NEED FOR SPEED

WARUM SCHNELLE WEBSEITEN HEUTE UNABDINGBAR SIND

Matthäus Michalik, Elena Jung

Erst vor wenigen Wochen kündigte Google an, die Seitenladegeschwindigkeit mobiler Webseiten noch stärker im Algorithmus zu gewichten. Es ist an der Zeit, dass jeder Webseiten-Betreiber sich Gedanken über seine Seitenladegeschwindigkeit macht, diese überprüft und anschließend Vorkehrungen zur Performance-Optimierung trifft. Doch längst ist Page-Speed nicht nur ein Thema der Suchmaschinenoptimierung, auch die Conversion-Rate kann von einer schnellen Seite profitieren und zudem wird mit einer schnellen Seite ein besseres Nutzererlebnis geschaffen. Das kann aus einem einmaligen Besucher einen wiederkehrenden Nutzer machen.

DIE AUTORIN

Elena Jung ist Senior SEO Consultant bei der Claneo GmbH. Dort leitet sie das strategische und technische Onpage-Team. Mit ihrer über 7-jährigen Erfahrung berät sie Unternehmen bei komplexen Strategien und technischen Herausforderungen.

DER AUTOR

Matthäus Michalik ist Gründer und Geschäftsführer der Claneo GmbH. Mit seinem Team berät er internationale Unternehmen in verschiedenen Performance-Marketing-Disziplinen, insbesondere im SEO-, SEA- & Content-Marketing-Bereich.

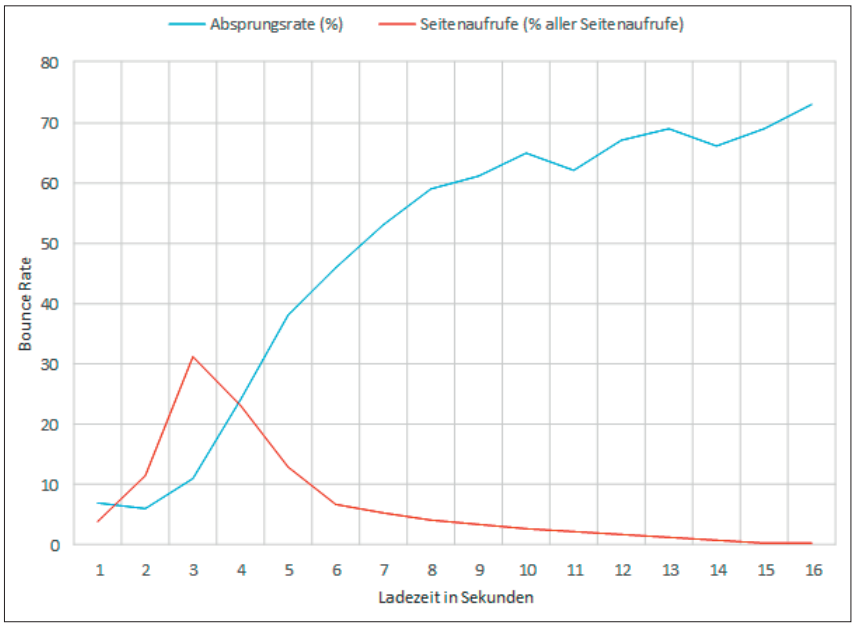

Abb.1: Pingdom-Studie: steigende Absprungrate bei steigender Ladezeit

> „Laut einer Studie von Akamai können 100 Millisekunden zu einem Rückgang der Conversion-Rate von bis zu 7 % führen."

Wie schnell eine Webseite aufgerufen werden kann, hängt von vielen Faktoren ab, und so ist es für Webseiten-Betreiber oft schwierig, eine schnelle Webseite für ihre Nutzer zu gewährleisten. Selbst in der heutigen Zeit, in der wir mobil über HSDPA surfen und zu Hause einen Glasfaser-Anschluss haben, sind Webseiten zu langsam.

Nutzer hassen es, zu warten, egal, ob in der Offline- oder in der Online-Welt. Ein schnelles direktes Feedback ist immer gewünscht und so haben Nutzer eine hohe Anforderung an Webseiten. Insbesondere auf mobilen Endgeräten ist die Geduld von Nutzern endlich. Lädt eine Webseite zu langsam, wird der Nutzer dort nicht kaufen und springt wieder zurück in die Suchergebnisse, um eine andere Webseite in den Top 10 zu besuchen. Bereits 2012 hat Amazon errechnet, dass eine Ladezeit länger als 100 Millisekunden den Umsatz um etwa 1 % reduziert, und auch aktuelle Studien von Pingdom belegen, dass die Bounce-Rate bei zunehmender Ladezeit steigt.

Mit zunehmender Ladezeit sinkt auch die Conversion-Rate. Laut einer Studie von Akamai können 100 Millisekunden zu einem Rückgang der Conversion-Rate von bis zu 7 % führen. Das ist besonders für Online-Shops kritisch, da die Conversion-Rate direkt an die Business-KPIs gekoppelt ist. Wenn Webseiten nicht den Erwartungen der Nutzer entsprechen und keine schnelle Ladezeit gewährleistet werden kann, gehen Marken und Unternehmen Kunden und Umsätze verloren.

Betrachtet man die technischen Aspekte, so hat die Seitenladegeschwindigkeit auch einen direkten Einfluss auf das Crawling einer Webseite. Google reserviert für das Crawling einer Webseite ein gewisses Zeit-Kontingent. Ist eine Webseite langsam, können weniger URLs besucht und verarbeitet werden – die Crawl-Rate sinkt. Insbesondere für sehr große Webseiten mit vielen Unterseiten ist ein effizientes Crawling für ein gutes Ranking entscheidend.

Doch auch wenn an diesen Beispielen deutlich wird, wie wichtig PageSpeed ist – in der Suchmaschinenoptimierung ist das ein Faktor von vielen. Wenn man als Webseiten-Betreiber seine Zielgruppe nicht kennt, für diese keine relevanten Inhalte zur Verfügung stellt und andere technische Baustellen hat, kann die Seitenladegeschwindigkeit noch so gut sein, sie ist nicht das Allheilmittel für eine schlecht optimierte Webseite.

Es gibt viele verschiedene Aspekte der PageSpeed-Optimierung, angefangen von optimierten Bildern bis hin zu der richtigen Server-Struktur, und so lassen sich einige einfach umsetzen, andere Probleme sind komplexer.

