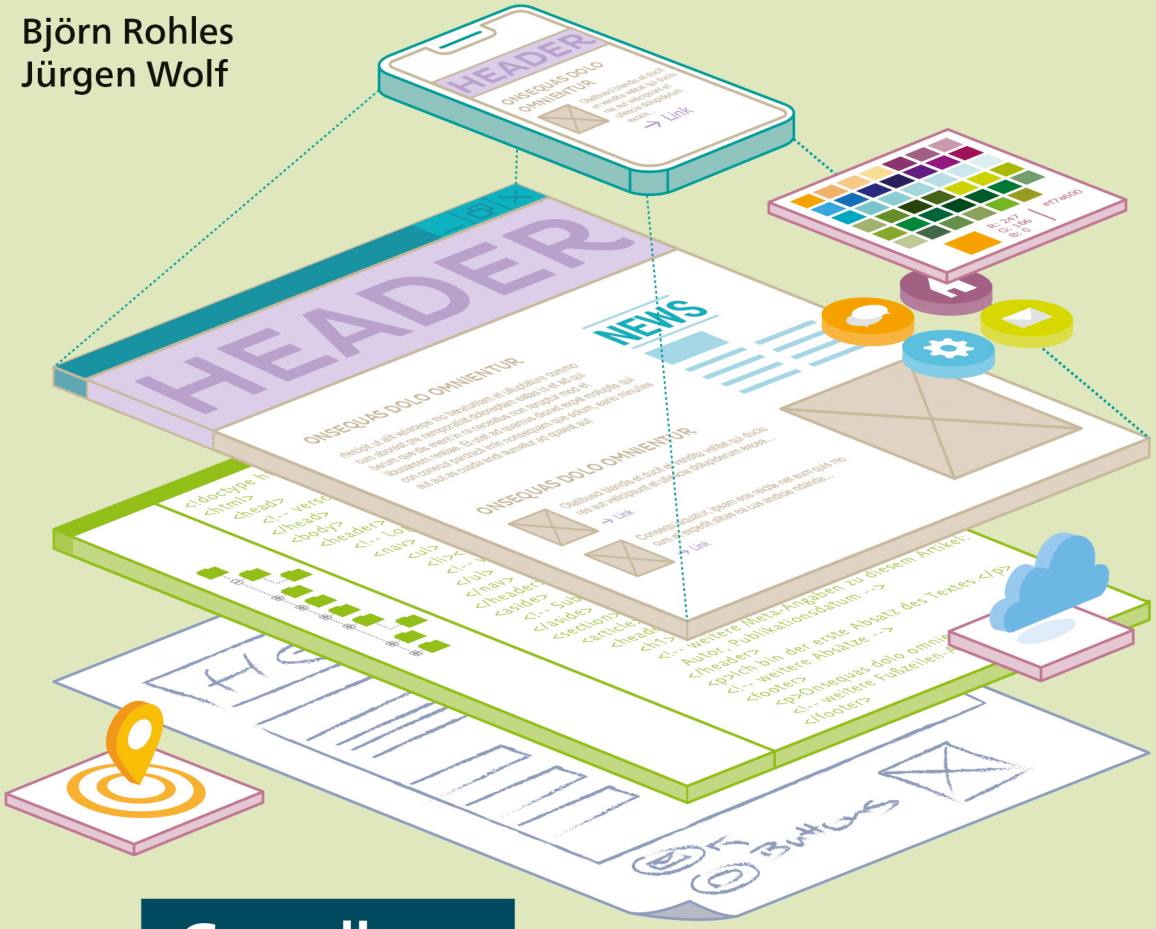
Björn Rohles
Jürgen Wolf



Grundkurs Gutes Webdesign

Alles, was Sie über Gestaltung im Web wissen sollten

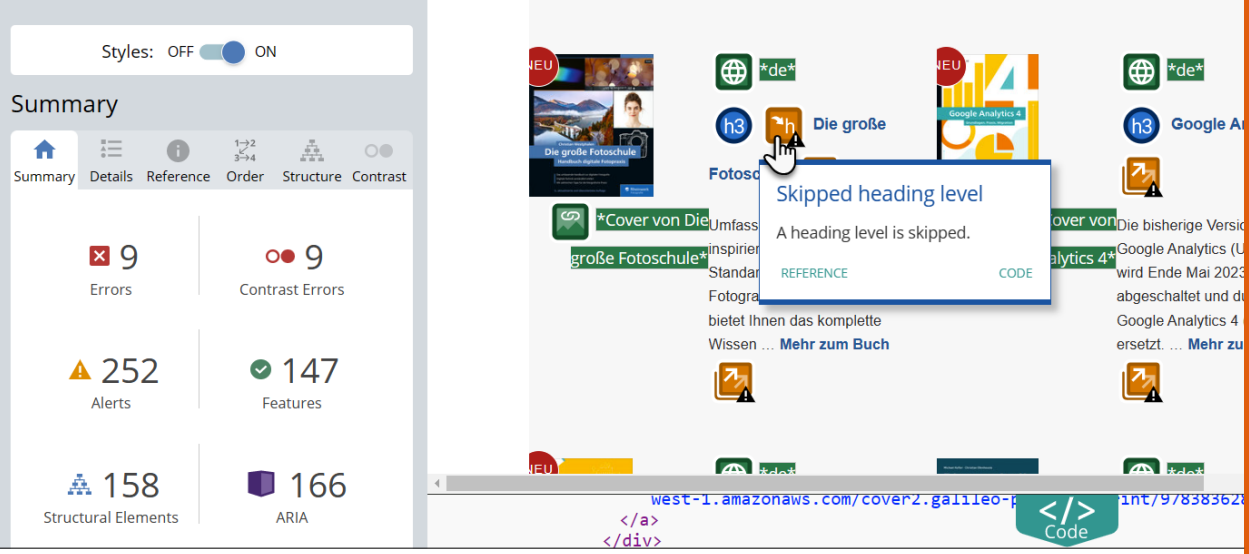
1476 px

154 px

- Konzeption, CSS-Layout, Test und Optimierung
- Typografie, Farbe und Grafiken im Web
- Usability und UX, Responsive Webdesign

Code-Beispiele und Bonusinhalte zum Download

Rheinwerk Design



Testen und optimieren

Wie Sie die Qualität einer Website sichern

- Wie teste ich Websites?
- Wie optimiere ich Ladezeiten?
- Wie überprüfe ich Usability und Accessibility?
- Wie achte ich auf Nachhaltigkeit?

9 Testen und optimieren

Konzeption, Struktur und Layout, Typografie, Farben, Grafiken – mit all diesen Mitteln haben Sie in den bisherigen Kapiteln gearbeitet, um eine gute Gestaltung zu erreichen. Ein Aspekt fehlt Ihnen noch – und er ist zugleich einer der wichtigsten: Qualitätssicherung. Lernen Sie in diesem Kapitel, wie Sie Ihre Website so testen und optimieren können, dass wirklich (fast) nichts mehr schiefgehen kann.

9.1 Funktionalitäten sicherstellen

Jeder Browser ist anders. Es ist daher absolut unmöglich, dass eine Website in allen Browsern *gleich* aussieht. Ihr Ziel sollte es jedoch sein, dass die Website in jedem Browser *gut* aussieht und funktioniert – wie das geht, erfahren Sie in diesem Abschnitt.

9.1.1 Browser-Statistiken abfragen

Zunächst einmal sollten Sie sich entscheiden, welche Browser überhaupt getestet werden sollen. Viele Designerinnen und Designer möchten darauf mit »Alle« antworten – das Problem dabei wäre jedoch, dass Ihnen das tagelange Anpassungen bescheren würde.

Sinnvoller ist es, die Entscheidung auf Basis von Statistiken zu treffen. Wenn Sie an einer Neugestaltung einer bestehenden Website arbeiten, gibt es häufig Erkenntnisse darüber, mit welchen Browsern das eigene Publikum arbeitet – nichts ist besser. Ansonsten müssen Sie sich an allgemeine Statistiken halten. Gute Quellen dafür sind Websites wie *www.w3counter.com/trends*, *http://gs.statcounter.com* und das Portal *https://de.statista.com*.

◀ **Abbildung 9.1**
Browser-Marktanteile in Deutschland im Juni 2022 (*https://gs.statcounter.com*)

9.1.2 Testumgebung vorbereiten

Da sich viele Browser weitgehend standardkonform verhalten, reicht es normalerweise, die aktuellen beiden Versionen zu testen. Eine Faustformel ist, die relevanten Browser auf allen Desktop- und Mobilplattformen sowie assistierende Technologien zu testen. Bei einigen Browsern ist es möglich, mehrere Versionen auf einmal zu installieren. Sinnvoll ist auch, sich mit Hilfe virtueller Maschinen und Lizenzen der wichtigsten Betriebssysteme Testumgebungen zu schaffen.

Tool-Tipp: Virtual Box
Virtual Box (*www.virtualbox.org*) ist eine kostenlos verfügbare Software-Lösung, mit der Sie verschiedene virtuelle Maschinen parallel installieren können.

Das Ende des Internet Explorers | Alte Versionen des Internet Explorers waren in der Vergangenheit eines der Hauptthemen beim Browser-Testen, weil sich der Browser oft nicht an Webstandards hielt und daher häufig spezifische Sonderlösungen notwendig waren. Seit 15. Juni 2022 gehört dies nun der Vergangenheit an, denn der Support des bis dahin noch relevanten Internet Explorers 11 wurde nun endlich eingestellt. Der Browser wird künftig per Windows Update automatisch entfernt, und alle Verknüpfungen des IE 11 zeigen nun auf den Microsoft-Browser Edge. Microsoft hat aber die alte IE-11-Engine neben der Chromium-Engine im Edge-Browser implementiert, um weiteres Testen zu ermöglichen.

Internet Explorer ignorieren?
Der Support des IE 11 für Privathaushalte ist vom Tisch. Allerdings kann es sein, dass er in einigen Behörden und Unternehmen noch ein wenig weiterlebt, bis zum Ende der LTSC-Laufzeit im Jahre 2029. Mit dem schwindenden Support für IE dürften jedoch auch diese Organisationen zunehmend vom Internet Explorer Abstand nehmen.

Alle Browser haben umfangreiche Entwicklertools an Bord, mit denen Sie verschiedenste Situationen simulieren können. Es lohnt sich, einen Blick in die Dokumentation und die Onlinekurse zu werfen, die von den Browser-Herstellern zur Verfügung gestellt werden – hier lernt man immer wieder Kniffe, die das Testen vereinfachen. Für Google Chrome erreichen Sie diese beispielsweise unter *https://developers.google.com/web/tools/chrome-devtools/*.

Breakpoints testen | Responsive Webdesign stellt Sie vor die Herausforderung, neben verschiedenen Browsern auch unterschiedliche Viewport-Breiten testen zu müssen. Das funktioniert sehr gut mit den Entwicklungstools der Browser. Abbildung 9.2 zeigt dies am Beispiel von Chrome. Mit einem Klick auf ❸ wird das Tool zum Testen von Responsive Designs aktiviert. Anschließend lässt sich das Browser-Fenster im Bereich ❶ manuell anpassen. Alternativ stehen unter ❷ verschiedene Voreinstellungen für Viewport, Zoomstufe und Pixeldichte bereit.

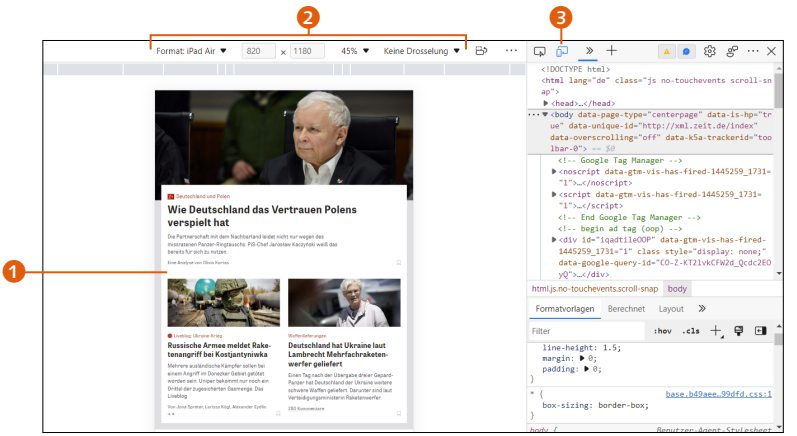
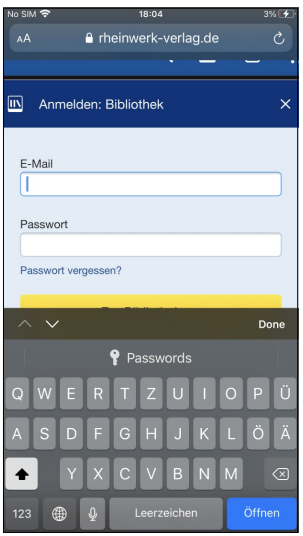


Abbildung 9.2 ▸
Testen des Responsive Designs mit Hilfe von Chrome

Geräte simulieren | Solche Browser-Tools sind ein gutes Hilfsmittel zum Testen von Breakpoints. Allerdings berücksichtigen sie nicht alle Aspekte, die für die User Experience auf mobilen Geräten relevant sind. Ein typisches Beispiel sind Formularfelder. Während ein Browser-Tool letztlich nicht das Verhalten des Browsers verändert, blendet ein Smartphone beim Aktivieren des Felds eine passende Tastatur ein.



▲ Abbildung 9.3
E-Mail-Formularfeld mit HTML auf dem iPhone-Simulator

In der App-Entwicklung wird mit Simulatoren und Emulatoren gearbeitet, die Teil der Entwicklungsumgebungen sind – damit lassen sich auch Websites testen. Die Möglichkeiten hängen dabei von der verwendeten Plattform ab. So stellt Apple seine Entwicklungssoftware Xcode ausschließlich für Macs bereit. Sie lässt sich kostenlos aus dem App Store herunterladen. Nach der Installation findet sich der iPhone-Simulator unter Xcode • Open Developer Tools • Simulator.

