnet, und die Applikation wird angezeigt. Der Watch-Modus

bedeutet, dass ein Skript die Dateien Ihrer Applikation beobachtet und den Browser automatisch mit der neuesten Version des Quellcodes aktualisiert, sobald Sie Änderungen an einer Datei speichern. Standardmäßig ist die Applikation unter *http://localhost:3000/* erreichbar. Ist dieser Port bereits belegt, wird die Applikation auf einen anderen freien Port gebunden.

Bei der Datei *index.html* im Verzeichnis *public* handelt es sich um eine einfache HTML-Datei. Diese weist lediglich eine Besonderheit auf: Es existiert ein `<div>`-Element, das den Wert `root` für das `id`-Attribut hat. In diesen Container fügt React die Applikation ein. Für einen Benutzer bedeutet das, dass zunächst für kurze Zeit eine weiße Seite angezeigt wird und dann erst die Applikation sichtbar ist. In der Regel ist diese weiße Seite nur so kurz zu sehen, dass es einem Benutzer nicht auffällt. Sollte es für Ihre Applikation dennoch länger dauern, weil die Applikation etwa sehr umfangreich ist, können Sie dieses Problem mithilfe von *Serverside Rendering* lösen.

Serverside Rendering

Die Idee hinter Serverside Rendering ist, dass die Single-Page-Applikation nicht erst im Browser des Benutzers aufgebaut wird, sondern bereits serverseitig vorbereitet wird.

Zu diesem Zweck wird React in einer Node.js-Umgebung ausgeführt und verrichtet die gleiche Arbeit wie auch im Browser – mit dem Unterschied, dass die HTML-Struktur nicht direkt angezeigt wird, sondern in einer JavaScript-Zeichenkette, die dann zum Browser gesendet wird. Das Ergebnis ist, dass der Benutzer bereits eine vorbereitete React-Applikation erhält. React übernimmt anschließend nur noch die Kontrolle über die bis dahin statische HTML-Struktur und sorgt dafür, dass diese dynamisch auf die Interaktionen des Benutzers reagieren kann. Dieser Prozess wird *Hydration* genannt. Ist er abgeschlossen, verhält sich die Applikation wie eine reguläre React-Applikation.