Wie wird eine Seite überhaupt geladen?

Um zu verstehen, welche Möglichkeiten man bei der Performance-Optimierung einer Webseite hat, muss man verstehen, wie eine Webseite von einem Browser angefragt und verarbeitet wird.

Die erste Ressource, die ein Browser anfragt, sobald ein Nutzer eine URL eingibt, ist das HTML-Dokument. Der Browser fragt nach einem DNS-Lookup der Domain (hier wird die Domain in eine IP umgewandelt) die Datei auf dem jeweiligen Server, der über eine IP angesteuert wird, an und erhält das Dokument.

Parsing, Rendering und Layouting

Anschließend wird das HTML-Dokument vom Browser gelesen. Das erfolgt im Quellcode von oben nach unten. Stößt der Browser auf weitere Dateien, wie z. B. CSS-, JavaScript-Dateien oder Bilder, so fragt der Browser diese

> „Our goal is actually to make improvements to search that just answer the users information need and get them to their answer faster and faster." – *Google*

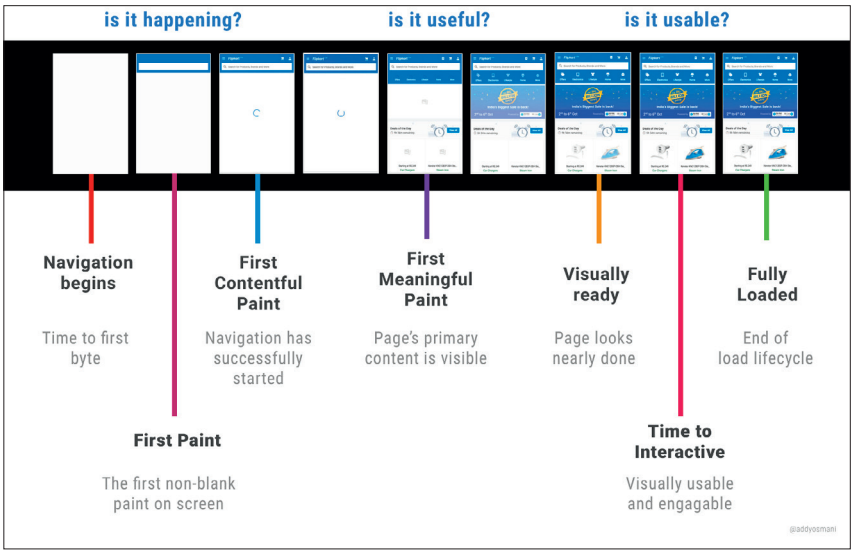

Abb.2: Aufbau einer mobilen Webseite – vom TTFB zu vollständigen Seite *(Quelle: @addyosmani)*

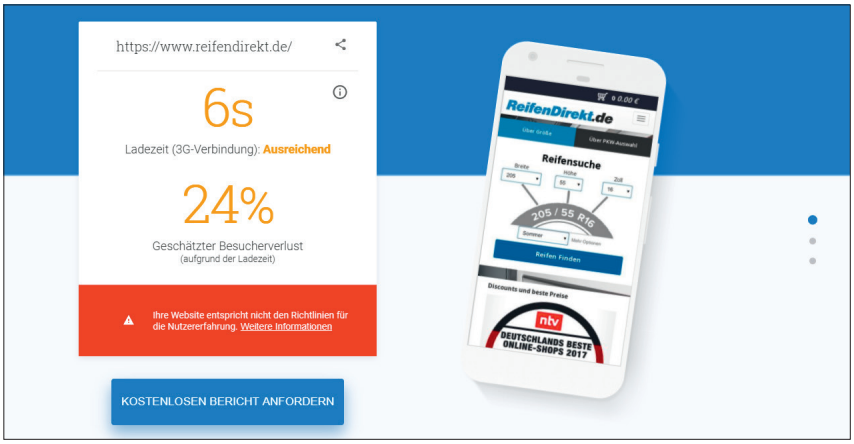

Abb.3: Google Test My Site am Beispiel von reifendirekt.de

ebenfalls beim Server an. Anschließend wird das Dokument vom Browser geparst und geladen. Erst sobald alle Dateien geparst und geladen wurden, kann der Browser die Webseite rendern. Je nachdem, wie eine Webseite und der HTML-Quellcode konzipiert sind, muss der Browser auf verschiedene Dateien warten.

Moderne Browser können mehrere Anfragen parallel stellen. Somit stellen Browser sicher, dass alle Webseiten-Assets schnell geladen werden. Unter Webseiten-Assets versteht man einzelne Dokumente, die für das Darstellen einer Webseite benötigt werden. Das Problem ist jedoch, dass moderne Webseiten auf sehr viele Assets zurückgreifen und es somit zu Abhängigkeiten kommt bzw. einzelne Dateien sich gegenseitig blockieren.

Als „First Paint" bezeichnet man das erste optische Feedback, das ein Nutzer bekommt – er sieht somit keinen weißen Bildschirm mehr, sondern die ersten Inhalte werden geladen. Der First Paint ist eine der wichtigsten KPIs, wenn es darum geht, die Ladezeit zu messen oder zu monitoren.

Welche Besonderheiten gibt es im mobilen Bereich?

Im mobilen Bereich ist die Seitenladegeschwindigkeit sehr von dem verwendeten Netz abhängig. Das Problem ist die hohe Latenz der verschiedenen Netze und somit kommt es zu Verzögerungen, um Daten-Pakete zu verschicken. Umso wichtiger ist es, den „First Paint", also das erste optische Feedback für den Nutzer, nach vorne zu verlagern und so schnell wie möglich dazustellen. Dazu muss man sicherstellen, dass die Roundtrips zwischen Browser und Server auf ein Minimum reduziert werden. Unter Roundtrip wird die Anfrage vom Browser zum Server und vom Server wieder zurück zum Browser verstanden. Das kann z. B. eine Anfrage des Browsers an den Server und ein Datenpaket vom Server an den Browser sein, also eine „Kommunikations-Runde".

Wie schätzt man den aktuellen Stand ein?