Die Entwicklungsumgebung für Android (»Android Studio«) steht hingegen für Mac, Windows und Linux unter *https://deve-*

loper.android.com/studio/index.html zur Verfügung. Die Software kommt mit einem umfangreichen Emulator, der virtuelle Geräte in allen möglichen Konfigurationen erlaubt. Google stellt unter *https://developer.android.com/studio/run/emulator.html* Informationen bereit, wie das Tool konfiguriert werden kann.

Natürlich verfügt auch Microsoft über eine Entwicklungsumgebung namens »Visual Studio Code« (*https://code.visualstudio.com)*.

Testen auf echten Geräten | Sinnvoll ist außerdem das Testen auf echten Geräten. Dazu haben sich sogenannte *Open Device Labs* etabliert – Testlabore, die Zugang zu verschiedenen Geräten bieten. Sie finden solche öffentlichen Testlabore in vielen größeren Städten vor (z. B. *http://odl-muc.de* in München oder *https://hamburg.opendevicelab.de* in Hamburg). Natürlich können Sie dort auch selbst Geräte spenden.

Die Alternative dazu lautet, sich ein eigenes kleines Device-Lab aufzubauen (»Own Device Lab«). Auf jeden Fall sollten Sie alte Smartphones und Tablets nicht wegwerfen, sondern ihnen eine Zweitkarriere als Testgerät spendieren. Eine sinnvolle Investition kann auch ein Service wie BrowserStack (*www.browserstack.com*) sein, mit dem Sie remote auf echten Geräten testen können.

Nun, da Sie Darstellungs- und UX-Probleme identifizieren können, taucht die nächste Frage auf: Wie kann ich darauf reagieren?

Gebrauchtmärkte
Auch auf Gebrauchtmärkten wie eBay können Sie zu geringen Preisen fündig werden – auch, weil es für ein Device Lab kein Problem darstellt, wenn das Display eines Geräts gesprungen ist oder der Akku nicht mehr allzu lange durchhält. Achten Sie dabei auf eine sinnvolle Auswahl von Geräten aller Preisklassen – auch einige exotische Kandidaten sollten dabei sein.

9.1.3 Feature-Unterstützung prüfen und reagieren

Ihr Ziel sollte sein, allen Nutzerinnen und Nutzern eine positive User Experience zu ermöglichen. Statt einzelne Browser anzusprechen, konzentriert sich die Webentwicklung zunehmend darauf, welche Features unterstützt werden, und liefert alternative Stilangaben.

Die wichtigste Plattform mit Informationen zur Browser-Unterstützung ist »Can I Use« (*https://caniuse.com*) – dort können Sie ein Feature eingeben und die Unterstützung prüfen. Was aber macht man, wenn eine relevante Zahl von Personen noch mit Browser-Versionen unterwegs ist, die eine CSS-Eigenschaft nicht unterstützen? Sie brauchen eine Möglichkeit, die unterstützten Features abzufragen.

Feature-Unterstützung-Bibliothek
Für einzelne APIs, Elemente, Methoden oder Eigenschaften kann sich die manuelle Implementation von Feature-Unterstützung mit JavaScript lohnen. Wenn Sie allerdings diese Feature-Unterstützung sehr regelmäßig verwenden, dann kann eine bewährte Feature-Unterstützung-Bibliothek wie Modernizr (*https://modernizr.com*) oder Feature.js (*https://featurejs.com*) sinnvoller sein.

Feature-Unterstützung mit JavaScript testen | Eine Möglichkeit ist es, die Feature-Unterstützung mit purem JavaScript zu prüfen, ohne extra Skripte oder eine Bibliothek verwenden zu müssen und Ressourcen zu verbrauchen. Gerade wenn Sie ein oder zwei Features auf deren Unterstützung hin prüfen wollen, ist eine Bibliothek etwas überdimensioniert. Möchten Sie z. B. prüfen, ob der Browser die Geolocation-API implementiert, um den aktuellen Standort abzufragen, können Sie dies mit folgenden Zeilen JavaScript tun:

```
if ("geolocation" in navigator) {
navigator.geolocation.getCurrentPosition(function(position) {
    // Position über eine Map wie Google anzeigen
  });
} else {
  // Alternative, weil die Geolocation-API nicht geht
}
```

▲ **Listing 9.1**
Feature-Überprüfung mit reinem JavaScript

Ähnlich können Sie auch mit einem HTML-Element vorgehen, indem Sie es mit JavaScript im Speicher via `Document.createElement()` erzeugen und dann überprüfen, ob es existiert. Zum Beispiel:

```
function supports_canvas() {
   return !!document.createElement('canvas').getContext;
}
…
if(supports_canvas()) {
   // Browser kann canvas
}
```

Listing 9.2 ▸
Feature-Überprüfung auf HTML-Elemente

Auf ähnliche Weise können Sie mit Hilfe von `Document.createElement()` ein Element anlegen und im Speicher prüfen, ob eine bestimmte Methode oder Eigenschaft dazu existiert.

Feature Queries mit `@supports` | JavaScript oder eine Bibliothek wie Modernizr funktionieren gut, allerdings fühlt es sich komisch an, ein JavaScript auszuführen, um CSS-Eigenschaften zu erkennen. Eine Alternative ist die Erkennung von CSS-Features in CSS

selbst mit `@supports`. So können Sie beispielsweise prüfen, ob der Browser die CSS-Flexbox unterstützt, und sie erst dann aktivieren, wenn das der Fall ist – in allen anderen Fällen greifen alternative Regeln. Hier ist ein Beispiel, das ein responsives Raster auf Basis von Flexbox aktiviert, sofern diese Technologie unterstützt wird:

```
.row { … } /* Standardangaben: Grid mit float … */
@supports ( display: flex ) {
  .row {  display: flex; } /* … Grid mit der Flexbox … */
}
```

◀ **Listing 9.3**
Erkennung von CSS-Features direkt in CSS

`@supports` erlaubt auch komplexere Abfragen mit den Operatoren AND, OR und NOT.

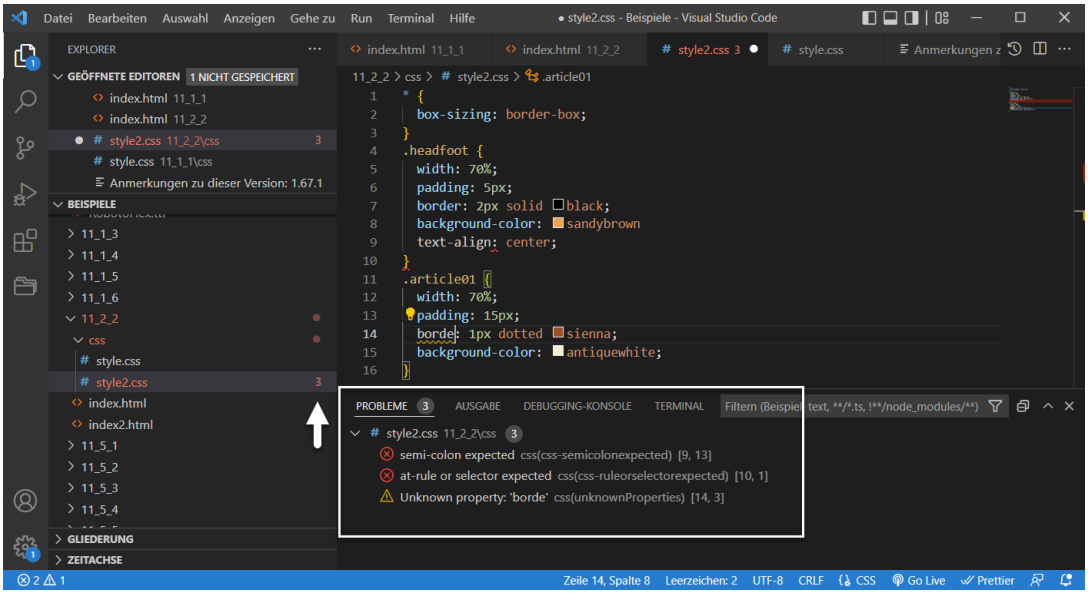
9.1.4 HTML und CSS validieren

HTML und CSS folgen klaren Regeln, und hin und wieder macht jeder beim Schreiben von Code einen Fehler. Bei der Suche danach hilft ein sogenannter **Validator**, dessen Aufgabe es ist, den HTML-Quelltext nach kleinen Schnitzern zu durchsuchen. Für validen Code sprechen zahlreiche Gründe, z. B. bessere Accessibility und Auffindbarkeit durch Suchmaschinen, konsistente Interpretation durch alle Browser sowie einfachere Wartbarkeit und Fehlersuche.

JSLint
Bei der Qualitätssicherung von JavaScript-Code kann beispielsweise JSLint (*http://jslint.com*) helfen.

Viele Webbrowser bieten bereits eine Funktion an, die die Validierung übernimmt. Auch Plug-ins stehen für Browser sowie HTML-Editoren bzw. Entwicklungsumgebungen zur Verfügung. Die offiziellen Validatoren des W3C für HTML (*http://validator.w3.org*) sowie CSS (*https://jigsaw.w3.org/css-validator*), der Living Validator (*http://html5.validator.nu)* oder Total Validator (*www.totalvalidator.com*) helfen Ihnen ebenfalls bei der Validierung. Den Quelltext zieht sich das Tool wahlweise per URL, Datei-Upload oder Direkteingabe. Nach kurzer Analyse erhalten Sie eine Übersicht von Fehlern.

Beachten Sie dabei: Validatoren sind Hilfsmittel, die typische Fehler wie falsche Verschachtelungen oder ungültige Elemente erkennen können. Sie liefern jedoch keine absoluten Wahrheiten – auch eine valide Seite kann unzugänglich oder unbenutzbar sein.



▲ Abbildung 9.4
Für uns unverzichtbar: eine Validation von HTML und CSS bereits während des Schreibens von HTML und CSS, wie hier mit Visual Studio Code von Microsoft

Neben den technischen Browser-Tests sollten Sie auch an die Menschen denken: Sind Usability und Accessibility so gut wie geplant?

9.2 Usability, User Experience und Accessibility testen

Zur Evaluation von Usability, User Experience und Accessibility können Sie auf eine Reihe von Hilfsmitteln zurückgreifen. Außerdem können Sie mit fortlaufenden Tests für eine stetige Verbesserung sorgen.

9.2.1 Accessibility mit Tools testen

Automatisierte Tests können Sie mit einer Reihe von Werkzeugen durchführen. Ein sehr guter Vertreter dieser Gattung ist das Tool WAVE (*https://wave.webaim.org*). Sie können es jedoch nur auf veröffentlichte Websites anwenden.

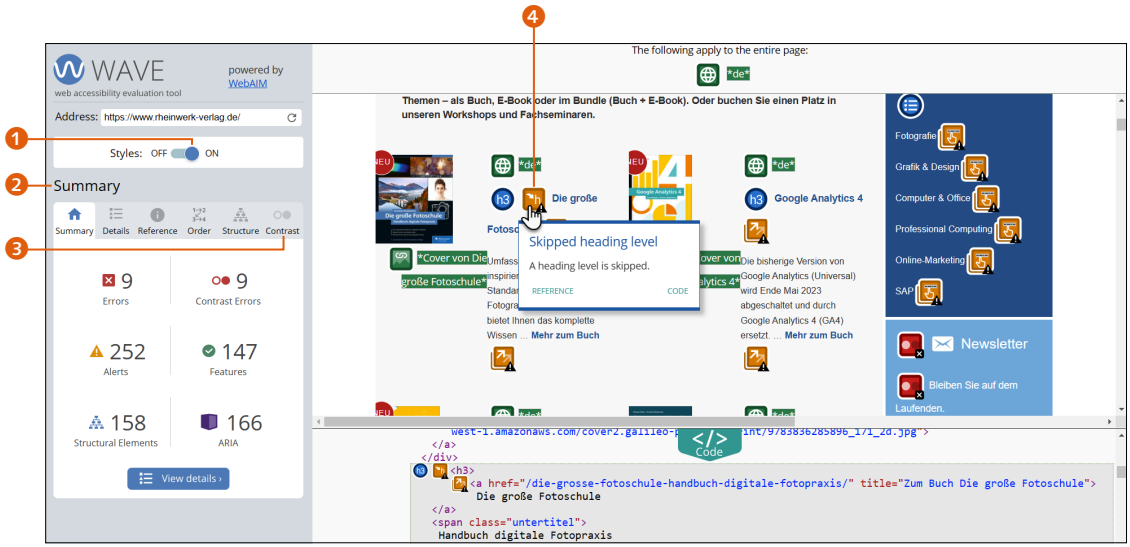
Nachdem Sie die URL in das Textfeld eingegeben haben, zeigt Ihnen WAVE eine Analyse möglicher Accessibility-Probleme an. Mit dem Schalter STYLES ❶ können Sie die Website mit und ohne CSS anschauen. Der Bereich SUMMARY ❷ listet die gefundenen Fehler nach Kategorie sortiert auf. Wichtig sind hier natürlich alle Angaben unter ERRORS, denn dabei handelt es sich um klare Fehler. Die anderen Bereiche sollten Sie genau prüfen, denn es handelt sich um Hinweise auf häufige Fehlerquellen. Durch einen Klick auf eines der Symbole ❹ erhalten Sie Details zu den problematischen Aspekten. Die Kontrastfehler können Sie sehr gut analysieren, wenn Sie die Schaltfläche CONTRAST unter ❸ wählen.

Stark

Das Unternehmen Stark (*www.getstark.co*) hat ein praktisches Tool entwickelt, um Accessibility-Probleme zu finden und zu lösen. Es lässt sich in Browser und zahlreiche Prototyping-Anwendungen integrieren. Einige Grundfunktionen sind kostenlos nutzbar.