Der Buildprozess von React modifiziert die Datei *index.html*, sodass die verarbeiteten und optimierten JavaScript-Dateien der Applikation geladen werden. Als erste Datei wird hier *index.js* geladen. In Listing 10.1 sehen Sie einen Auszug aus dieser Datei.

```
const root = ReactDOM.createRoot(
  document.getElementById('root') as HTMLElement
);
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Listing 10.1 Auszug aus der index.js-Datei

React ist in die zwei Teile Renderer und Reconciler unterteilt. Der Renderer sorgt dafür, dass die Struktur der Applikation für die Zielplattform aufgebaut wird. Der Reconciler, das Herzstück von React, verwaltet den Komponentenbaum und berechnet bei Änderungen die neue Version der Applikation, die dem Benutzer präsentiert wird. Der Reconciler ist so implementiert, dass er nicht plattformspezifisch ist, er kann also sowohl im Browser als auch für mobile Applikationen verwendet werden. Der Renderer hingegen ist plattformspezifisch. In unserem Beispiel kommt der DOM-Renderer zum Einsatz, der React im Browser darstellt. Ein weiterer populärer Renderer ist beispielsweise React Native, mit dem Sie native mobile Apps erzeugen können.

In Listing 10.1 übergeben Sie der `render`-Methode des `root`-Objekts eine Komponentenhierarchie, die in JSX geschrieben ist. Das `root`-Objekt erzeugen Sie mit der createRoot-Methode, der Sie eine Referenz auf den DOM-Knoten übergeben, in den die Applikation eingefügt wird.

JSX – JavaScript and XML

Bei JSX handelt es sich um eine Syntaxerweiterung für JavaScript, die es Ihnen erlaubt, HTML-Zeichenketten direkt in JavaScript zu schreiben. Dabei gibt es einen Unterschied zwischen React-Elementen und React-Komponenten. Elemente beginnen mit einem Kleinbuchstaben und werden direkt in HTML-Tags übersetzt. Ein Beispiel hierfür ist `<div></div>`. Komponenten beginnen mit einem Großbuchstaben und müssen vor ihrer Verwendung zunächst importiert werden. Den Komponenten liegen JavaScript-Funktionen beziehungsweise -Klassen zugrunde. Ein Beispiel für die Einbindung einer Komponente ist `<App />`.

Innerhalb eines JSX-Ausdrucks können Sie wiederum JavaScript schreiben, allerdings müssen Sie dieses in geschweifte Klammern einfassen. Und so lassen sich JSX und JavaScript beliebig ineinanderschachteln.

Bei JSX handelt es sich um gewöhnliche JavaScript-Ausdrücke, die Sie Variablen zuweisen oder als Rückgabewert von Funktionen verwenden können.

Sie müssen JSX nicht zwingend in einer React-Applikation verwenden. Statt Elemente und Komponenten als Tags zu schreiben, können Sie auch die `createElement`-Funktion nutzen. Der Nachteil dieser Variante ist, dass der Quellcode sehr schnell unübersichtlich wird und gerade die Verschachtelung hier nur umständlich abgebildet werden kann.

Die `StrictMode`-Komponente, die in der Datei *index.js* verwendet wird, ist mit dem Strict Mode von JavaScript vergleichbar. Diese Komponente schaltet zusätzliche Überprüfungen und Warnungen für React-Applikationen ein, um potenzielle Fehler-

quellen aufzuzeigen. So werden Sie beispielsweise gewarnt, wenn Sie unerwartete Seiteneffekte auslösen oder unsichere Lifecycle-Methoden verwenden. Mehr zum Thema StrictMode erfahren Sie unter *https://reactjs.org/docs/strict-mode.html*.

Die Datei *index.js* bindet standardmäßig die `App`-Komponente ein. Dabei handelt es sich um eine von Create React App vorbereitete Wurzelkomponente. Sie bildet den Einstieg in den Komponentenbaum und damit auch in unser nächstes Thema: Komponenten in React.

10.3 Komponenten – die Bausteine einer React-Applikation

In React gibt es zwei Arten von Komponenten: *Klassenkomponenten* und *Funktionskomponenten*. Wie die Namen bereits andeuten, basieren die einen auf Klassen, die anderen auf Funktionen. Lange Zeit war die Rede von Smart- und Dumb-Components, also von intelligenten und dummen Komponenten. Das spielte auf die Tatsache an, dass Funktionskomponenten bis vor einigen Releases nur zur Darstellung von Strukturen verwendet werden konnten, aber über keinen eigenen Zustand oder Lifecycle verfügen konnten. Das änderte sich jedoch mit dem Release 16.8 von React, in dem die Hook-API eingeführt wurde. Seit dieser Version können Sie auch in Funktionskomponenten einen lokalen Zustand und den Lebenszyklus der Komponente verwalten. Dieses Release markierte auch den Niedergang der schwergewichtigeren Klassenkomponenten bis zu dem Punkt, an dem heute kaum noch Klassenkomponenten in modernen React-Applikationen eingesetzt werden. Aus diesem Grund konzentrieren wir uns in diesem Kapitel auch ausschließlich auf Funktionskomponenten.

Ein großer Vorteil von React ist, dass Komponenten sehr leichtgewichtig sind. Eine Komponente besteht aus einer Funktion, die eine JSX-Struktur zurückgibt. Im ersten Schritt beginnen wir mit einer statischen `List`-Komponente, die die Kontaktliste anzeigen soll, die wir später von der REST-API holen. Listing 10.2 enthält den Quellcode der statischen `List`-Komponente.

```
const contacts = [
  {
    id: 1,
    firstName: 'Max',
    lastName: 'Mustermann',
    email: 'maxmustermann@example.com',
  },
  {
    id: 2,
    firstName: 'Erika',
    lastName: 'Mustermann',
```

```
    email: 'erikamustermann@example.com',
  },
];

function List() {
  return (
    <table>
      <thead>
        <tr>
          <th>ID</th>
          <th>Vorname</th>
          <th>Nachname</th>
          <th>E-Mail</th>
        </tr>
      </thead>
      <tbody>
        {contacts.map((contact) => (
          <tr key={contact.id}>
            <td>{contact.id}</td>
            <td>{contact.firstName}</td>
            <td>{contact.lastName}</td>
            <td>{contact.email}</td>
          </tr>
        ))}
      </tbody>
    </table>
  );
}