Getreu dem Motto „Our goal is actually to make improvements to search that just answer the users information need and get them to their answer faster and faster" ist Google stets bemüht, Webseiten-Betreibern zu helfen, ihre Seitenladegeschwindigkeit zu verbessern. Dazu stellt Google verschiedene Ressourcen zur Verfügung (*developers.google.com/speed/*), aber auch einzelne Tools wie die Google PageSpeed Insights (*developers.google.com/speed/pagespeed/insights/*), das kürzlich gestartete Google Test My Site (*testmysite.withgoogle.com/intl/de-de*) oder das Google Lighthouse-Tool (*developers.*

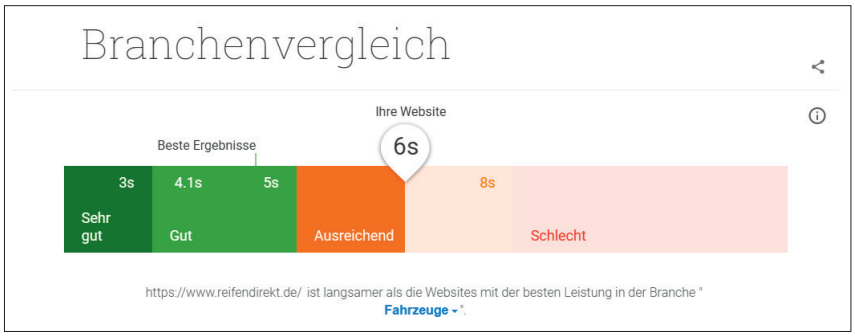

Abb.4: Google Test My Site – Seitengeschwindigkeit von reifendirekt.de im Branchenvergleich

Abb.5: Google PageSpeed Insights geben Aufschluss über die Page-Performance einer URL

google.com/web/tools/lighthouse/) helfen dabei, mehr über die Beschaffenheit einer Seite zu erfahren.

Mittels Google Test My Site lässt sich die Performance der mobilen Seite überprüfen und mit dem Branchen-Durchschnitt vergleichen. Außerdem gibt das Tool Aufschluss, wie viele Nutzer man schätzungsweise durch die aktuelle Ladezeit der mobilen Webseite verliert und ob die Webseite Googles Richtlinien zur optimalen Nutzererfahrung entspricht.

Mit Google PageSpeed Insights bekommt man einen ersten schnellen Überblick zum PageSpeed einer Seite. Die Darstellung und Bewertung ist allerdings eher für den ersten Eindruck geeignet.

- » Antwortzeit des Servers reduzieren
- » Bilder optimieren
- » CSS reduzieren
- » HTML reduzieren
- » JavaScript reduzieren
- » Zielseiten-Weiterleitungen vermeiden
- » Browser-Caching nutzen
- » Komprimierung aktivieren
- » JavaScript- und CSS-Ressourcen, die das Rendering blockieren, in Inhalten „above the fold“ (ohne Scrollen sichtbar) beseitigen
- » Sichtbare Inhalte priorisieren

Will man tiefer in die Probleme einsteigen, hilft Googles eigener Browser Chrome. Mit den Chrome-Developer-Tools kann man eine Webseite auf ihre einzelnen Bestandteile hin überprüfen und Google Lighthouse hilft, Performance-Probleme zu identifizieren.

Natürlich gibt es noch weitere Tools, die helfen, die Ladezeit einer Webseite zu analysieren. Eins der am häufigsten verwendeten und selbst von Google genutzten Tools ist WebPagetest (*www.webpagetest.org*). Als weitere kostenlose Tools sind zu nennen der Pingdom Website Speed Test (*tools.pingdom.com*), Yellow Lab Tools (*yellowlab.tools*) oder auch GTmetrix (*gtmetrix.com*).

Alle Optimierungsmaßnahmen im Überblick

Bilder sind einer der größten Optimierungshebel

Bilder und Grafiken sind die am häufigsten verwendeten Assets auf einer Webseite und so steckt hier oft viel Potenzial zu deren Optimierung. Es gibt verschiedene Formate, die für Bilder und Grafiken verwendet werden, doch nicht alle diese Formate sind ideal für das Internet. Je nach Anwendungsfall kann man zwischen JPG, GIF, PNG, SVG und WebP unterscheiden. Leider wird WebP noch nicht von allen Browsern unterstützt - siehe Abbildung 6.

Die Größe und Qualität eines Bildes ist entscheidend und bestimmt die Dateigröße des Dokuments. So sollte man darauf achten, dass Bilder auf einer

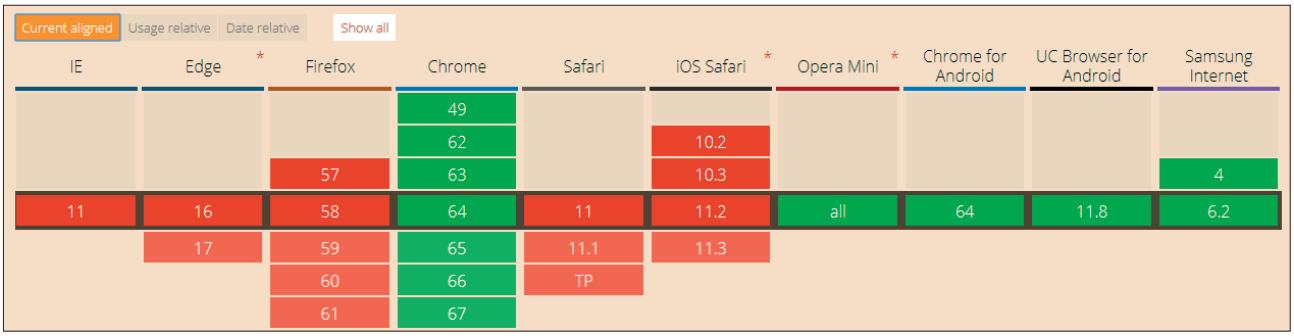

Abb.6: Das WebP-Bild-Format wird nicht von allen gängigen Browsern unterstützt *(Quelle: caniuse.com)*

Webseite immer in der exakten Größe, in der sie auch verwendet werden, implementiert werden. Das browserseitige Herunterskalieren von Dateien sorgt für unnötig große Bild-Dateien. So sieht man oft, dass das Logo im Header, welches eigentlich nur wenige Pixel breit und hoch ist, als originales Logo, welches für den Druck verwendet wird, hochgeladen wurde.