Braillezeile und Screenreader

Sie sollten auch die Lesbarkeit mittels Braillezeile oder Screenreader überprüfen, wie blinde Menschen sie verwenden. Um hier einen Eindruck zu bekommen, wie die Website für blinde Menschen ausgegeben wird, haben sich NVDA (*www.nvaccess.org*) oder JAWS (*www.freedomsci.de*) bewährt.



▲ Abbildung 9.5
Analyse möglicher Accessibility-Probleme mit WAVE

Eine weitere gute Möglichkeit ist die Arbeit mit Browser-Erweiterungen wie den Lighthouse Accessibility Audits in Chrome (*https://developer.chrome.com/docs/lighthouse/accessibility/*). Damit lassen

sich Accessibility-Audits direkt im Browser durchführen. Natürlich kann ein Test wie dieser nicht alle Schwachstellen finden – er kann Ihnen zwar sagen, ob ein `alt`-Text bei einem Bild vorhanden ist, nicht aber, ob sein Inhalt für Menschen Sinn macht. Bei solchen Fragen hilft Ihnen eine manuelle Analyse.

9.2.2 Websites ohne CSS und Bilder analysieren

Ein praktisches Werkzeug ist die Erweiterung »Web Developer Toolbar« von Chris Pederick (*https://chrispederick.com/work/web-developer/*). Nach der Installation können Sie damit Bilder und CSS mit einem Klick deaktivieren. Prüfen Sie nun, ob die Website noch sinnvoll nutzbar ist. Stimmt die Reihenfolge der Elemente? Geht nichts Wesentliches an Inhalt verloren? Zudem können Sie die Sprachausgabe Ihres Betriebssystems aktivieren und evaluieren, ob der Inhalt verständlich bleibt.

Fortwährende Tests
Usability, User Experience und Accessibility sind fortwährende Themen für jede Website: Sie müssen immer wieder daran feilen. Am besten nehmen Sie sich regelmäßig einige Minuten Zeit, um Ihre Website auf mögliche Schwachstellen hin zu überprüfen – das gilt insbesondere, wenn sich regelmäßig etwas an der Struktur verändert.

9.2.3 Analytics

Ein weiterer Tipp: Wenn Sie eine Analyse-Software wie beispielsweise Google Analytics (*www.google.com/analytics)* verwenden, können Sie sehr genau erkennen, welche Bereiche Ihrer Website für die Nutzerinnen und Nutzer interessant sind – und daran lassen sich oft Problemstellen finden, beispielsweise wenn in einem Shop nach dem Aufruf des Warenkorbs nicht gekauft, sondern abgebrochen wurde.

9.2.4 Testen mit Nutzerinnen und Nutzern

Der wichtigste Faktor bei der Optimierung von Usability, User Experience und Accessibility bleibt natürlich der Mensch selbst. Und so verwundert es nicht, dass menschzentrierte Gestaltung ohne Feedback von Nutzerinnen und Nutzern nicht möglich ist.

Mit Usability-, User-Experience- und Accessibility-Tests können Sie Erkenntnisse über das Verhalten Ihrer Nutzerinnen und Nutzer auf der Website gewinnen – so etwas lässt sich auch ohne Probleme selbst durchführen. Da dieses Thema problemlos ein ganzes Buch füllen könnte, möchten wir uns hier auf ein typisches Vorgehen konzentrieren und zur Vertiefung auf weiterführende

Literatur verweisen, etwa das »Praxisbuch Usability und UX« von Jens Jacobsen und Lorena Meyer. Wir gehen bei diesem Beispiel davon aus, dass Sie Ihre Ideen so oft wie möglich testen und die Ergebnisse bei der weiteren Arbeit berücksichtigen (sogenannte »formative Tests«).

Sie können zum Testen eine fertige Website verwenden, aber auch die vielen verschiedenen Prototypen, die Sie in Kapitel 3 kennengelernt haben. Tatsächlich sollten Sie so früh und so oft wie möglich testen – jeder Test erlaubt es Ihnen, Ihr Angebot möglichst gut auf die Bedürfnisse der Zielgruppe auszurichten. Achten Sie darauf, Personen zu rekrutieren, die auch wirklich zu Ihrer Zielgruppe gehören. Sie können sich dazu auch an einen Ort begeben, wo Sie diese Menschen treffen können. Wenn Sie beispielsweise junge Menschen ansprechen möchten, ist ein Café in der Nähe einer Universität empfehlenswert. Je nachdem, wie lange Ihr Test dauert, sollten Sie eine kleine Aufmerksamkeit als Entschädigung bereithalten.

Definieren Sie vorab eine Reihe von typischen Aufgaben, die während des Tests gelöst werden sollen. Außerdem hilft es, ein Szenario zu formulieren, bei dem Sie die Situation beschreiben, in die sich die Teilnehmenden hineinversetzen sollen (z. B. »Sie arbeiten in der Personalabteilung eines Unternehmens. Eines Tages …«). Achten Sie bei den Szenarien und Aufgaben auf verständliche Formulierungen, und vermeiden Sie, Anhaltspunkte zu geben. So würde eine Formulierung wie »Suchen Sie nach einem Doppelzimmer« nahelegen, die Suchfunktion zu verwenden – besser ist eine neutrale Formulierung wie »Sie benötigen ein Doppelzimmer«. Im Idealfall sollten die Aufgaben unabhängig voneinander sein.

Eine typische Testsession läuft wie folgt ab:

- Begrüßen Sie die Teilnehmenden, bauen Sie durch Small Talk ein Vertrauensverhältnis auf, und erläutern Sie ausführlich, welche Daten Sie sammeln möchten. Lassen Sie Zeit für Fragen, und holen Sie sich eine explizite, schriftliche Einverständniserklärung ein. Verdeutlichen Sie, dass Sie die Website testen, nicht die Person.
- Wenn Sie möchten, können Sie in einem Gespräch vorab einige Fragen zum Thema der Website stellen.

Wie viele Tests?
Eine der zentralen und viel diskutierten Fragen ist, wie viele Tests Sie eigentlich durchführen sollten. Wie so oft gibt es hier keine allgemein gültige Antwort – mehr zum Thema lesen Sie unter *www.nngroup.com/articles/how-many-test-users/*. Für die formativen Tests, die wir hier behandeln, genügen oft schon fünf bis acht Personen, wenn Sie aus den Ergebnissen lernen und Ihre Verbesserungsideen erneut testen.

Tipp: Interview-Leitfaden und Beobachtungsbögen
Meist hilft es, im Vorfeld einen Interview-Leitfaden zu formulieren, in dem Sie die Informationen und Fragen ausformulieren, die Sie berücksichtigen möchten. Das hilft dabei, nichts zu vergessen.

Auch ist es empfehlenswert, sich einen Beobachtungsbogen zusammenzustellen, auf dem Sie Ihre Notizen festhalten können. Ein Beispiel finden Sie im Download-Bereich bei den Code-Beispielen in Kapitel_09.

- Während der Tests lohnt es sich, wenn die Teilnehmenden verbalisieren, was in ihren Köpfen vorgeht (lautes Denken). Wenn Sie möchten und die Teilnehmenden einverstanden sind, können Sie die Testsessions per Screen- und Audio-Recording aufzeichnen. Auf jeden Fall jedoch sollten Sie sich Notizen machen, beispielsweise über einen Beobachtungsbogen.
- Geben Sie den Teilnehmenden jede Aufgabe auf einer eigenen Seite. Sie sollten selbst entscheiden, wann sie glauben, eine Aufgabe erfüllt zu haben, und dann zur nächsten Aufgabe übergehen.
- Ihre eigene Rolle sollte während der Moderation sehr neutral sein, um das Ergebnis nicht zu beeinflussen. Wenn Ihre Teilnehmenden Fragen stellen, notieren Sie sich diese, und greifen Sie sie später wieder auf. Wenn Sie Rückfragen haben, stellen Sie neutrale Fragen wie »Was geht Ihnen gerade durch den Kopf?«.
- Im Anschluss sollten Sie den Test in einem abschließenden Gespräch erneut durchgehen. Stellen Sie offene Fragen wie »Wie haben Sie Aufgabe 1 erlebt?«, und achten Sie darauf, Ihre Beobachtungen systematisch mit den Teilnehmenden durchzugehen. Selbstverständlich sollten Sie auch Raum für Rückfragen von Seiten der Teilnehmenden geben.

Verhalten während der Moderation
Im »Moderator's Survival Guide« finden Sie viele ausgezeichnete Videos, wie Sie sich bei der Moderation *nicht* verhalten sollten: *https://vimeo.com/user21642263*.

Nach den Tests geht es an die Auswertung. Achten Sie bei den qualitativen Aussagen auf typische Muster, die immer wieder aufgetreten sind – ganz ähnlich, wie Sie es auch in der Nutzerforschung getan haben (Kapitel 3). Priorisieren Sie Ihre Beobachtungen, etwa in kosmetische, leichte, schwere und gravierende Probleme. Hilfreich sind auch Kennzahlen dazu, wie viele Teilnehmende die einzelnen Aufgaben lösen konnten. Verwenden Sie diese Ergebnisse, um Ihre Gestaltung in der nächsten Iteration zu verbessern.

9.2.5 Heuristische Evaluation und Cognitive Walkthroughs

Neben solchen Tests gibt es natürlich auch die Möglichkeit, eine Website mit kritischem Blick durchzugehen und selbst auf häufige Fehler zu achten (je nach Ausrichtung genannt *Usability Review*, *UX Review* oder *Accessibility Review*). Man bezeichnet dies auch als

Expert Review, weil Expertinnen und Experten ihre Einschätzung abgeben. Bei allen diesen Verfahren gilt, dass sie Tests mit Nutzerinnen und Nutzern ergänzen, aber niemals ersetzen können.

Eine Möglichkeit ist die *heuristische Evaluation*, die sich an etablierten Prinzipien von Usability und Accessibility (den *Heuristiken*) orientiert. Es gibt sehr viele unterschiedliche Heuristiken, und die Auswahl gehört zur Konzeption einer heuristischen Evaluation dazu. Zu den bekanntesten gehören die zehn Usability-Heuristiken von Jakob Nielsen und Rolf Molich (*www.nngroup.com/articles/ten-usability-heuristics/*):

- **Sichtbarkeit des Status**: Eine Gestaltung sollte jederzeit informieren, was gerade vorgeht (Feedback).
- **Übereinstimmung von System und Wirklichkeit**: Das Design sollte die Sprache der Nutzerinnen und Nutzer sprechen. Sofern möglich, sollten Konventionen aus der Wirklichkeit genutzt werden, so dass Informationen natürlich und logisch erscheinen.
- **Kontrolle und Freiheit**: Nutzerinnen und Nutzer brauchen »Notausgänge« aus allen Vorgängen (etwa Undo und Redo).
- **Beständigkeit und Standards**: Das gesamte Design sollte konsistent sein, etwa in Bezug auf Formulierungen, Handlungen und Situationen. Konventionen sollten befolgt werden.
- **Fehlervermeidung**: Fehler sollten vermieden werden.
- **Wiedererkennung statt Erinnerung**: Die kognitive Last von Menschen sollte verringert werden. Statt Informationen behalten zu müssen, sollten sie sichtbar sein, wenn sie gebraucht werden.
- **Flexibilität und Effizienz**: Abkürzungen und individualisierbare Funktionen können fortgeschritteneren Anwenderinnen und Anwendern helfen, effizient mit einer Gestaltung zu interagieren.
- **Ästhetisches und minimalistisches Design**: Unnötige Informationen und Dekorationen sollten vermieden werden.
- **Hilfestellung beim Erkennen, Bewerten und Beheben von Fehlern**: Beim Auftreten von Fehlern sollte eine Gestaltung hilfreiche Fehlermeldungen in deutlicher Sprache verwenden.
- **Hilfe und Dokumentation**: Ein System sollte Hilfestellungen bieten, wenn Nutzende sie benötigen.

Anzahl von Expertinnen und Experten im Team
Die Fragen, wie viele Personen bei einer heuristischen Evaluation mitmachen sollten und wie gut sie Probleme finden, sind sehr umfangreich erforscht worden. Meist werden drei bis fünf Personen eingesetzt. Eine lesenswerte Einführung finden Sie unter *www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/*.

Bei der Durchführung ist es von zentraler Bedeutung, die Meinungen mehrerer Expertinnen und Experten zu kombinieren und genau auf deren Erfahrung und Wissen zu achten. Bei komplexen Thematiken empfiehlt es sich, auch Personen mit dem entsprechenden Fachwissen in das Review-Team hinzuzuziehen, z. B. aus dem Produktmanagement. Eher nicht geeignet ist das Verfahren, wenn Sie es einzeln oder ohne das entsprechende Hintergrundwissen durchführen möchten – dann wäre das Risiko hoch, dass Sie viele Probleme nicht finden.

Die Expertinnen und Experten gehen die Website auf Basis der gewählten Heuristiken zunächst individuell durch und dokumentieren ihre Beobachtungen. Danach diskutieren sie die Ergebnisse im Team, priorisieren sie (meist mit vier Stufen) und sammeln Lösungsvorschläge.

Ein weiteres Verfahren, bei dem Expertinnen und Experten einen kritischen Blick auf eine Website werfen, ist der *Cognitive Walkthrough*. Dabei werden aber keine Richtlinien herangezogen, sondern konkrete Aufgaben durchgegangen, bei denen sich die Expertinnen und Experten in die Köpfe der Zielgruppe hineinversetzen.

9.3 Performance: Lade- und Renderingzeiten im Griff

Auswirkungen guter Performance
Unter *https://wpostats.com* finden Sie zahlreiche Statistiken zur Bedeutung guter Website-Performance. Im Onlineshopping erwarten wir beispielsweise, dass eine Seite in zwei Sekunden geladen ist – bereits nach drei Sekunden brechen viele Menschen den Vorgang ab.