export default List;
```

Listing 10.2 Komponente zur Darstellung der Kontaktliste (src/List.js)

Für die Listendarstellung nutzen wir das `contacts`-Array, das aus zwei Einträgen besteht. Das Array wird außerhalb der Komponente definiert, da es unabhängig von dieser ist. Außerdem wird die Komponentenfunktion unter bestimmten Bedingungen von React häufiger aufgerufen, was dazu führt, dass das `contacts`-Array ebenfalls neu erzeugt werden würde. Der Kern der Komponente ist die `List`-Funktion. Wie Sie wissen, unterscheiden sich Komponenten von Elementen durch einen großen ersten Buchstaben, also beginnt auch die Komponentenfunktion mit einem Großbuchstaben. Die Komponente gibt eine HTML-Tabelle zurück. JSX ist in gewisser Weise die Template-Sprache von React mit dem Unterschied, dass JSX über keine Möglichkeit logischer Operationen oder Schleifen verfügt. In diesem Fall müssen Sie auf natives

JavaScript zurückgreifen. Im Fall der `List`-Komponente soll für jeden Datensatz eine Tabellenzeile erzeugt werden. Eine solche Art von Iteration über eine Array-Struktur erreichen Sie in React in der Regel über die `map()`-Methode des Arrays. Im Gegensatz zur `forEach()`-Methode produziert `map()` eine Ausgabe, in diesem Fall ein Array von JSX-Elementen, die in die umgebende Struktur eingebunden und schließlich gerendert werden. So entsteht dann die finale Tabelle. Im Beispiel fällt außerdem das `key`-Attribut im `tr`-Element auf. Dieses wird von React intern zur Zuordnung von Elementen verwendet. Über dieses `key`-Attribut ist React in der Lage zu entscheiden, ob ein Listenelement bei einer Änderung der Darstellung wiederverwendet werden kann oder ob ein Element neu erzeugt werden muss. Es handelt sich hierbei also um eine Performanceoptimierung.

Damit Sie das Ergebnis im Browser sehen können, müssen Sie die `List`-Komponente zunächst exportieren, was mit dem `export default`-Statement in der letzten Zeile geschieht. Anschließend binden Sie die Komponente in die Applikation ein. Eine passende Stelle ist die App-Komponente in der Datei *App.js*. Die angepasste Version dieser Datei sehen Sie in Listing 10.3.

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

function App() {
  return <List />;
}

export default App;
```

Listing 10.3 Einbindung der List-Komponente (src/App.js)

Von der generierten App-Komponente bleibt lediglich der Rahmen der `App`-Funktion sowie der `default export` bestehen. Den Rest können Sie löschen und stattdessen die `List`-Komponente importieren und einbinden. Wechseln Sie nun in Ihren Browser, sehen Sie die Kontaktliste wie in Abbildung 10.3.

ID	Vorname	Nachname	E-Mail
1	Max	Mustermann	maxmustermann@example.com
2	Erika	Mustermann	erikamustermann@example.com

Abbildung 10.3 Listenansicht im Browser

10.3.1 State – der lokale Zustand einer Komponente

Das Problem mit der bisherigen Implementierung der `List`-Komponente ist, dass sie vollständig statisch ist. Die Komponente bezieht die Daten zur Anzeige aus einem

konstanten JavaScript-Array. Auch wenn Sie das Array jetzt anpassen würden, beispielsweise, indem Sie nach 10 Sekunden ein neues Element in das Array einfügen, bleibt die Anzeige wie bei der initialen Darstellung. Das liegt an der Art, wie React Änderungen an Komponenten verfolgt und diese darstellt: Eine Komponente wird neu gezeichnet, wenn sich ihr lokaler Zustand, der State, ändert oder wenn sie veränderte Eingabeparameter, auch Props genannt, erhält. Zunächst widmen wir uns dem State der Komponente und sehen uns später die Props einer Komponente genauer an.

Mit der `useState`-Funktion können Sie einen lokalen Zustand für eine Komponente definieren. Diese Funktion akzeptiert einen initialen Wert für den State der Komponente und gibt ein Array zurück. Das erste Element dieses Arrays ist ein Objekt, über das Sie lesend auf den State der Komponente zugreifen können. Mit der Funktion, die Sie als zweites Array-Element erhalten, können Sie den State verändern. Eine Komponente wird nur neu gezeichnet, wenn Sie den State über diese Funktion manipulieren. Für unsere Kontaktliste bedeutet das zunächst nur wenig Veränderung, wie Sie in Listing 10.4 sehen können.