Des Weiteren wird sehr oft vergessen, Meta-Daten eines Bildes zu entfernen. Die Meta-Angaben geben Aufschluss, wo oder mit welcher Kamera das Bild geschossen wurde. Das sind Informationen, die den Nutzern keinen Vorteil bringen und somit nichts auf der Webseite bzw. in der Bild-Datei zu suchen haben. Es gibt verschiedene Tools, die solche Meta-Daten entfernen und das Bild bei gleichbleibender Qualität komprimieren. Der Webseitenbesucher sieht zwischen dem originalen und einem für das Web komprimierten Bild keinen Unterschied.

Tools für Bildkomprimierung:

» *jpegmini.com*
» *tinypng.com*
» *kraken.io*

Diese Dienste bieten eine Weboberfläche, mit der jeder Nutzer seine Bilder optimieren kann. Benötigt man eine API, kann man zu Kraken.io (*kraken.io*) oder ShortPixel (*shortpixel.com*) greifen. Nutzt man Wordpress, kann man Plug-ins wie SmashIT oder EWWW Image Optimizer verwenden.

Neben dem Dateiformat, der richtigen Größe sowie der verlustfreien Optimierung der Bilder ist auch deren Integration entscheidend. Eine Möglichkeit, Bilder erst zu laden, wenn diese wirklich benötigt werden, ist Lazy Loading. Mittels Lazy Loading werden Bilder versetzt geladen und erst ausgespielt, wenn der Nutzer tatsächlich in dem Bereich des Bildes crawlt. Insbesondere bei Online-Shops, die sehr viele Produkte auf einer Kategorie-Seite ausspielen, ist Lazy Loading eine gute Möglichkeit, Ladezeit zu sparen.

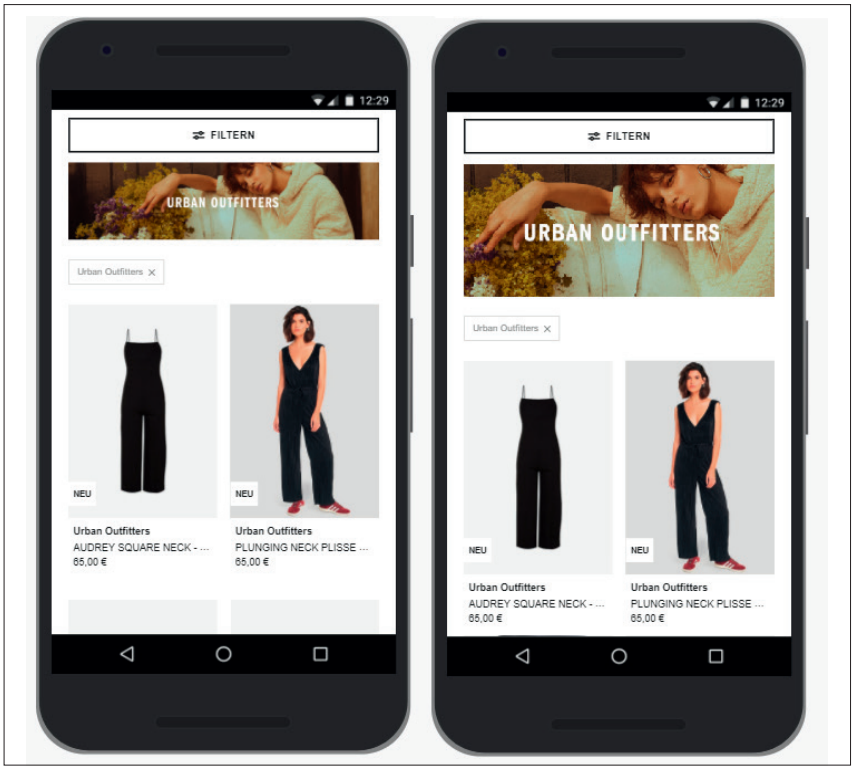

Abb. 7: Beispiel von Responsive Images (scrset + size vs. picture)

CSS-Sprites

Mit dem Einsatz von CSS-Sprites können immer wieder verwendet Grafiken, die z. B. im Header oder Footer einer Webseite integriert sind, als eine Datei geladen werden. Statt jede Grafik einzeln zu laden, werden diese in einer CSS-Sprite-Datei zusammengefasst und einmalig durch den Browser geladen. Das sorgt für weniger Anfragen durch den Browser und somit für eine schnellere Ladegeschwindigkeit.

CSS-Sprites muss man sich wie eine große Schablone vorstellen, die alle relevanten Grafiken enthält. Anschließend wird mittels CSS-Befehlen die benötigte Grafik innerhalb des CSS-Sprites ausgerufen.

Responsive Images

Die Grundidee von Responsive Images ist ganz einfach. Ruft ein Nutzer eine Website von einem Gerät mit einem großen Bildschirm auf, erhält er ein großes Bild, welches für seine Viewport-Größe passt. Ruft ein Nutzer eine Website von einem mobilen Gerät mit einem kleinen Bildschirm auf, erhält er ein kleines Bild, welches für seinen Viewport passt. Denn ein Smartphone-Nutzer sollte vor allem in einem mobilen Netzwerk nicht das riesige Bild für den Desktop-Nutzer in hoher Auflösung laden müssen, wenn es am Ende sowieso nur ganz klein dargestellt wird.

Seit einiger Zeit gibt es neue HTML-Elemente wie ‹srcset› und ‹picture›, die diese Logik unterstützen. Beide Elemente werden mittlerweile von den üblichen Browsern unterstützt. Der Internet Explorer und Opera Mini bilden aktuell die Ausnahmen und müssen mit Polyfills (z. B. picturefill.js) aufgerüstet werden.

scrset + size:

» ‹srcset› listet das Set der möglichen Bilder mit ihren jeweiligen Größen auf
» ‹size› definiert die Bedingungen, ab welcher Größe welches Bild aus dem Set verwendet werden darf

picture:

» ‹picture› definiert ebenfalls unterschiedliche Grafiken für unterschiedliche Viewports, allerdings können

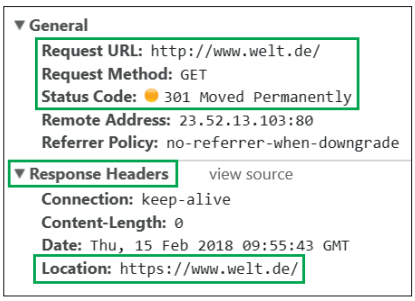

Abb. 8: Der Browser fragt die URL beim Server an (Request) und bekommt in der Antwort (Response) die neue URL (Location) mitgeteilt