Zu guter Usability und Accessibility gehören auch möglichst kurze Lade- und Renderingzeiten. Menschen sind im Internet eher ungeduldig und haben wenig Verständnis für langsame Websites. Tatsächlich hängt der Erfolg von Websites stark mit ihrer Performance zusammen. Swappie, ein Anbieter für gebrauchte, instandgesetzte iPhones, konnte durch Performance-Optimierung beispielsweise den Umsatz bei der mobilen Nutzung um 42 % steigern (*https://web.dev/swappie/*). Performance kann sich also im wahrsten Sinne des Wortes lohnen und ist ein fundamentaler Bestandteil der User Experience.

Performance besteht aus objektiver Performance, die Sie mit Hilfe von Kennzahlen messen können, sowie subjektiver Performance, also wie schnell sich eine Website *anfühlt*. Letztere ist naturgemäß schwieriger zu messen, allerdings gibt es Empfehlun-

gen, wie Sie von Kennzahlen darauf schließen können, wie Menschen die Performance einer Website wahrnehmen. Zum Glück gibt es Techniken, die Sie zur Verbesserung der Performance einsetzen können. Wir werden in den folgenden Abschnitten viele dieser Techniken versammeln, empfehlen Ihnen aber auch, die Wahrnehmung von Performance in Tests zu thematisieren. Allgemein gesprochen sollten Sie Nutzenden ein Feedback geben, wann immer etwas geschieht, das eine gewisse Zeit dauert. Sinnvoll kann es auch sein, das Publikum aktiv zu involvieren, damit die Wartezeit nicht mehr so negativ auffällt. Das geht beispielsweise über eine Animation, wie Sie es in Kapitel 6 gelernt haben.

9.3.1 Performance als Designentscheidung

Um der immer weiter steigenden Bedeutung von Performance Rechnung zu tragen, hat der Webentwickler Tim Kadlec die Idee des Performance-Budgets vorgeschlagen. Dabei geht es darum, performancerelevante Messwerte möglichst früh im Projekt zu definieren, denn jede Designentscheidung hat Auswirkungen auf die Performance. Damit vermeiden Sie, Performance nur ganz am Ende eines Projekts in den Blick zu nehmen, wenn viele Entscheidungen bereits getroffen wurden. In der Praxis kann das so aussehen, dass Sie einen festen Richtwert festlegen, der nicht überschritten werden darf.

Es gibt verschiedene Möglichkeiten, ein Performance-Budget zu definieren:

- **Basierend auf quantitativen Größen**: »Diese Seite soll eine Größe von 400 kB nicht überschreiten und maximal 15 Requests ausführen.«
- **Basierend auf der User Experience**: »Unsere Seite soll eine Ladezeit von unter 1,5 Sekunden (bei DSL 16000) haben.«
- **Basierend auf dem Wettbewerb**: »Unsere Seite soll 20 % schneller laden als die unserer Mitbewerber.«

Sinnvoll ist es zudem, das Performance-Budget in kleinere Bausteine zu unterteilen. So könnten Sie beispielsweise festlegen, dass alle eingesetzten Webfonts zusammen maximal 100 kB groß sein dürfen. Und beim Setzen eines Performance-Budgets lohnt ein Blick in den Chrome UX Report (*https://developer.chrome.com/*

Quellen und Lesetipps zu Performance

Zur Vertiefung empfehlen wir die folgenden Quellen:

- Performance-Checkliste (Vitaly Friedman): *www.smashingmagazine.com/2021/01/front-end-performance-2021-free-pdf-checklist/*
- Web Performance (MDN): *https://developer.mozilla.org/en-US/docs/Web/Performance*
- The Impact of Web Performance: *https://simplified.dev/performance/impact-of-web-performance*

Automatisieren
Wenn Sie die Auslastung Ihres Performance-Budgets nicht immer wieder selbst nachrechnen möchten, hat Tim Kadlec unter *https://github.com/tkadlec/grunt-perfbudget* ein Plug-in geschrieben, um den Prozess mit Hilfe des Tools Grunt zu automatisieren.

docs/crux/). Dabei handelt es sich um ein großes Datenset verschiedener Performance-Kennzahlen, das auf monatlicher Basis veröffentlicht wird.

Der Vorteil eines Performance-Budgets liegt darin, dass Sie die Auswirkungen von Design- und Content-Entscheidungen auf die Performance möglichst früh diskutieren können. Soll auf einer Seite ein neues Feature eingebaut werden (z. B. ein weiteres Carousel), das die festgelegte Grenze überschreitet, muss optimiert werden oder ein anderes Feature weichen. Damit wirken Sie dem Phänomen des »Feature Creep« entgegen: Im Laufe der Zeit werden immer neue Funktionen eingeführt – nach und nach geht damit die Performance in den Keller.

9.3.2 Speed-Tests und Dev-Tools nutzen

Ein Performance-Budget hat also handfeste Vorteile, lebt aber davon, dass Sie die Performance auch messen können. Google bietet mit Page Speed ein beliebtes Werkzeug zur Messung an (*https://developers.google.com/speed/pagespeed/insights*). Es gibt Ihnen neben Anhaltspunkten in Form von Noten auch ganz konkrete Empfehlungen.

Empfehlenswert ist außerdem der »Service Web Page Test« (*www.webpagetest.org*), der die effektive Ladezeit einer Website misst und mit einem Diagramm die Ladereihenfolge und -dauer verschiedener Ressourcen darstellt. Einen Blick wert ist auch »Gift of Speed« (*www.giftofspeed.com*).

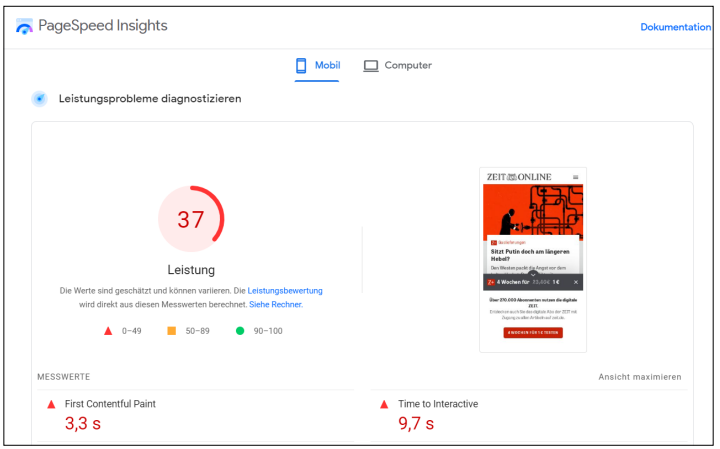
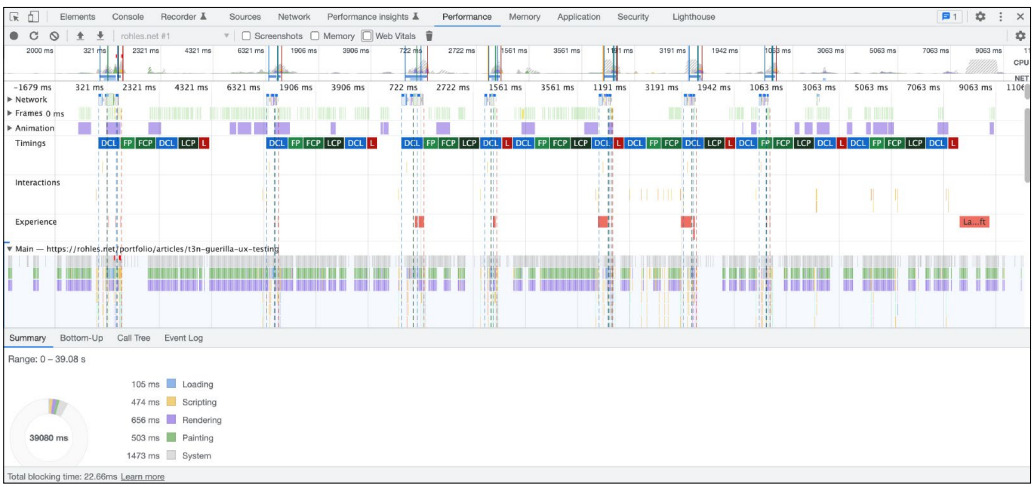


Abbildung 9.6 ▸
Google Page Speed

Natürlich haben auch die Entwicklungswerkzeuge der Browser Tools zur Messung der Performance an Bord. Abbildung 9.7 zeigt ein Beispiel, wie Google Chrome die Performance bei der Nutzung einer Website aufzeichnet. Für die Praxis lohnt sich ein Blick in die Dokumentation Ihres Entwicklungsbrowsers (für Chrome etwa: *https://developer.chrome.com/docs/devtools/evaluate-performance/*), in der alle Kennzahlen und Tools erläutert werden.

▼ **Abbildung 9.7**
Beispiel aus dem Performance-Tab in den Entwicklungswerkzeugen von Google Chrome



9.3.3 Performance-Kennzahlen auswählen und verstehen

Es gibt sehr viele Kennzahlen, die Sie zur Performance-Messung heranziehen sollten. Denn auch wenn der Wunsch verständlich ist, Performance in einer einzigen Zahl ausdrücken zu wollen – es ist nicht möglich, weil es viele verschiedene Faktoren zu berücksichtigen gibt. Unser Ziel ist es daher, Ihnen einige Empfehlungen als Orientierung mitzugeben, die Ihnen ermöglichen sollen, die richtigen Kennzahlen für Ihren eigenen Anwendungsfall auszuwählen.

Kleiner Wegweiser durch den Metrik-Dschungel | In seiner großartigen (und umfangreichen) Performance-Checkliste unterscheidet Vitaly Friedman zwischen vier Arten von Metriken:

- **Quantitative Metriken** (z. B. Zahl der Requests) sind zwar einfach zu messen und können gut als Alarmsignale dienen, lassen aber nicht so gut auf die User Experience zurückschließen.

Rendering?
Der Begriff *Rendering* bezeichnet den Prozess, mit dem ein Browser eine Website darstellt. Wir werden in Abschnitt 9.3.7 ausführlich darauf eingehen.

- **Meilenstein-Metriken** schauen den Ladeprozess einer Website im Detail an. Ein Beispiel ist »Time To Interactive« (TTI). Dieser Begriff beschreibt den Zeitpunkt, ab dem Nutzerinnen und Nutzer mit einer Website interagieren können.
- **Rendering-Metriken** (z. B. »Largest Contentful Paint«, LCP, siehe direkt im Anschluss) nehmen das Rendering von Websites in den Blick. Sie eignen sich zur allgemeinen Performance-Optimierung (»Ist ein Content schnell geladen?«), erfassen aber nicht, ab wann sich die Website interaktiv anfühlt.
- Hin und wieder kommt es vor, dass Websites **eigene Metriken** definieren. Dazu lohnt ein Blick auf die Performance API (*https://developer.mozilla.org/en-US/docs/Web/API/Performance_API*).

Für eine sinnvolle Performance-Optimierung empfiehlt es sich, Metriken verschiedener Art zu kombinieren. Zur Orientierung möchten wir Ihnen zwei Varianten mitgeben: die Core Web Vitals und RAIL.

Mehr zu Core Web Vitals
Der Entwickler Simon Hearne hat unter *https://simonhearne.com/2020/core-web-vitals/* einen ausgezeichneten Artikel mit vielen Tipps zur Messung und Optimierung von Core Web Vitals verfasst. Und unter *https://web-vitals.pazguille.me* können Sie die Core Web Vitals einer Website mit denen von Wettbewerbern vergleichen.

Core Web Vitals | Hinter dem Begriff »Core Web Vitals« verbergen sich eine Reihe von Kennzahlen und Richtlinien, die Google im Frühjahr 2020 angekündigt hat. Mittlerweile sind sie auch als Ranking-Kriterien in den Suchalgorithmus gewandert – daher finden Sie die Auswertung auch in der Search Result Console von Google. Ziel ist es, Performance aus Sicht der User Experience zu bewerten (siehe Tabelle 9.1). Das Team hat Richtwerte definiert, wie Menschen die Kriterien empfinden (*https://web.dev/defining-core-web-vitals-thresholds/*).

Tabelle 9.1 ▸
Übersicht über die Core Web Vitals mit Richtwerten für eine »gute« Bewertung

Kriterium	Beschreibung	Richtwert
Largest Contentful Paint als Indikator der Ladezeit	Rendering-Zeit für das größte Bild oder den größten Textblock innerhalb des Viewports	unter 2,5 Sekunden
First Input Delay als Indikator der Interaktivität	Zeitpunkt, ab dem der Browser auf Eingaben von Nutzerinnen und Nutzern reagieren kann	unter 100 Millisekunden

Kriterium	Beschreibung	Richtwert
Cumulative Layout Shift als Indikator der visuellen Stabilität	Sprunghafte Veränderungen im Layout während des Ladens (errechnet aus Größe und Entfernung, siehe *https://web.dev/cls/*)	unter 0,1

◀ **Tabelle 9.1**
Übersicht über die Core Web Vitals mit Richtwerten für eine »gute« Bewertung (Forts.)

RAIL | Das RAIL-Modell (*https://web.dev/rail/*) ist ein anderes Verfahren, Performance-Kennzahlen aus Sicht der Nutzerinnen und Nutzer zu interpretieren. RAIL steht für die vier Aspekte Response, Animation, Idle und Load. Ziel ist es, dass sich Websites und Web-Apps performant anfühlen. Tabelle 9.2 enthält Definitionen und Richtwerte.