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

const initialContacts = […];

function List() {
  const [contacts, setContacts] = useState(initialContacts);
  return (
    <table>…</table>
  );
}

export default List;
```

Listing 10.4 Lokaler State in der List-Komponente

Das bestehende Array benennen Sie in `initialContacts` um, um deutlich zu machen, dass es sich hierbei um den Startwert des lokalen States handelt. Bei der Verwendung der `useState`-Funktion wird in der Regel das Destructuring-Feature von JavaScript verwendet, bei dem Sie angeben können, dass das erste Array-Element in unserem Fall in der Konstanten `contacts` und das zweite Element in der Konstanten `setContacts` gespeichert wird.

Nutzen Sie nun die `setContacts`-Funktion, um den State zu manipulieren, zeichnet React die Komponente neu. Und genau an dieser Stelle entstehen die Probleme für uns: Rufen Sie innerhalb der Komponenten-Funktion ein `setContacts` auf, führt dies zu einer Veränderung des States; die Komponente wird neu gezeichnet, also muss

React die Komponenten-Funktion erneut ausführen, was ein abermaliges `setContacts` nach sich zieht, sodass sich Ihre Applikation in einer Endlosschleife befindet. Die Lösung ist an dieser Stelle, den Lebenszyklus der Komponente zu nutzen, um eine Aktion einmalig auszuführen.

10.3.2 Der Lebenszyklus einer Komponente

Der Lebenszyklus einer React-Komponente besteht aus drei Stationen:

- **Mount**: Die Komponente wird initial in den Komponentenbaum eingehängt.
- **Update**: Der State oder die Props der Komponente ändern sich.
- **Unmount**: Die Komponente wird ausgehängt. An dieser Stelle sollten Sie geöffnete Ressourcen wie beispielsweise Datenströme wieder schließen.

Ähnlich wie schon beim State kommt auch bei der Implementierung des Lifecycles einer Komponente die Hook-API von React in Form der `useEffect`-Funktion zum Einsatz. Sie akzeptiert zwei Parameter: eine Funktion, die Effekt-Funktion, und ein Array mit Abhängigkeiten. Die Funktion wird standardmäßig bei jeder Aktualisierung ausgeführt. Ist dies nicht erwünscht, können Sie das Verhalten mit dem zweiten Parameter beeinflussen. Geben Sie ein leeres Array an, wird die Effekt-Funktion lediglich einmalig beim Einhängen der Komponente ausgeführt. Dieser Fall deckt also das Mounten der Komponente ab. Geben Sie im Abhängigkeits-Array Referenzen auf Variablen an, wird die Effekt-Funktion nur ausgeführt, wenn sich diese Variablen ändern.

Um die dritte Phase des Lebenszyklus der Komponente abzudecken, definieren Sie eine Funktion, die die Effekt-Funktion zurückgibt. Diese wird ausgeführt, wenn die Komponente ausgehängt, also nicht weiter dargestellt wird.

Generell sollten Sie darauf achten, dass Sie keinerlei Seiteneffekte wie beispielsweise Timeouts, Intervalle oder auch Serverkommunikation direkt in einer Komponente ausführen, sondern immer in einer Effekt-Funktion. Der Grund hierfür ist, dass die Ausführung der Komponenten-Funktion von React abgebrochen und entweder zu einem späteren Zeitpunkt weiter ausgeführt oder neu gestartet werden kann.

Für unsere `List`-Komponente bedeutet das, dass Sie eine Effekt-Funktion mit einem leeren Abhängigkeits-Array definieren und dort mit der REST-API kommunizieren, um die Daten zu laden. Sobald der Server geantwortet hat, können Sie die Informationen in den State schreiben. Listing 10.5 zeigt die Integration des Effect-Hooks.

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

function List() {
  const [contacts, setContacts] = useState([]);
```

```
  useEffect(() => {
    fetch('http://localhost:8001/api/contacts')
      .then((response) => response.json())
      .then((data) => setContacts(data));
  }, []);

  return (
    <table>…</table>
  );
}

export default List;
```

Listing 10.5 Serverkommunikation in der List-Komponente (src/List.js)

Durch das leere Array, das der `useState`-Funktion beim Aufruf übergeben wird, stellt React bei der initialen Darstellung der Komponente eine leere Liste dar. Anschließend wird die Effekt-Funktion ausgeführt. Das leere Abhängigkeits-Array sorgt dafür, dass die Daten nur einmalig beim Einhängen der Komponente vom Server geladen werden. Innerhalb der Effekt-Funktion senden Sie mithilfe der Fetch-API einen `GET`-Request an die REST-API, um die Daten auszulesen. Im ersten Schritt behandeln Sie die asynchrone Rückmeldung des Servers mithilfe einer Promise und extrahieren daraus die JSON-Daten. Diese wiederum asynchrone Operation liefert schließlich die eigentlichen Daten, die Sie mithilfe der `setContacts`-Funktion in den lokalen State der Komponente schreiben. Dieser Funktionsaufruf sorgt dafür, dass die Komponente mit den Daten des Servers neu gezeichnet wird. Im nächsten Schritt sorgen wir nun dafür, dass die Applikation auch optisch etwas ansprechender gestaltet wird.