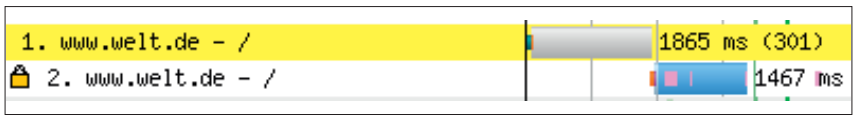

Abb. 9: Webpagetest Wasserfall-Ansicht

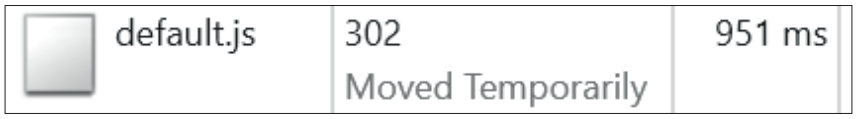

Abb. 10: Ansicht mit Chrome-Developer-Tools › Netzwerk-Ansicht

die Bilder hier auch komplett ausgetauscht und auch mit anderen Seitenverhältnissen hinterlegt werden

Weiterleitungen sollten in der mobilen Welt vermieden werden

Weiterleitungen sind serverseitige Anweisungen, die den Nutzer von einer URL zu einer anderen senden. Die bekanntesten sind permanente 301- und temporäre 302-Weiterleitungen.

Beispiel: Eine Website wurde von HTTP auf HTTPS umgestellt. Durch eine alte Verlinkung gelangt der Nutzer auf die HTTP-Seite und wird auf die HTTPS-Variante weitergeleitet.

http://www.welt.de/ → 301 → **https:**//www.welt.de/

In allen Weiterleitungs-Fällen wird vom Browser ein Dokument angefragt, welches nicht mehr vorhanden ist. Der Server beantwortet diese Anfrage mit der Information, unter welcher URL die gewünschte Seite jetzt zu finden ist. Daraufhin muss die neue URL erneut vom Server abgefragt werden. Diese doppelte Kommunikation zwischen Browser und Server kostet Zeit. Während dieser Wartezeit erhält der Nutzer keinerlei optisches Feedback.

Beispiel: Getestet wurde auf einem Samsung Galaxy S7 innerhalb eines LTE-Netzwerks. Auch wenn keine Daten übertragen werden, kostet die Weiterleitung in diesem Beispielfall **1,8 Sekunden.**

Innerhalb mobiler Netzwerke sind die erforderlichen Kommunikationsschritte, um eine Website aufzubauen, von der Latenz des Netzwerkes abhängig. Die Latenzzeiten können je nach Netzwerkstandard unterschiedlich hoch sein und somit noch mal längere Wartezeiten für den Nutzer bedeuten.

Nicht nur beim Laden des HTML-Dokuments kann es zu Weiterleitungen kommen. Auch andere Ressourcen, die innerhalb des Ladeprozesses angefragt werden, können unter einer anderen URL verfügbar sein, als im HTML hinterlegt ist.

Beispiel: Beim Laden von welt.de wird eine JavaScript-Ressource (default.js) weitergeleitet. Diese Weiterleitung dauert fast eine Sekunde.

Fazit: Es gibt legitime Gründe für Weiterleitungen, aber es sollte berücksichtigt werden, dass Weiterleitungen zu erheblichen Geschwindigkeitsproblemen führen. Jede interne Weiterleitung, die entfernt werden kann, wird die Seite schneller machen.

CSS verschönert die Seite, blockiert sie aber auch

Stylesheets, also CSS-Ressourcen, sind, wie der Name bereits sagt, für das Styling einer Website zuständig. In diesen Dateien werden alle Style-Angaben definiert wie Farben, Schriftarten oder Anordnungen von Elementen u. v. m. Aus diesem Grund gehören CSS-Ressourcen standardmäßig zu den Dateiarten, die den Rendering-Prozess blo-

Abb. 11: welt.de ohne CSS

ckieren, da ohne ihr Vorhandensein die Website nur in einer sehr rudimentären Form dargestellt werden könnte.

Zur Verringerung dieses Effekts, dass der Nutzer lange kein optisches Feedback erhält, weil das CSS noch nicht fertig geladen ist, kann man sich verschiedener Methoden bedienen, um das CSS zu optimieren. Style-Angaben für den sichtbaren Bereich (above the fold) sollte man unbedingt von den Style-Angaben für den nicht sofort benötigten Bereich (below the fold) trennen.

Teil 1 – Inline-CSS

» Style-Angaben anstatt in eine CSS-Datei direkt in das HTML reinschreiben. Dieses Vorgehen nennt man „Inline-CSS". Inline-CSS hat den Vorteil, dass keine extra Ressource vom Server angefragt werden muss, sondern die Stylings sofort mit dem ersten HTML mitgeliefert werden. Der Nachteil ist, dass die HTML-Datei natürlich etwas

größer wird und man die Style-Angaben innerhalb des HTML-Dokuments nicht cachen kann. Aus diesem Grund ist die Nutzung von Inline-CSS nur für die Style-Angaben sinnvoll, die für den sofort sichtbaren Bereich benötigt werden.

» Dabei gibt es eine weitere wichtige Limitierung, die unbedingt beachtet werden sollte. Innerhalb eines Roundtrips können nur 14 KB übertragen werden. Das bedeutet, dass der Above-the-Fold(ATF-)Bereich unbedingt in diese 14 KB passen muss. Aus diesem Grund muss das Inline-CSS wirklich sehr gut durchdacht ausgewählt werden.

Teil 2 – Verlagertes CSS:

» Alle weiteren Style-Angaben, die nicht sofort im oberen Bereich benötigt werden, in einem Stylesheet lassen, welches erst nach dem ATF-Bereich geladen wird. Auf diese Weise kann der kritische Webseitenbereich bereits geladen werden, ohne dass ein großes Stylesheet das Rendering blockiert.