Kriterium	Beschreibung	Richtwert
Response	Reaktionen auf Handlungen der Nutzerinnen und Nutzer, die sich unverzögert anfühlen sollten (z. B. Klicks auf Buttons)	Handlungen innerhalb von **100 ms** durchführen (Tipp: **50 ms** nutzen, da der Browser noch andere Dinge ausführen muss); sonst Feedback geben
Animation	visuelle Darstellung von Übergängen zwischen Zuständen	jedes Bild (Frame) der Animation in **10 ms** erzeugen
Idle	Zeit, in der Anwenderinnen und Anwender nicht mit dem Browser interagieren (»der Browser hat also nichts zu tun«)	Zeit mit weniger wichtigen Hintergrundaktivitäten nutzen (Beispiel: zuerst direkt sichtbare Inhalte laden, dann den Rest nachladen)
Load	Laden und Aufbau von Webseiten, sollte auch bei 3G-Verbindungen annehmbar schnell sein	Inhalte in unter **5 s** laden (2 s für wiederholte Besuche)

◀ **Tabelle 9.2**
RAIL-Modell

Maßnahmen festlegen | Nachdem Sie die für Sie relevanten Performance-Kennzahlen ausgewählt und gemessen haben, sollten Sie geeignete Maßnahmen festlegen, um eine bessere Performance zu erreichen. Für schlechte Performance kann es viele Gründe geben, etwa ungenutzter Code, zu viele Anfragen nach Dateien, mangelnde Optimierung von Dateien oder unvorteilhafter Aufbau von Seiten (Rendering). In den folgenden Abschnitten werden wir auf diese Thematiken nach und nach eingehen.

9.3.4 Ungenutzten Code entfernen

Ein erster Schritt bei der Performance-Optimierung von Websites ist, unnötigen Datenballast zu entfernen. Dazu zählt, den Quelltext aufzuräumen sowie die Dateigröße selbst zu reduzieren.

Während der Entwicklung und im Livebetrieb schleicht sich immer mal wieder Code ein, der später nicht mehr genutzt wird und entfernt werden kann. Mit den Entwicklungswerkzeugen oder Tools wie »Unused CSS« (*https://unused-css.com*, kostenpflichtig) lässt sich das CSS auf ungenutzte Selektoren und Angaben hin untersuchen. In Google Chrome (siehe Abbildung 9.8) ist das entsprechende Tool (genannt »Coverage«) zunächst ausgeblendet. Mit einem Klick auf den Button 1 und Run command 2 können Sie ein Menü 3 öffnen. Dort können Sie nach »coverage« suchen und den Befehl Show Coverage ausführen. Danach findet sich das Coverage-Tool als Tab.

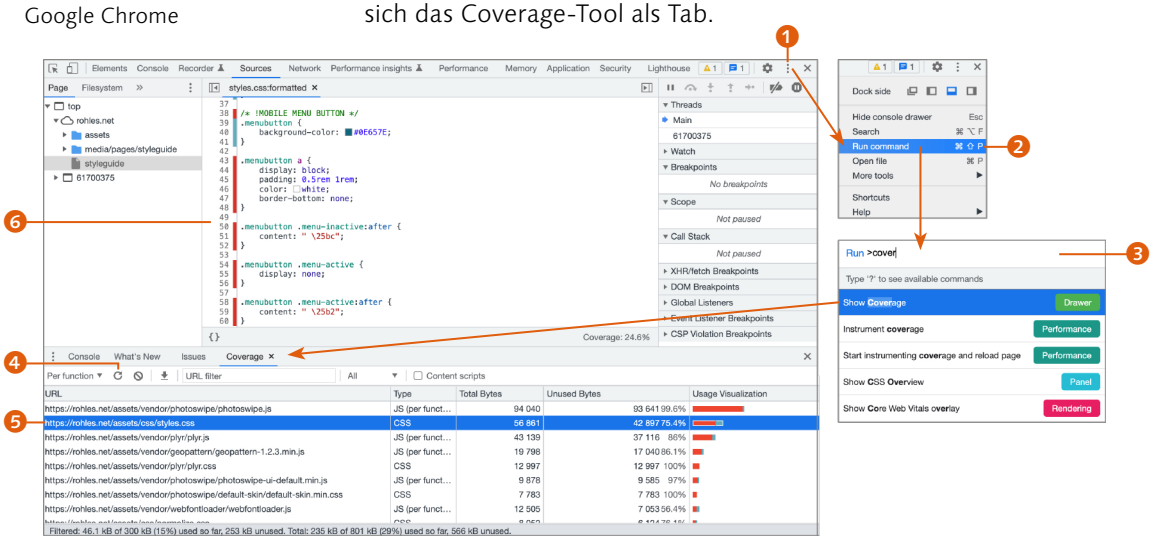


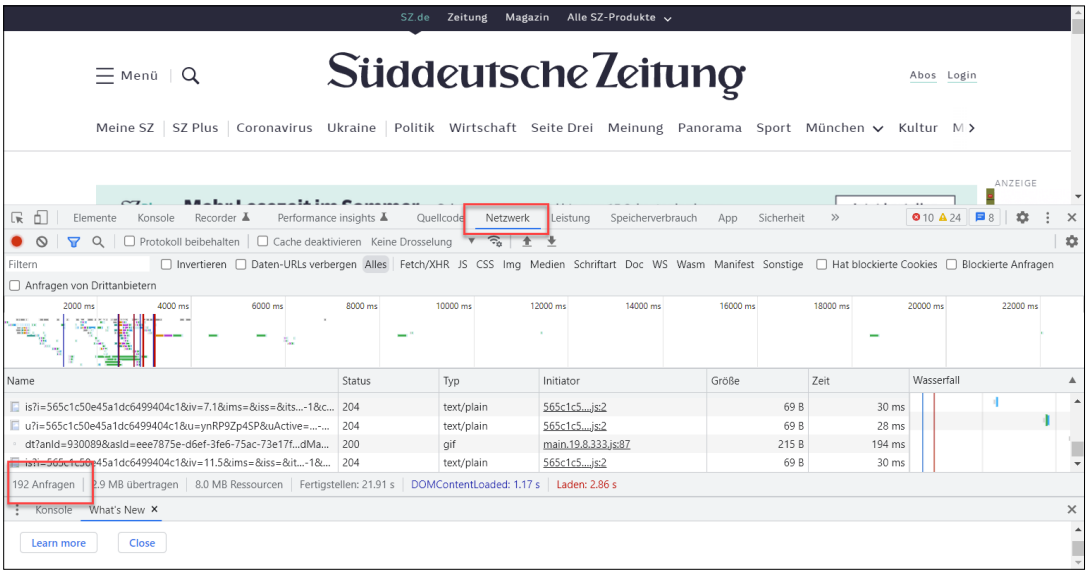
Abbildung 9.8 ▼
Coverage-Tool in den Entwicklungswerkzeugen von Google Chrome

Ein Klick auf den Button mit dem gedrehten Pfeil ❹ lädt die aktuelle Seite neu und zeichnet dabei auf, welche Code-Teile verwendet werden. Im Bereich ❺ können Sie eine Datei auswählen, die dann bei ❻ angezeigt wird. Dort sind alle Code-Teile, die bei der aktuellen Seite nicht verwendet wurden, rot markiert. Wenn Sie dies auf verschiedenen Seiten nutzen, erhalten Sie einen guten Einblick davon, welche Teile ungenutzt bleiben.

9.3.5 Server-Anfragen optimieren

Wenn eine Website mit einem Browser aufgerufen wird, fordert dieser Daten vom Server an. Eine solche Anfrage wird auch als HTTP-Request bezeichnet. HTTP steht für *Hypertext Transfer Protocol* und beschreibt den Vorgang von der Anfrage einer Datei vom Browser bis zum Zurücksenden dieser Datei vom Server.

Nun ist es nicht so, dass eine Webseite in einem Rutsch vom Server an den Browser gesendet wird. Dies geschieht Datei für Datei. Und für jede Datei wiederum wird ein HTTP-Request nötig. Ergo, je mehr Dateien sich auf einer Seite befinden, desto mehr HTTP-Requests werden abgesetzt und umso länger dauert es, bis die Webseite komplett geladen ist.



▼ Abbildung 9.9
Ein Blick in die Entwicklertools von Google Chrome im Bereich NETZWERK zeigt links unten die Anzahl der HTTP-Requests (hier sind es 192 Anfragen bei *www.sueddeutsche.de*).

Tools und Lesetipps zu Protokollen

- Unterstützung testen: *https://tools.keycdn.com/http2-test* (HTTP/2) und *https://http3check.net/* (HTTP/3)
- Kostenlose E-Books mit Erläuterungen (Daniel Stenberg): *https://daniel.haxx.se/http2/* (HTTP/2) und *https://http3-explained.haxx.se/de* (HTTP/3)
- Sehr gute Erläuterung von HTTP/3: *https://blog.cloudflare.com/http3-the-past-present-and-future/*

Optimierungen beim Hosting

Erwähnenswert ist auch, dass es auch Performance-Optimierungen gibt, die Sie nur in Absprache mit Ihrem Hosting-Provider vornehmen können. Websites in günstigen Hosting-Tarifen sind meist weniger performant, weil Sie sich die Rechenleistung des Servers mit anderen Kundinnen und Kunden teilen (sogenanntes »Shared Hosting«).

Eine Frage des Protokolls | Es gibt verschiedene Versionen von HTTP, und hier liegt ein großes Optimierungspotenzial. HTTP/1.1 ist seit 1997 im Einsatz, hat allerdings seine Tücken. Der im Mai 2015 verabschiedete Nachfolger HTTP/2 möchte damit aufräumen. Besonders wichtig ist die Multiplex-Übertragung, bei der mehrere Dateien mit einer Verbindung übertragen werden, statt jedes Mal eine neue Verbindung aufzubauen. Konkret heißt das: Während Sie bei HTTP/1.1 versuchen sollten, möglichst *wenige* Requests auszuführen, profitiert HTTP/2 in gewissem Umfang von parallelen Übertragungen. Vitaly Friedman empfiehlt, etwa sechs bis zehn Pakete gleichzeitig anzuvisieren. HTTP/2 erfreut sich breiter Unterstützung – mittlerweile laufen über 60 % der Verbindungen über HTTP/2 (*https://almanac.httparchive.org/en/2020/http*).

Wichtig zu wissen ist, dass Browser HTTP/2 nur über verschlüsselte Verbindungen unterstützen. Das bedeutet, dass die Verbindung zwischen Client und Server mit Hilfe eines Schlüssels automatisch so konfiguriert wird, dass die übertragenen Daten nicht mitgelesen werden können. Technisch spricht man daher von »Transport Layer Security« (TLS) – im Gebrauch ist aber auch noch die ältere Bezeichnung »Secure Sockets Layer« (SSL). Dank Initiativen wie Let's Encrypt (*https://letsencrypt.org*) ist Verschlüsselung bei vielen Hosting-Providern mittlerweile kostenfrei erhältlich. Verschlüsselung ist auch aus anderen Gründen sinnvoll (etwa durch Vorteile im Suchmaschinen-Ranking, höheres Vertrauen auf Seiten der Menschen oder Schutz beim Verschicken von Formularen) und sollte daher aktiviert werden.

Mit HTTP/3 steht ein weiterer Nachfolger bereits in den Startlöchern. Der wichtigste Grund für die Weiterentwicklung liegt in der bereits erwähnten Multiplex-Übertragung von HTTP/2. Bei der Übertragung werden die Daten in kleine Pakete aufgeteilt, die über das Netzwerk verschickt werden. Bei HTTP/2 (und HTTP/1.1) kommt dabei das sogenannte »Transmission Control Protocol« (TCP) zum Einsatz. HTTP/2 erlaubt es, sehr viele Übertragungen gleichzeitig über eine einzige TCP-Verbindung zu machen. Problematisch ist allerdings, dass immer mal wieder ein Paket in einer dieser Übertragungen verloren gehen kann und neu übertragen werden muss. Weil TCP nichts von den parallel laufenden Übertragungen weiß, müssen bei HTTP/2 dann *alle* Übertragungen warten, bis das fehlende Paket angekommen ist. Zur

Lösung ersetzt HTTP/3 TCP mit dem neuen Übertragungsprotokoll QUIC, bei dem die einzelnen Übertragungen unabhängig voneinander laufen.

Zur Umstellung auf HTTP/2 oder HTTP/3 ist es notwendig, die passende Server-Version bzw. das passende Server-Modul zu installieren und zu konfigurieren. Sprechen Sie dazu mit Ihrem Hosting-Provider.

HTTP-Requests prüfen und optimieren | Bei der Arbeit probieren Kreative immer mal wieder etwas aus, das sie später doch nicht benötigen, etwa eine Bibliothek hier oder ein Plug-in dort. Schnell vergisst man dann, diese Dateien wieder zu entfernen. Es lohnt sich daher, alle Requests durchzugehen:

- **Entfernen Sie unnötige Dateien**: Oftmals werden sehr viele Bilder auf einer Website verwendet, von denen nicht alle einen großen Nutzen haben. Und auch andere Dateien sammeln sich schnell an. Prüfen Sie daher alle Requests kritisch, und entfernen Sie alles, was nicht unbedingt benötigt wird.
- **CSS-Dateien und Skripte zusammenfassen**: Für jede CSS-Datei wird ein HTTP-Request nötig und die Ladezeit verlangsamt. Hier könnten Sie optimieren, indem Sie verschiedene CSS-Dateien zu einer zusammenfassen. Dasselbe gilt für Skriptdateien.
- **Prüfen Sie andere Elemente auf der Seite**: Oftmals verlangsamen Social-Media-Integrationen zu Facebook, Twitter oder Videos von externen Quellen die Ladezeit einer Website erheblich. In dem Fall sollten Sie das Element nochmals überdenken.
- **Beschleunigen Sie die Verteilung von Daten mit einem Content-Delivery-Network (CDN)**: CDNs wie beispielsweise Akamai (*www.akamai.com*) oder Cloudflare (*www.cloudflare.com*) sind kostenpflichtige Services, die ein Netzwerk von regional verteilten Servern betreiben. Beim Besuch einer Website können sie die Anfragen an den Server weiterleiten, der die beste Performance verspricht – beispielsweise, weil er physisch näher bei den Clients ist. Das lohnt sich vor allem für kommerzielle Websites mit einem großen, verteilten Publikum.