10.4 Styling von Komponenten

Zugegebenermaßen sieht die Kontaktliste aktuell noch nicht sonderlich atemberaubend aus. Das liegt jedoch vor allem an der Tatsache, dass wir uns bisher lediglich mit der Struktur, also dem HTML, und der Dynamik, also JavaScript und React selbst, beschäftigt haben und das Thema Styling komplett ignoriert haben. React bietet Ihnen an dieser Stelle einige Möglichkeiten, um Komponenten mithilfe von CSS zu stylen. Grundsätzlich können Sie auf das gesamte Repertoire an CSS-Features zurückgreifen.

Inhalt

7 Web-APIs verwenden 221

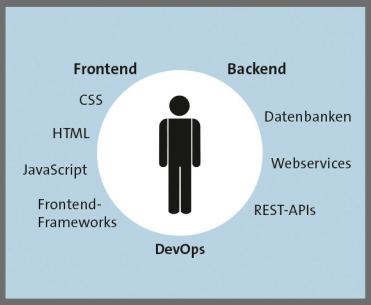
8 Webseiten für Barrierefreiheit optimieren 247

Fullstack-Entwicklung

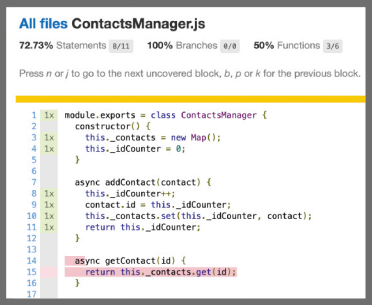
»Das neue Standardwerk für Webentwickler«

Ihr Guide zur Fullstack-Webentwicklung

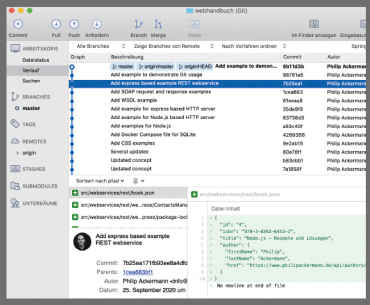
Moderne Webanwendungen setzen auf das Zusammenspiel zahlreicher Technologien am Front- und Backend. Wie Sie die passenden Frameworks, Sprachen und Tools einsetzen und im Dschungel der Webentwicklung nicht den Überblick verlieren, zeigt Ihnen Philip Ackermann in diesem umfassenden Handbuch.



Fullstack im Überblick



Webprogrammierung verstehen



Webprojekte managen

Grundlagen des Webs verstehen

Wenn Sie die Zusammenhänge zwischen Front- und Backend verstehen wollen, brauchen Sie ein gutes Fundament. Von Grund auf zeigt Ihnen dieser Entwickler-Guide, wie Sie moderne Webanwendungen programmieren.

Front- und Backend meistern

Lernen Sie das Gesamtsystem moderner Webarchitekturen und Programmiersprachen kennen. Dieses Buch hilft Ihnen, die Zusammenhänge der Elemente einer Webapplikation zu verstehen. Vertiefen Sie Ihr Wissen über HTML, CSS und JavaScript, die Grundsteine jeder modernen Webseite.

Webprojekte organisieren und managen

Machen Sie Ihre Projekte fit für den Einsatz: Lernen Sie, wie Sie Ihre Webanwendungen testen, deployen und verwalten. Erfahren Sie, wie Sie Versionen organisieren, Sicherheitslücken vorbeugen, die Webperformance optimieren und mit Scrum arbeiten.

Philip Ackermann ist CTO der Cedalo GmbH und Autor zahlreicher IT-Bücher und Fachartikel zu JavaScript, Java und Node.js. Als langjähriger Webentwickler gibt er sein Wissen in der Fullstack-Entwicklung in diesem Buch weiter.

Aus dem Inhalt

- Webseiten erstellen mit HTML, CSS, JavaScript und PHP
- Webprotokolle und -formate kennenlernen und verwenden
- Web-APIs: DOM, Ajax u. v. m.
- Barrierefreie Webseiten
- Single-Page-Applikationen mit React
- Mobile Webanwendungen
- Serverseitige Programmierung
- Webservices: SOAP, REST, GraphQL
- Relationale Datenbanken und NoSQL
- Webanwendungen testen, deployen, hosten und absichern
- Webprojekte organisieren und verwalten
- Projektmanagement mit Scrum

Rheinwerk Computing

€ 49,90 [D] € 51,30 [A]

Gedruckt in Deutschland
Papier aus nachhaltiger Waldwirtschaft
Mineralölfreie Druckfarben

Für Windows, macOS und Linux
Webseiten

ISBN 978-3-8362-9084-5