» Da jedes extra Stylesheet, das beim Server angefragt werden muss, die Ladezeit erhöht, sollte es nach Möglichkeit nur ein oder zwei Stylesheets pro URL geben. Auf diese Weise werden die Anfragen an den Server für diese einzelnen Ressourcen so gering wie möglich gehalten. Denn jeder Roundtrip ist in mobilen Netzwerken auch von der Netzwerk-Latenz abhängig und kann somit die Ladezeit wesentlich beeinflussen. Das heißt, sollten aktuell mehr Stylesheets existieren, müssen diese unbedingt zusammengefasst werden. CSS ist dateiunabhängig und so funktioniert das Zusammenlegen meist einfach nur mit Copy & Paste. Oft entstehen mehrere CSS-Dateien, weil die Frontend-Entwickler den Überblick behalten wollen und dafür einzelne Dateien anlegen, ohne diese am Ende dann wieder zusammenzufassen. Wordpress Themes enthalten beispielsweise auch oft standardmäßig mehrere Stylesheets.

Keine @Imports:

Die @Import-Funktion, die innerhalb von Stylesheets genutzt werden kann, um weitere CSS-Dateien zu integrieren, triggert einen weiteren Server-Request. Das heißt, Dateien sollten wirklich zusammengefasst werden und nicht nur über @Import zusammengefügt werden.

Advanced-Tipp: Zukünftig wird über HTML-Imports eine Möglichkeit unterstützt, CSS-Dateien zu laden, welche nicht den Rendering-Prozess blockieren.

Scripts können verlagert werden und die Ladezeit verbessern

Genau wie CSS sind auch JavaScript-Dateien Ressourcen, die standardmäßig den Aufbau der Seite blockieren. Das liegt daran, dass JavaScript Teile des Seitenaufbaus abfragen und verändern kann. Aus diesem Grund wird der Aufbau der Seite angehalten, bis das Script geladen und ausgeführt wurde. Die Position, wo ein Script eingebunden wurde, ist daher entscheidend. Stören in entscheidenden Bereichen, wie im sofort sichtbaren Bereich (ATF), keine Scripts den Aufbau der Seite, kann diese viel schneller geladen und angezeigt werden.

Anders als bei CSS müssen JavaScript-Elemente nicht zwangsläufig einen sichtbaren Effekt haben. Es kann also sein, dass die Seite ganz normal mit HTML und CSS aufgebaut werden kann und durch JavaScript schon vorhandene Elemente nur verändert werden. In diesem Fall müssen die Scripts nicht direkt im oberen Bereich der Seite geladen werden.

Beispiele:

» Eine Akkordeon-Box ist bereits zu sehen, aber erst durch einen Klick und JavaScript wird die Box aufgeklappt.

» Ein Bild ist bereits zu sehen, aber erst durch einen Klick und JavaScript gibt es einen Lightbox-Effekt und das Bild wird noch mal in einer großen Bildergalerie-Ansicht angezeigt.

Es ist aber auch möglich, dass JavaScript direkt im Above-the-Fold-Bereich benötigt wird oder sogar die gesamte Website mittels JavaScript aufgebaut wird. In diesem Fall sollte man analog zu CSS die Inhalte der Scripts teilen.

Teil 1 – Inline-JavaScript: JavaScript-Inhalte, die für den initialen Aufbau der Seite oder für den ATF-Bereich unabdingbar sind, sollten genau wie CSS-Inline eingefügt werden. Dieser Teil muss sehr klein sein und schnell ausgeführt werden.

Teil 2 – Verlagertes JavaScript:

JavaScript-Inhalte, die nicht sofort benötigt werden, sollten ganz ans Ende des HTML-Dokuments verschoben werden. Den gleichen Effekt haben die Attribute „defer“ oder „async“. Sie können zu der Einbindung des Scripts hinzugefügt werden und senden verschiedene Botschaften:

» Async sagt: „Das Script muss nicht genau an dieser Stelle ausgeführt werden. Es darf parallel zum Seitenaufbau geladen werden, ohne diesen zu blockieren, und wird erst darauffolgend ausgeführt.“

```
<script async src="beispiel.js">
```

» Defer sagt: „Das Script muss erst am Ende ausgeführt werden. Es darf parallel zum Seitenaufbau geladen werden, ohne diesen zu blockieren, und wird erst ganz am Ende des Ladeprozesses ausgeführt.“

```
<script defer src="beispiel.js">
```

Je nach Abhängigkeiten der Elemente oder Scripts kann dies einen praktischen Unterschied machen. Im Idealfall arbeitet man mit Entwicklern zusammen, um sicherzustellen, dass alle Java-

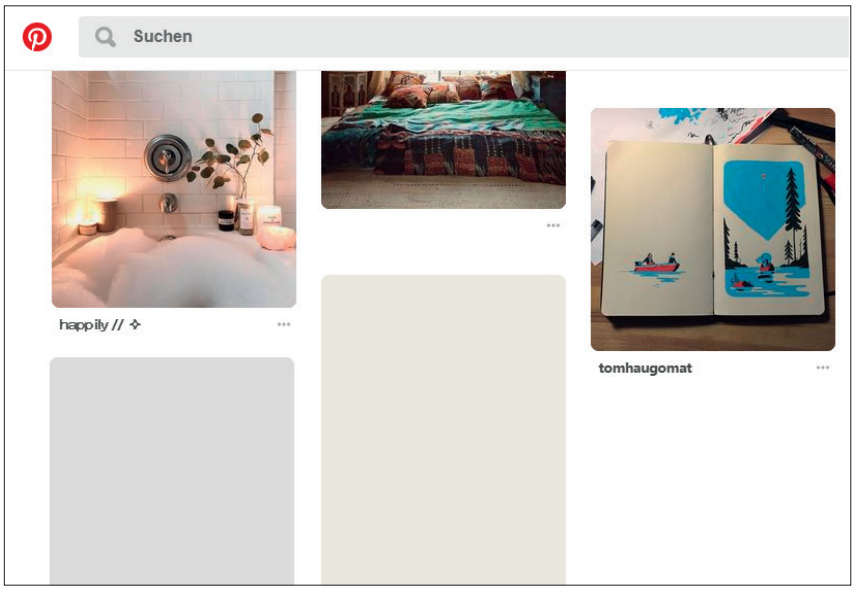

Abb. 12: Pinterest lädt immer erst neue Bilder nach, wenn man auch wirklich weiter nach unten scrollt

Script-Dateien asynchron oder verlagert geladen werden und die Funktionalität der Seite trotzdem gewährleistet bleibt.