Requests mit Data-URIs reduzieren | Eine weitere Möglichkeit zum Einsparen von HTTP-Requests ist die Arbeit mit Data-URIs. Das lässt sich beispielsweise bei Grafiken oder Webfonts nut-

Ressourcen priorisieren
Mit Hilfe des `rel`-Attributs von `link`- und anderen HTML-Elementen können Sie beeinflussen, welche Ressourcen Browser zuerst laden sollen. Mehr zu diesem Thema lesen Sie in einem sehr guten Beitrag von Brian Jackson: *www.keycdn.com/blog/resource-hints*.

CDNs verstehen und einsetzen
Katie Hempenius hat unter *https://web.dev/content-delivery-networks/* einen hervorragenden Beitrag geschrieben, wie CDNs funktionieren und wie man sich für eines entscheidet.

zen. Dabei werden die Grafiken nicht im CSS oder HTML referenziert, sondern direkt dort eingebettet – und zwar in einer Base64-codierten Form. Diese Zeichenketten sind lang und unübersichtlich:

Listing 9.4 ▸
Data-URI als `src`

```
<img width="100" height="200" alt="Flugzeug beim Start" src=
"
AAAePET[…]" />
```

Die Syntax einer Data-URI startet stets mit `data:`. Es folgen Angaben zu Dateityp und Codierung, schließlich ein Komma und die Rohdaten des Bildes. Bei der Konvertierung helfen die Onlinetools unter *http://websemantics.co.uk/online_tools/image_to_data_uri_convertor* oder *https://boazsender.github.io/datauri*.

Data-URIs geben Ihrer Website einen Performance-Boost, haben jedoch auch eine Reihe von Nachteilen:

- Die Arbeit mit ihnen ist unübersichtlich, da im Quelltext sehr lange Zeichenketten auftauchen.
- Außerdem ist es aufwendiger, ein Bild auszutauschen – statt es per FTP einfach zu überschreiben, muss es neu codiert und an allen referenzierten Stellen ausgetauscht werden.
- Data-URIs sparen Requests, sind aber auch oft etwas größer als die Ursprungsdateien. Hier hilft Komprimierung mit gzip.
- Ihre Verwendung sollte mit gutem Caching verbunden werden: Wenn Data-URIs beispielsweise im CSS eingebunden sind, vergrößert sich dadurch das Dokument und dessen Ladezeit – das CSS wird daher erst später ausgewertet. Hier hilft es also, das CSS für die spätere Verwendung großzügig zu cachen.

9.3.6 Dateigröße optimieren

Datenkosten
Auf *https://whatdoesmysitecost.com* erhalten Sie eine Einschätzung, was der Aufruf einer Website in verschiedenen Ländern der Welt kostet (in $).

Nachdem Sie den Code aufgeräumt und die Zahl der Requests reduziert haben, geht es nun darum, die Dateigröße einer Website so klein wie möglich zu machen. Bei Schriften und Bildern haben wir dieses Thema bereits in den vorherigen Kapiteln besprochen, aber es gibt noch weitere Möglichkeiten.

Daten komprimieren | Mit dem Komprimierungsverfahren gzip steht eine Technologie zur Verfügung, mit der im Schnitt 70 % der Daten eingespart werden können. Das Verfahren greift besonders

bei Texten und spart einiges an Zeichen, indem auf bereits vorhandene Informationen verwiesen wird, statt sie erneut zu übertragen. Gzip ist ein verlustfreies Verfahren und wird von allen modernen Browsern verstanden.

Der Aufwand ist erfreulich gering: Ob gzip auf Ihrem Server funktioniert, können Sie mit *www.giptest.com* ausprobieren. Wenn Ihre Website auf einem Apache-Server läuft (das ist meistens der Fall), lässt es sich sehr einfach mit einer Anpassung der ».htaccess«-Datei aktivieren – dabei handelt es sich um eine Konfigurationsdatei des Servers. Es gibt zwei Module (*mod_gzip* und *mod_deflate*), die sich dazu eignen – beide hat Patrick Sexton unter *https://varvy.com/pagespeed/enable-compression.html* erläutert. Der Artikel enthält außerdem die Anweisungen für Server mit nginx und Litespeed. Im Zweifel hilft Ihnen der Support Ihres Hosting-Anbieters sicher gerne weiter.

Eine noch bessere Kompression als gzip erreicht der Brotli-Algorithmus, der von Google entwickelt wurde und unter einer freien MIT-Lizenz steht. Brotli erfreut sich heute einer breiten Unterstützung von Seiten der Browser. Auf der Website *www.brotli.pro* finden Sie Anleitungen, wie Sie Brotli auf verschiedenen Servern einrichten können.

CSS- und Skriptdateien minimieren | CSS- und Skriptdateien können ebenfalls minimiert werden. Dabei werden Leerzeichen, Absätze und Kommentare entfernt – das Ergebnis ist eine spürbare Reduzierung der Dateigröße. In jedem Fall sollten Sie jedoch eine nicht minimierte Kopie der Dateien aufheben, denn die minimierten Versionen sind für Überarbeitungen zu unübersichtlich.

Manuell ist dieser Prozess natürlich ein sehr hoher Aufwand. Zum Glück gibt es jedoch Helfer wie den Minifier (*www.minifier.org*) oder CSS Nano (*https://cssnano.co*), die diese Aufgabe übernehmen können.

Caching | Trotz aller Optimierungsmaßnahmen ist es immer noch mühsam, Dateien immer wieder neu laden zu müssen, wenn sie mehrmals benötigt werden. So werden beim Aufruf der Homepage beispielsweise viele Dateien geladen, die auch auf Unterseiten benötigt werden. Auch gibt es Dateien, die sich nicht verändert haben, wenn jemand einige Tage später erneut auf einer

Beispiel
Gregor Meier bringt in seinem Buch »Pagespeed Optimierung« (Hanser) folgendes einfache Beispiel: Aus dem Satz »Auch ein kleiner Beitrag ist ein Beitrag« kann durch Komprimierung der Satz »Auch ein kleiner Beitrag ist -4 -3« werden. Statt schon einmal genannte Wörter zu wiederholen, wird also nur noch auf ihre Position verwiesen – das spart selbst in diesem einfachen Beispiel schon sieben Zeichen.

Werfen Sie dazu einen Blick auf den Bonusinhalt »Weiterleitungen_htaccess« im Download-Bereich.

Website landet. In Fällen wie diesen kommt der **Browser-Cache** zum Einsatz. Hier landen Inhalte im lokalen System und werden von dort genutzt, statt sie erneut vom Server anzufragen. Moderne Browser cachen Inhalte automatisch, allerdings können Sie ein bisschen nachhelfen. Man kann dem Browser nämlich einen Richtwert mitgeben, wie lange die einzelnen Inhalte im lokalen Speicher gültig sein sollen.

Diese Richtwerte werden auf einem Apache-Server in die ».htaccess«-Datei geschrieben. Dateien, die sich selten ändern, werden mit langen Werten versehen (etwa `1 month` oder `1 year`), andere Dateitypen wie HTML werden nicht gecacht. Ein Beispiel:

Verzeichnisse mit .htaccess konfigurieren
Die ».htaccess« (Hypertext Access) ist eine Konfigurationsdatei auf den häufig verwendeten Apache-Webservern, mit der verzeichnisbezogene Regeln aufgestellt werden können.

Listing 9.5 ▸
Angaben zum Browser-Caching in der ».htaccess«

```
<IfModule mod_expires.c>
  ExpiresActive On
  ExpiresByType image/gif "access plus 1 year"
  ExpiresByType image/png "access plus 1 year"
  ExpiresByType image/jpeg "access plus 1 year"
  ExpiresByType text/css "access plus 1 year"
  ExpiresByType text/javascript "access plus 1 year"
  ExpiresByType application/javascript "access plus 1 year"
  ExpiresByType application/x-javascript "access plus 1 year"
</IfModule>
```

Server-Cache
Auch der Server selbst verfügt über einen Cache, in dem er häufig benötigte Dateien ablegen kann. Wir werden auf diese Thematik später noch eingehen.

Nach Ablauf der angegebenen Zeitdauer würde ein Browser die Datei erneut anfordern. Die Anweisungen funktionieren nur, wenn das Modul *Expires* aktiviert ist – sprechen Sie im Zweifel mit Ihrem Hosting-Provider. Nachdem Sie diese Angaben in die ».htaccess« Ihres Servers geschrieben haben, können Sie mit Google Page Speed prüfen, ob der Browser-Cache aktiv ist.

Falls Sie nach Aktivierung des Browser-Caches doch noch einmal etwas an den Dateien ändern möchten, können Sie eine Versionsnummer an den Dateinamen (*Fingerprinting*, z. B. »image_v1.png«) oder als Parameter an die URL anhängen (z. B. »image.png?v=1«). Damit weiß der Browser Ihrer Besucherinnen und Besucher, dass er die neue Version herunterladen soll.

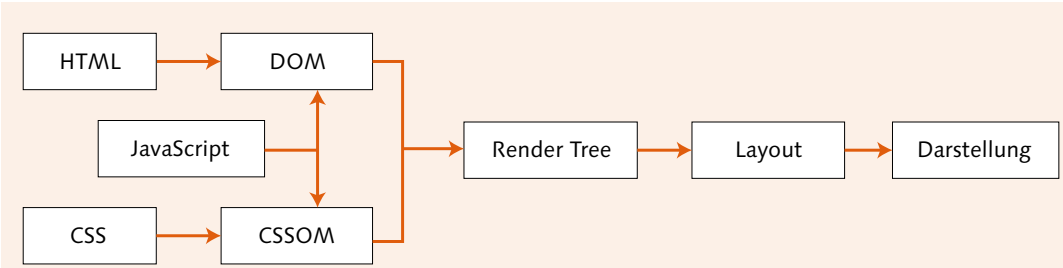
9.3.7 Webseiten so schnell wie möglich rendern

Mit performanceorientiertem Design, Reduzierung der Requests und Optimierung der Dateigröße haben Sie wichtige Schritte für

eine gute Performance unternommen. Trotz allem wird jedoch noch immer etwas heruntergeladen. Aber auch während dieser Ladezeit können Sie die User Experience optimieren, indem Sie für optimiertes Rendering sorgen.

Ein Crashkurs über das Rendering von Webseiten | Zunächst lohnt sich ein kurzer Abstecher in die Funktionsweise eines Browsers, um die folgenden Erläuterungen zu verstehen. Wie kommt ein Browser von dem Code, den Sie auf einem Server ablegen, dazu, eine fertige Webseite anzuzeigen?

▼ **Abbildung 9.10**
Rendering-Prozess des Browsers



- Beim Aufruf einer URL empfängt der Browser HTML-Dokumente.
- Daraus muss er nun etwas konstruieren, das er auf dem Bildschirm anzeigen kann. Dazu muss er den Quelltext interpretieren und in das **Document Object Model** (**DOM**) umwandeln. Der Begriff beschreibt, was der Browser aus Ihrem Quellcode herausinterpretiert – er wird beispielsweise bei einem Fehler versuchen, den Quelltext sinnvoll zu ergänzen. Das DOM enthält den Inhalt. Je einfacher Ihr Quelltext ist, desto schneller ist der Browser mit dem Interpretieren.
- Noch kann der Browser den Inhalt jedoch nicht auf dem Bildschirm anzeigen – er weiß ja noch gar nicht, wie er aussehen soll. Daher lädt er das CSS herunter (sofern vorhanden), interpretiert es und erstellt ein weiteres Modell: das **CSS Object Model** (**CSSOM**). Erst wenn das gesamte CSS verarbeitet wurde, kann der Browser etwas auf dem Bildschirm anzeigen. Daher bezeichnet man CSS als *render-blocking* – die Anzeige von Inhalten wird so lange blockiert, bis das CSS abgearbeitet wurde. Je kompakter und einfacher das CSS ist, desto schneller

kann der Browser es verstehen. Verschachteln Sie Ihre CSS-Selektoren also nicht stärker als notwendig.

- Content ist da, Gestaltung ist da – reicht das aus, um eine Webseite darzustellen? Noch nicht, denn zuerst muss der **Render-Tree** erstellt werden. DOM und CSSOM enthalten nämlich noch Inhalte, die gar nicht sichtbar sind – `meta`-Angaben im HTML oder mit CSS ausgeblendete Inhalte beispielsweise.
- Schließlich erzeugt der Browser das Layout, indem er die Größenverhältnisse errechnet. Ein `article` mit `width: 50%` würde beispielsweise bei einem 320-px-Viewport 160 px groß sein, auf einem 1 200-px-Viewport jedoch 600 px.
- Mit diesen Informationen hat der Browser alles, was er zur Anzeige einer Webseite benötigt. Ein Browser durchläuft diese Schritte jedoch mehrmals. So muss er beispielsweise das Layout neu aufbauen, wenn gescrollt wird.

JavaScript verändert diesen Prozess ein wenig: Sie können damit Inhalte erzeugen oder auch die Anzeige von Elementen steuern – JavaScript kann also sowohl das DOM als auch das CSSOM verändern. Nehmen wir einmal an, ein Browser beginnt damit, das DOM zu konstruieren. Irgendwo entdeckt er ein JavaScript. Was muss er tun?

- Ohne das Skript zu kennen, kann der Browser nicht wissen, ob es das DOM verändert. Also muss er mit der Konstruktion des DOMs aufhören und das Skript zuerst interpretieren. JavaScript ist *parser-blocking*, weil es die Konstruktion des DOMs verzögert.
- Gleichzeitig kann der Browser das JavaScript nicht ausführen, bevor er auch das CSSOM aufgebaut hat – es könnte ja genauso gut sein, dass mit dem Skript die Darstellung beeinflusst werden soll.