Genau wie bei Stylesheets sollte auch bei den JS-Dateien versucht werden, so viele Dateien wie möglich zusammenzufassen, um Server-Request zu reduzieren. Der Prozess ist allerdings nicht so leicht wie bei CSS, da es in JS Abhängigkeiten gibt, die beachtet werden müssen, welche beim einfachen Zusammenkopieren kaputtgehen können.

JavaScript ist nicht immer schädlich für die Ladezeit. Es kann auch genutzt werden, um die Datenmenge, die geladen werden muss, zu reduzieren, z. B. indem Inhalte, die nicht above the fold liegen, also below the fold, auch erst dann geladen werden, wenn der Nutzer anfängt, dorthin zu scrollen – sogenanntes Lazy Loading.

Minification: Dateien sollten in ihrer kleinstmöglichen Form bereitgestellt werden

Minifizierung bedeutet, eine Datei am Ende des Entwicklungsprozesses in ihrer kleinstmöglichen Form abzuspeichern. Das funktioniert besonders gut mit JavaScript- und CSS-Dateien. Im Web lassen sich diverse Tools finden, die sich auf verschiedene Dateiformate spezialisiert haben. Diese Tools verändern natürlich nicht die Funktionalität des Codes, sondern fassen diesen nur so klein wie möglich zusammen. Bei dem Minifizierungsprozess werden alle Leerzeichen und Zeilenumbrüche entfernt, was die Dokumente etwas kleiner macht.

CSS-Tools:

» CSS Nano: *github.com/ben-eb/cssnano*
» CSSO: *github.com/css/csso*

JavaScript-Tools:

» Uglify: *github.com/mishoo/UglifyJS2*
» Google Closure Compiler: *developers.google.com/closure/compiler/*

Da es allerdings sehr lästig sein kann, Dateien nach ihrer Bearbeitung immer wieder durch Online-Tools zu jagen, ist es wesentlich entspannter, sogenannte Build-Tools wie Grunt oder Gulp in die Entwicklungsumgebung zu integrieren. Mit deren Hilfe kann man den Minifizierungsprozess automatisieren, mit eigens dafür entwickelten Funktionen wie z. B. gulp-minify oder gulp-uglify.

DNS-Lookups meiden

DNS steht für Domain – Name – System. Jeder Domainname muss in eine IP-Adresse umgewandelt werden.

```
// ==ClosureCompiler==
// @compilation_level SIMPLE_OPTIMIZATIONS
// @output_file_name default.js
// ==/ClosureCompiler==

// ADD YOUR CODE HERE
function hello(name) {
  alert('Hello, ' + name);
}
hello('New user');
```

Abb. 13: Closure Compiler Beispiel-Code: vorher 102 Bytes

```
function hello(a){alert("Hello, "+a)}hello("New user");
```

Abb. 14: Closure Compiler Beispiel-Code: Nachher 68 Bytes = 33 % Ersparnis

Beispiel: welt.de → 195.50.176.200

Für jede Ressource, die angefragt wird, muss pro Domain ein DNS-Lookup getätigt werden, d. h., der Server muss die IP-Adresse für die Domain erst nachschlagen. Dieser Prozess kann den Ladeprozess einer Website erheblich verlangsamen, vor allem, wenn viele DNS-Lookups notwendig sind, weil viele verschiedene Domains angefragt werden müssen.

Im Normalfall liegen die meisten Ressourcen wie Bilder, Stylesheets und Scripts auf der eigenen Domain. Werden aber z. B. Ressourcen von anderen Domains eingebunden, muss ein DNS-Lookup getätigt werden.

Beispiele:

» Für ein Facebook-Widget muss *facebook.de* nachgeschlagen werden
» Für einen Pinterest-Pin-Button muss pinterest.de nachgeschlagen werden
» Für ein Tracking-Pixel von Optimizly muss optimizly.com nachgeschlagen werden
» usw.

Es sollte genau aufgeschlüsselt werden, für welche Extra-Funktionen DNS-Lookups notwendig sind. Vor allem im „Above-the-Fold"-Bereich, also in dem oberen ersten Bereich der Seite, sollten keine zusätzlichen DNS-Lookups erforderlich sein. Am Ende sollte eine gute Balance zwischen PageSpeed und DNS-Lookups gefunden werden. Diese Entscheidungen sollten durch das Entwickler-Team und einige Tests begleitet werden.

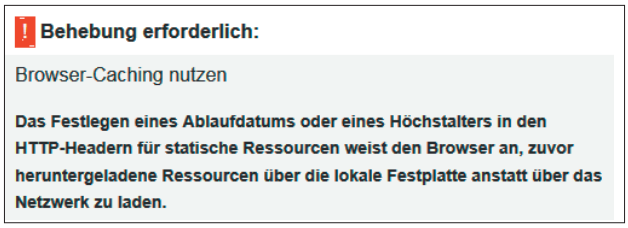

Abb. 15: Google PageSpeed Insights: Dateien ohne Ablauf-Zeit oder mit zu kurzer Cache-Dauer

Abb. 16: Google-Chrome-Developer-Tools › Netzwerk-Ansicht: Dieser Chache läuft nach 60 Tagen ab

Wie schnell antwortet der Server?

Der Browser fragt: „Hey Server, gib mir doch bitte beispiel.de/rote-schuhe.html."

Der Server antwortet: „Hey Browser, hier ich schicke dir beispiel.de/rote-schuhe.html."

Die Zeit, die für diese Kommunikation benötigt wird, bis das allererste Byte übertragen wird, nennt man „Time to First Byte" oder eben Serverantwortzeit. Im Normalfall sollte eine Dauer von 200 ms nicht überschritten werden, was auch der Richtwert aus den Google Guidelines besagt. Denn egal, wie gut eine Website in Bezug auf Performance optimiert ist, wenn die Serverantwortzeit sehr hoch ist, kann die Seite einfach keine schnelle Ladezeit mehr erreichen.

Die „Time to First Byte" ist von drei Komponenten abhängig. Um einer langen Serverantwortzeit auf den Grund zu gehen, sollte man also die folgenden Aspekte genauer betrachten:

» Wie ist die aktuelle Serverlast durch Traffic/Ressourcen?
» Wie ist die Server-Software aktuell konfiguriert?
» Wie gut sind das aktuelle Hosting und die Hardware?