Kritisches CSS auslagern | Eine Möglichkeit ist die Arbeit mit kritischem CSS. Huch, kritisches CSS? Was ist das? Man versteht darunter alle CSS-Angaben, die für die unmittelbar beim Aufruf einer Seite sichtbaren Inhalte benötigt werden. Ein Blick in das Verhalten eines Browsers beim Aufruf einer Website verdeutlicht das Verfahren:

- Externe Stylesheets blockieren die Anzeige von Inhalten (*render-blocking*). Der Browser lädt also zunächst die HTML-Dateien

und fordert dann die Stylesheets an – er zeigt jedoch noch keine Inhalte, bis er mit dem Herunterladen der Stilangaben fertig ist.

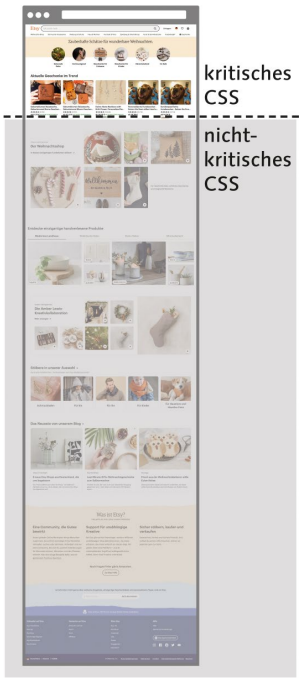
- Prinzipiell ist dieses Vorgehen sinnvoll, denn schließlich wüsste der Browser zwar, *was* er zeigen soll, aber noch nicht, *wie*. Gerade bei komplexen Stylesheets verlängert sich die Wartezeit jedoch deutlich, besonders in Mobilfunknetzen.
- Die Idee von »kritischem CSS« ist nun, das CSS in zwei Teile aufzusplitten: alles, was für die direkt sichtbaren Inhalte notwendig ist, sowie alles andere.

Das klingt an dieser Stelle seltsam – haben wir unsere CSS-Dateien nicht gerade erst zusammengefügt, um weniger Requests zu erzeugen? Ja, genau – und deshalb soll das kritische CSS als `style`-Block direkt in die HTML-Seiten hineingeschrieben werden, während das übrige CSS wie gewohnt über eine Datei nachgeladen wird. Die Idee dahinter: Es gibt keinen Grund, warum eine Nutzerin oder ein Nutzer auf Style-Angaben warten sollte, die noch nicht zu sehen sind. Der Browser kann somit früher mit dem Rendern des oberen Contents beginnen und alles andere nachladen. Prinzipiell sieht das Vorgehen so aus:

```
<!doctype html>
  <head>
    <style> /* kritisches CSS inline */ </style>
    <script> loadCSS("pfad/zu/nichtkritischen/styles.css");
    </script>
  </head>
  <body>...</body>
</html>
```

▲ **Listing 9.6**
Prinzip des kritischen CSS

Manuell ist das sehr aufwendig: Sie müssten sich Ihre Website bei verschiedenen Viewport-Größen anschauen und alle CSS-Angaben identifizieren, die für den sichtbaren Bereich nötig sind. Anschließend kopieren Sie diese aus der CSS-Datei direkt ins HTML. Glücklicherweise hat Jonas Ohlsson mit dem »Critical Path CSS Generator« ein Tool entworfen, das dies übernimmt. Dazu geben Sie zunächst unter ❶ die URL der Webseite ein und



▲ **Abbildung 9.11**
Kritisch ist CSS immer dann, wenn es für die Anzeige im aktuellen Viewport notwendig ist (*www.etsy.com*) – diese Stile sollten so schnell wie möglich bereitstehen.

Tipp: Lieber doppelte Angaben behalten
Theoretisch könnten Sie das kritische CSS aus der externen Datei entfernen, sobald es im `<head>` der HTML-Datei untergebracht ist. Wir raten Ihnen jedoch, eher davon abzusehen, denn die Wartbarkeit würde darunter doch arg leiden, und die Ersparnis in der Dateigröße wäre selten groß.

kopieren anschließend unter ❸ das gesamte CSS hinein. Ein Klick auf den Button ❷ extrahiert nun die kritischen Stilangaben. Diese kopieren Sie anschließend in den `<head>` Ihrer Webseite, während das übrige CSS asynchron geladen wird. Dieses Beispiel verwendet dazu das JavaScript »loadCSS«, das unter *https://github.com/filamentgroup/loadCSS* ausführlich dokumentiert ist. Anschließend heißt es Testen, Testen, ob das Tool alles richtig erkannt hat.

Abbildung 9.12 ▸
Critical Path CSS Generator (*https://jonassebastianohlsson.com/criticalpathcssgenerator*)

Natürlich müssten Sie diesen Schritt für jede Seite und bei jeder Änderung wiederholen. In der Praxis arbeiten Entwicklerinnen und Entwickler daher meist mit Automatisierungstools wie beispielsweise:

- *https://github.com/addyosmani/critical*
- *https://github.com/pocketjoso/penthouse*

JavaScript asynchron laden | Auch JavaScript blockiert den Browser dabei, das DOM aufzubauen. Um das zu vermeiden, sollten JavaScript-Dateien entweder ganz unten im Quelltext (vor dem schließenden `</body>`) eingebunden oder asynchron geladen werden:

```
<script src="analytics.js" async></script>
```

▲ Listing 9.7
Asynchrones Laden von JavaScript

Das `async`-Attribut sorgt dafür, dass JavaScript die Konstruktion des DOMs nicht mehr blockiert – der Browser fährt mit dem Aufbau des DOMs fort, während er das Skript herunterlädt. `async`

garantiert jedoch nicht, dass das DOM auch fertig aufgebaut ist, wenn das Skript ausgeführt wird – es könnte also sein, dass das Skript etwas verändern möchte, von dem der Browser noch gar nichts weiß. Eine Alternative ist das `defer`-Attribut (statt `async`) – der Browser lädt dieses Skript nun parallel zur Konstruktion des DOMs herunter, führt es jedoch erst aus, nachdem das Dokument aufgebaut wurde. Vitaly Friedman empfiehlt, `defer` vorzuziehen und nicht mit `async` zu mischen.

Lazy Loading | Schließlich gibt es »Lazy Loading«. Bei diesem Verfahren lädt ein Browser Ressourcen nicht direkt beim Aufruf der Seite, sondern erst, wenn sie auch tatsächlich benötigt werden. Das wird bei Bildern häufig eingesetzt, die außerhalb des gerade sichtbaren Bereichs des Viewports sind.

Es gibt verschiedene Möglichkeiten der Umsetzung. Eine sehr gute Lösung auf Basis von JavaScript ist »lazysizes« von Alexander Farkas (*https://github.com/aFarkas/lazysizes*). Dort findet sich eine umfangreiche Dokumentation. Prinzipiell ist das Procedere so:

- Laden Sie das Skript herunter, und binden Sie es in die Webseite ein.
- Setzen Sie nun die Attribute `data-src` und `data-srcset` innerhalb des `img`-Elements ein. Lazysizes kümmert sich automatisch um das Laden der Grafiken, sobald sich die Scrollposition den Bildern nähert. Bei responsiven Bildern sieht das beispielsweise so aus:

```
<img data-sizes="auto" alt="…" data-src="image2.jpg"
data-srcset="image1.jpg 300w, image2.jpg 600w, image3.
jpg 900w" class="lazyload" />
```

- Deaktiviertes JavaScript kann unterstützt werden, indem das `img`-Element in einem `noscript`-Block wiederholt wird:

```
<noscript>
  <img src="image.jpg" alt="…"/>
</noscript>
```

Mittlerweile unterstützen sehr viele Browser Lazy Loading auch nativ, besonders für Bilder. Der Support für `<iframes>` ist noch nicht flächendeckend, aber das dürfte sich vermutlich beim Lesen des Textes bereits geändert haben. Natives Lazy Loading können

Alternativen zur Umsetzung
Natürlich gibt es viele weitere Möglichkeiten, Lazy Loading umzusetzen. Einige bekannte Beispiele sind:

- **Lazy Load** von Mika Tuupola (*http://www.appelsiini.net/projects/lazyload*)
- **ImageLoader** aus der YUI-Bibliothek (*http://yuilibrary.com/yui/docs/imageloader*)

◀ **Listing 9.8**
Einbindung von Bildern mit dem `data-src`-Attribut

◀ **Listing 9.9**
Fallback bei deaktiviertem JavaScript

Sie verwenden, indem Sie bei `<img>` oder `<iframe>` das Attribut `loading="lazy"` hinzufügen:

Listing 9.10 ▸
Natives Lazy Loading

```
<img src="bild-1.jpg" loading="lazy" alt="…">
```

Neben `lazy` gibt es noch `eager` (der Browser lädt das Bild sofort beim Aufruf der Seite) und `auto` (der Browser entscheidet).

9.4 Nachhaltigkeit bei der Webentwicklung

Was wir eben in Abschnitt 9.3 beschrieben haben, ist auch mit der Nachhaltigkeit verbunden. Wenn eine Website performant ist, bedeutet das auch eine Reduzierung von Datenübertragung und meistens auch eine Verringerung des CO_2-Ausstoßes. Trotzdem sollen hier noch ein paar Punkte speziell für die Webentwicklung erwähnt werden. Gerade, wenn eine Website von Anfang an sauber und ordentlich gestaltet und programmiert wurde, hat man bei der späteren Optimierung weniger Arbeit.

9.4.1 Sustainability-Budgets

Ähnlich wie beim Performance-Budget bietet sich ein Sustainability-Budget an, das von Anfang an bei Konzeption, Gestaltung und Entwicklung berücksichtigt wird. Statt also in Kilobytes oder Megabytes zu rechnen, geht man hier einen Schritt weiter und legt ein CO_2-Budget fest. Da man CO_2 nicht einfach berechnen kann, bietet sich die Website *www.websitecarbon.com* an.

Tom Greenwood schlägt in seinem Buch »Nachhaltiges Webdesign« vor, zunächst die Metriken von Websites Ihrer Mitbewerber zu messen, sofern diese für Nachhaltigkeit optimiert sind. Auf diese Weise entsteht eine Benchmark, nach der Sie sich richten können.

Legen Sie dann einen Content-Prototyp nur mit den Inhalten ohne weitere Elemente wie Bilder, Videos oder Skripte an. Dies ist das Best-Case-Szenario, also eine Messung dessen, was mindestens nötig ist, wenn ausschließlich Inhalte vermittelt werden sollen.

Mit Hilfe der Benchmark in der Branche und des individuellen Best-Case-Szenarios können Sie sich ein Budget festlegen. Ziel sollte es immer sein, mindestens genauso gut wie die Besten in der Branche zu sein.

Energieverbrauch vom Browser
Die Datenübertragung ist häufig der wichtigste Faktor, um den Energieverbrauch gering zu halten. Trotzdem sollte man hier nicht das Endgerät vergessen. Es nützt nichts, wenn Sie den Datenverbrauch gering halten, aber aufgrund aufwendiger Inhalte die CPU des Clients bis zum Anschlag aufdrehen. Safari bietet hier mit dem Energy Impact Monitor ein Tool an, das anzeigt, wie viel CPU-Leistung eine Website verbraucht.

9.4.2 Sauberer und schlanker Code

Hierzu gehören viele kleinere Bereiche. Dies fängt damit an, die Anzahl der Code-Zeilen möglichst gering und kurz zu halten und Code-Wiederholung zu vermeiden. Können Sie die Komplexität des Codes verringern, indem Sie die Anzahl der Abfragen reduzieren, dann sollten Sie dies tun. Vermeiden Sie außerdem unnötige Plug-ins und nicht verwendete Bausteine von Bibliotheken. Unter Umständen kann es daher sinnvoll sein, den Code selbst zu schreiben. Je sauberer und schlanker der Code ist, desto weniger wird die CPU des Clients beansprucht, was Energie spart und den Akku schont.

9.4.3 Effiziente Programmiersprache wählen

Sauberer und effizienter Code ist die eine Sache, aber auch die verwendete Programmiersprache ist von enormer Bedeutung, denn Programmiersprachen sind unterschiedlich effizient (siehe Tabelle 9.3). So gelten beispielsweise C, C++, Rust und Java als die *saubersten* Programmiersprachen, aber auch JavaScript arbeitet noch sehr energiesparend. Sehr schön ist auch zu sehen, dass PHP immer noch wesentlich weniger Energie benötigt als beispielsweise Python.

Programmiersprache	Energie
C	1,00
C++	1,34
Java	1,98
JavaScript	4,45
TypeScript	21,50
PHP	29,30
Ruby	69,91
Python	75,88
Perl	79,58

Ausführzeit und Speicherbelegung
Tabelle 9.3 legt den Fokus rein auf den Energieverbrauch. Nicht beachtet wurden hier die Ausführzeiten und die Speicherbelegungen der einzelnen Programmiersprachen. Wichtig zu erwähnen ist, dass es nicht darum geht, eine Sprache aufgrund des Energieverbrauchs zu beurteilen, sondern Energieverbrauch als einen Faktor bei der Wahl der Programmiersprache zu berücksichtigen. So wird man zukünftig kaum C oder C++ für die Webentwicklung verwenden.

◀ **Tabelle 9.3**
Energieeffizienz von Programmiersprachen mit Fokus auf Sprachen, die für das Web verwendet werden. Mehr Informationen zum Thema finden Sie unter *https://greenlab.di.uminho.pt/wp-content/uploads/2017/10/sleFinal.pdf*.

Generell empfiehlt es sich, immer die neueste Version einer Programmiersprache auf dem Server zu verwenden. So sind z. B. neuere Versionen von PHP deutlich schneller und benötigen weniger Ressourcen.

Weniger JavaScript | Wie auch bei Bildern und Videos sollten Sie sich stets fragen, ob Sie einen bestimmten JavaScript-Code unbedingt brauchen. Animationseffekte lassen sich beispielsweise ressourcenschonender mit CSS realisieren als mit JavaScript, und nicht jede einfache Website braucht eine große Bibliothek oder ein Frontend-Framework. Damit ist natürlich nicht gemeint, komplett auf JavaScript zu verzichten, sondern vielmehr, JavaScript möglichst sinnvoll zu nutzen, weil es neben dem Datenumfang auch die Auslastung der CPU erhöht. Das gilt ganz besonders auch für die Analytics-Tracking-Skripte (*www.wholegraindigital.com/blog/plausible-vs-google-analytics/*).

Statische Seiten einbauen | Heute werden Websites in der Regel auf Anforderung von einem CMS auf einem Server generiert, meist mit PHP. Solche dynamischen Websites verbrauchen aber erheblich mehr Systemaufrufe (bis 20-mal mehr) und Energie als eine statische HTML-Seite. Statische Seiten haben also Vorteile in Bezug auf schnellere Ladezeit. Das heißt aber natürlich nicht, dass Sie komplett zurück in die 90er gehen sollen und wieder ausschließlich statische Websites erstellen müssen. Aber vielleicht gibt es einzelne Seiten (etwa Dokumentation, Landingpages, Portfolio …), die statisch ausgeliefert werden können, oder aber Sie nutzen einen Static-Site-Generator (siehe Kasten rechts).

Seiten-Caching nutzen | Anstelle von statischen Websites können Sie auch das Seiten-Caching nutzen. Hierbei wird eine Seite beim ersten Aufruf zunächst dynamisch generiert und in einem Cache auf dem Server gespeichert. Alle folgenden Aufrufe erhalten dann die im Cache gespeicherte Version. Hierfür gibt es CMS-basierte Caching-Lösungen (etwa WP Rocket, *https://wp-rocket.me/* für WordPress) oder serverseitige Caching-Lösung (etwa Varnish, *www.varnish-software.com/*). Caching liefert eine Website praktisch aus, als wäre diese statisch.

Generatoren für statische Seiten
Es gibt auch Static-Site-Generatoren, die CMS-betriebene Websites in statische Websites umwandeln können. Bekannte Open-Source-Static-Site-Generatoren sind Jekyll (*https://jekyllrb.com*), Hugo (*https://gohugo.io*) oder Gridsome (*https://gridsome.org*). Viele dieser Generatoren bieten eine Migration von gängigen CMS wie WordPress an.

9.4.4 Progressive Web Apps (PWAs)

Schließlich können Lösungen sinnvoll sein, die aus dem Bereich der Progressive Web Apps (PWAs) kommen. Eine Progressive Web App ist keine native, also in Programmiersprachen der Mobilsysteme geschriebene App, hat aber Funktionen, die sie wie eine App wirken lassen. So kann man sich eine PWA als Icon auf den Homescreen legen, den Inhalt auch offline anzeigen und über Push-Nachrichten kommunizieren. Auch wenn Sie viele Funktionen von PWAs nicht für Ihre Website benötigen, so bieten diese eben die Möglichkeit, Dateien offline auf dem Gerät zu speichern. Speziell für Seiten, die Nutzerinnen und Nutzer immer wieder besuchen (etwa Dokumentationen, Referenzen etc.), lassen sich damit Datenübertragungen und Energie sparen. Damit können die Inhalte auch benutzt werden, wenn es mal keine Internetverbindung gibt. Eine tolle Dokumentation zu PWAs finden Sie auf MDN (*https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps*).

Werkzeuge für PWAs
Gerne verwendet zur PWA-Entwicklung wird die JavaScript-Bibliothek React (*https://reactjs.org*) oder Polymer (*https://polymer-library.polymer-project.org*). Wollen Sie eine bestehende Website in eine PWA umwandeln, dann können Sie dies mit dem PWA-Builder (*www.pwabuilder.com*) machen.

▲ **Abbildung 9.13**
Google Drive ist als PWA realisiert und erlaubt Offline-Zugriff auf Dokumente (Symbol mit dem Häkchen).

9.4.5 Bots blockieren

Bots sind Software-Anwendungen, die automatisierte Aufgaben erledigen, beispielsweise die Inhalte von Websites für Suchmaschinen indizieren (Webcrawler) oder gezielt zur (oft ungewollten) Verwendung herunterladen (Content Scraping). Es gibt also gute Bots und böse Bots. Auf jeden Fall verbrauchen Bots eine enorme Menge an Datenverkehr und erzeugen damit Emissionen, die sich einsparen lassen, wenn man den Großteil der Bots blockieren würde. Bots können beispielsweise über eine Firewall oder über einen Dienst wie Cloudflare (*https://blog.cloudflare.com/de-de/cleaning-up-bad-bots-de-de*) blockiert werden. Ebenso gibt es Webhoster, die solche Firewalls (Web Application Firewall) anbieten.

9.5 Nachhaltigkeit beim Webhosting

Websites werden auf Rechenzentren geladen und gehostet, die einen hohen Energiebedarf haben. Daher können Sie auch bei der Auswahl des Webhosters einen Einfluss auf die CO_2-Emissionen nehmen.

9.5.1 Around the world

Bei der Übertragung von Daten wird eine Menge an Energie benötigt. Hier ist es empfehlenswert, sich mit dem Server-Standort zu befassen. Wenn der auf den ersten Blick günstigere Webhoster seine Rechenzentren irgendwo im fernen Ausland stehen hat, dann ist dies kontraproduktiv für die Umwelt. Wenn Daten um den halben Erdball reisen müssen, wird mehr Energie benötigt und die Wartezeit beim Seitenaufbau gesteigert. Das Thema der Datensicherheit ist hier noch gar nicht berücksichtigt. Wenn Sie also eine Website einrichten, dann sollten sich die Rechenzentren möglichst am Standort befinden, wo auch die meisten Besucherinnen und Besucher zu erwarten sind. International tätige Unternehmen sollten sich auch mit dem Thema »Content-Delivery-Network« beschäftigen, das wir bereits in Abschnitt 9.3.5 behandelt haben.

9.5.2 Green Webhosting

Green-Hosting-Verzeichnis
Eine Liste solcher Anbieter sammelt die Green Web Foundation auf der Website *www.thegreenwebfoundation.org/directory/*.

Schließlich können Sie einen Webhoster auswählen, der seinen Strom nicht aus fossilen, sondern erneuerbaren Energiequellen bezieht, um so die CO_2-Bilanz zu verbessern.

Tom Greenwood hat in seinem Buch »Sustainable Web Design« (*https://abookapart.com/products/sustainable-web-design*) zusammengefasst, was es tatsächlich bedeutet, wenn ein Webhoster sagt, dass er ausschließlich grüne Energie verwendet. Folgende Möglichkeiten stehen einem grünen Webhoster zur Verfügung, um sich als nachhaltiger Webhoster zu bezeichnen:

- erneuerbare Energie selbst herstellen
- erneuerbare Energie fördern
- erneuerbare Energie einkaufen
- Zertifikate (RECs) für erneuerbare Energie kaufen
- CO_2-Kompensation kaufen

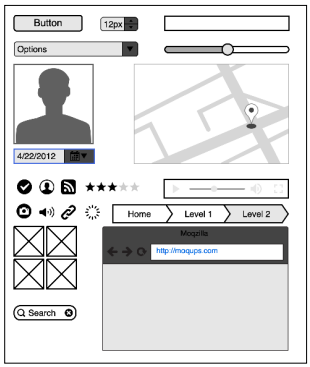
Während die ersten drei Punkte klar in die Richtung 100 % erneuerbarer Energie gehen, sind der Kauf von REC-Zertifikaten und die CO_2-Kompensation eher kritisch zu betrachten. Bei RECs ist nicht garantiert, dass man tatsächlich die Energie im selben Netz bezieht, und bei der CO_2-Kompensation kann der Strom im Rechenzentrum durchaus von fossilen Brennstoffen kommen. Tom Greenwood selbst bringt es auf den Punkt, dass der Mindeststandard sein sollte, dass man wenigstens erneuerbare Energie zukauft.

Und nun? | Mit den Erkenntnissen aus diesem Buch haben Sie eine gute Grundlage für Ihre weitere Arbeit im Webdesign, und Sie stehen am Anfang einer spannenden Reise. Werfen Sie unbedingt auch einen Blick auf die Inhalte im Download-Bereich zu diesem Buch – Sie finden dort viel Material, das es aus Platz- und Kostengründen nicht ins Buch geschafft hat. Wir wünschen Ihnen dabei viele gute Ideen, damit Sie Ihre Ziele im Netz erreichen. Schreiben Sie uns gerne – wir freuen uns, von Ihnen und Ihren Projekten zu hören.

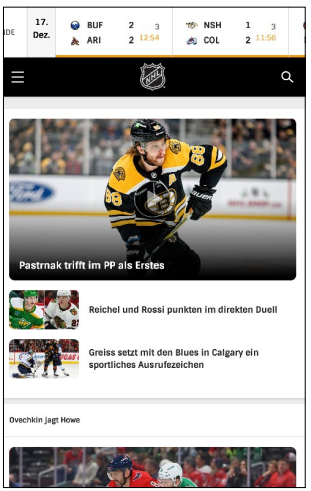
Auf einen Blick

Inhalt

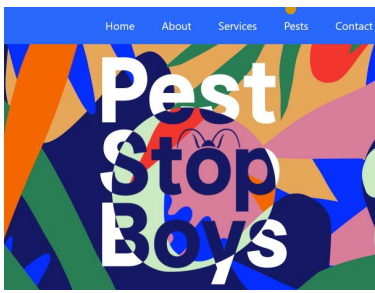
3 Konzeption und Design

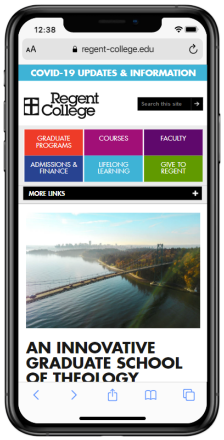


4 Layout und Komposition



5 Typografie im Web

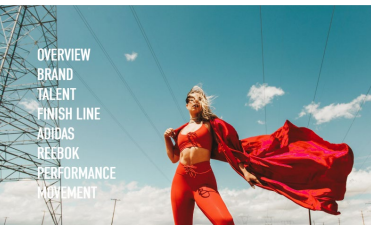


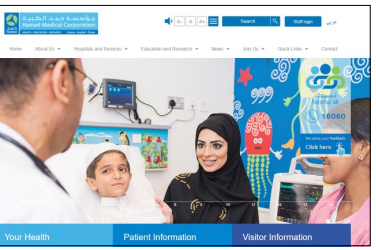


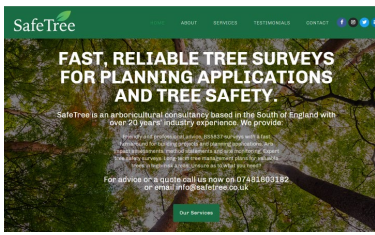
6 Navigationen und Interaktionen

7 Farbe im Web

8 Grafiken, Bilder und Multimedia





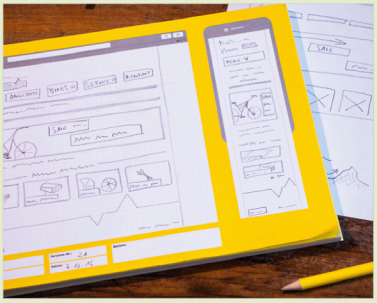


Grundkurs Gutes Webdesign

»Gutes Webdesign – modern und responsive!«

Alles, was Sie über Gestaltung im Web wissen sollten

Dieses Buch zeigt Ihnen, worauf es bei der Gestaltung moderner, attraktiver Websites ankommt. Es führt Sie durch den gesamten Gestaltungsprozess – von der Idee bis zur fertigen Website, zahlreiche Beispiele und Tipps für die technische Umsetzung inklusive!



Mit Layouts und Rastern arbeiten

Ein Farbkonzept entwickeln

Schriften gekonnt einsetzen

Das Einmaleins des Webdesigns

Schritt für Schritt erlernen Sie die Grundlagen guten Webdesigns. So gestalten Sie Websites, die durch gelungene Typo, farbige Akzente und gute Usability im Gedächtnis bleiben.

Website-Konzeption, User Experience und Nachhaltigkeit

Erfahren Sie, wie Sie mithilfe einer soliden Konzeption leicht zu bedienende und übersichtliche Websites designen, und tauchen Sie in aktuelle Themen wie Barrierefreiheit und Nachhaltigkeit ein.

Gestaltung und Technik

Mit einer cleveren Navigation und attraktiver Gestaltung entstehen Websites, die überall gut aussehen – dank HTML, CSS und Responsive Webdesign auch auf Tablet und Smartphone. Grundkenntnisse in HTML und CSS werden vorausgesetzt.

Code-Beispiele und Bonusinhalte zum Download


Der UX-Experte **Björn Rohles** schreibt Bücher und Fachartikel zu nutzungsfreundlicher digitaler Gestaltung. **Jürgen Wolf** ist Webentwickler und Autor mehrerer Standardwerke zur Programmierung.

Auf einen Blick

- Website-Konzeption
- Responsive Webdesign
- Usability und UX
- Gestaltungsgrundlagen
- Web-Typografie, Texte in CSS
- Farblehre, Farbe im Web
- Grafiken, Bilder, Icons, Video
- Navigation und Interaktion
- Buttons, Links, Animationen
- Informationsarchitektur
- CSS-Layouts, Raster
- HTML und CSS
- Barrierefreiheit
- Nachhaltigkeit im Webdesign
- Testen und optimieren
- Performance-Optimierung

Rheinwerk Design

€ 29,90 [D] € 30,80 [A]

Gedruckt in Deutschland
Papier aus nachhaltiger Waldwirtschaft
Mineralölfreie Druckfarben

Für Windows und Mac
Webdesign
ISBN 978-3-8362-9349-5