Wie stark wird der Server ausgelastet?

Je nachdem, wie groß eine Seite ist oder wie aufwendig es ist, ihre Komponenten zusammenzustellen, z. B. durch viele Datenbankabfragen im Hintergrund, wird der Server unterschiedlich stark belastet. Dazu kommt, dass natürlich ein höheres Traffic-Aufkommen und somit mehr Serveranfragen den Server stärker auslasten.

Beispiel: Der „reddit hug of death" – wenn eine Seite auf Reddit so beliebt ist und so viel unerwarteten Traffic bekommt, dass irgendwann die Server schlappmachen.

Logischerweise ist die Steigerung des Traffics für jede Website eine wünschenswerte Entwicklung. Die Ressourcen optimal anzupassen, ist daher ein gutes Mittel, um höhere Traffic-Aufkommen gut abfedern zu können. Die Ressourcen zu verringern oder zu verkleinern, hilft demzufolge dem Server, mehr Nutzer parallel zu „bedienen" und gleichzeitig schneller zu antworten, wodurch die Seiten schneller geladen werden können.

Wie ist die Server-Software aktuell konfiguriert?

Möchte man nicht direkt in ein leistungsstärkeres Hosting investieren, sollte man sich damit befassen, wie man das aktuelle Hosting besser nutzen kann. Dafür sollte man sich mit den Server-Software-Konfigurationen genauer beschäftigen und diese so optimal wie möglich einstellen. Beispiele für Server-Konfigurationen sind Caching oder Keepalive, diese werden im Verlauf des Artikels genauer erläutert.

Reicht das Hosting noch aus?

Wenn sich die Auslastung des Servers trotz aller Optimierungsmaßnahmen nicht verändert, sollte man in eine Verbesserung des Hostings investieren. Oftmals erzielt man nach einer gewissen Zeit mehr Traffic als zuvor, sodass Anpassungen unumgänglich werden.

Caching

Wenn eine Seite aufgerufen wird, werden alle benötigten Elemente wie das Logo, die Stylesheets usw. geladen. Browser Caching bedeutet, dass sich der Browser Dateien „merkt" – d. h., wenn der Nutzer dann eine weitere URL der gleichen Domain aufruft, dann hat sich der Browser das Logo usw. bereits gemerkt, wenn sie auf dieser URL ebenfalls benötigt werden. Aus diesem Grund ist der erste Aufruf einer Webseite, der sogenannte First View, immer langsamer als der Repeat View.

Der Browser hält die bereits bekannten Dateien vor, sodass sie nicht neu vom Server geladen werden müssen. Das ist natürlich eine enorme Zeitersparnis. Wie lange der Browser die „bereits bekannten" bzw. „alten" Dateien vorhalten darf, kann man als Webseitenbetreiber selber festlegen. Man kann damit sozusagen bestimmen, wann der Browser sich die Dateien wieder frisch ziehen soll.

Für unterschiedliche Dateiformate sind unterschiedliche Cache-Zeiten sinnvoll und können auch individuell an die Entwicklungs-Zyklen der jeweiligen Website angepasst werden.

Beispiel: Bilder werden einmal hochgeladen und dann meist nicht geändert. Wenn sie ausgetauscht werden, dann handelt es sich eher um ein neues Bild mit einem neuen Namen. Das heißt, Bilder könnten theoretisch eine lange Cache-Dauer von einem Jahr erhalten.

Generell gilt: je länger, umso besser. Aber auch kurze Cache-Zeiten sind besser als kein Caching. Ein Monat ist generell ein guter Richtwert, wenn man gar keine Idee hat, was eine angemes-

sene Cache-Zeit sein könnte. Sollte eine Datei doch zeitnah geändert und frisch angefragt werden, kann man mit dem sogenannten URL-Fingerprinting (Datei-Versionierung) arbeiten.

Existieren Ressourcen ohne Ablauf-Zeit oder ist die Cache-Dauer zu kurz, meldet Google im PageSpeed-Insights-Tool den Fehler wie auf Abb. 15 zu sehen ist.

Die Ablauf-Zeit des Caches kann man ebenfalls in der Server-Antwort finden, sie wird in Sekunden angegeben (Abb. 16).

Wie alle Server-Einstellungen wird auch das Caching in der Server-Konfigurationsdatei oder in der .htaccess festgelegt, was mit Unterstützung eines Entwicklers gehandhabt werden sollte (siehe Code 1).

```
ExpiresActive On
ExpiresByType image/png „access 3 month“
ExpiresByType image/jpg „access 3 month“
ExpiresByType text/css „access 2 month“
ExpiresDefault „access 1 month“
```

Code 1: Beispiel-Code für .htaccess

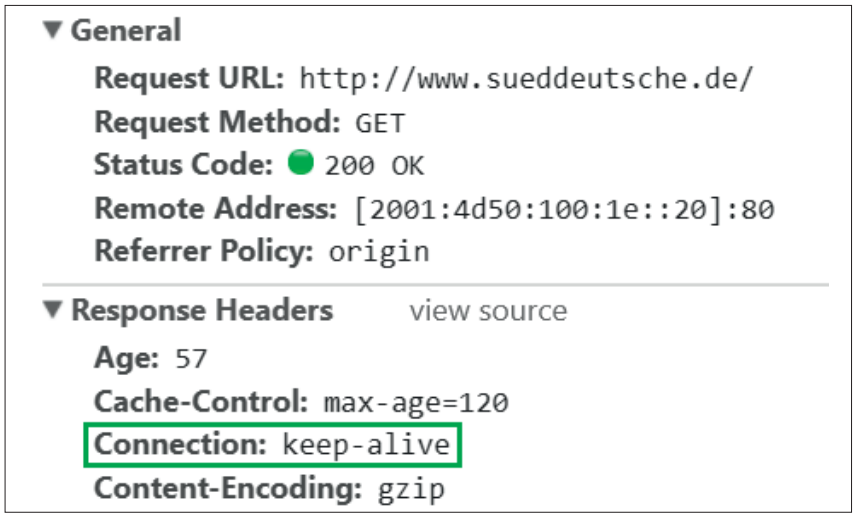

Abb. 17: Google-Chrome-Developer-Tools › Netzwerk-Ansicht

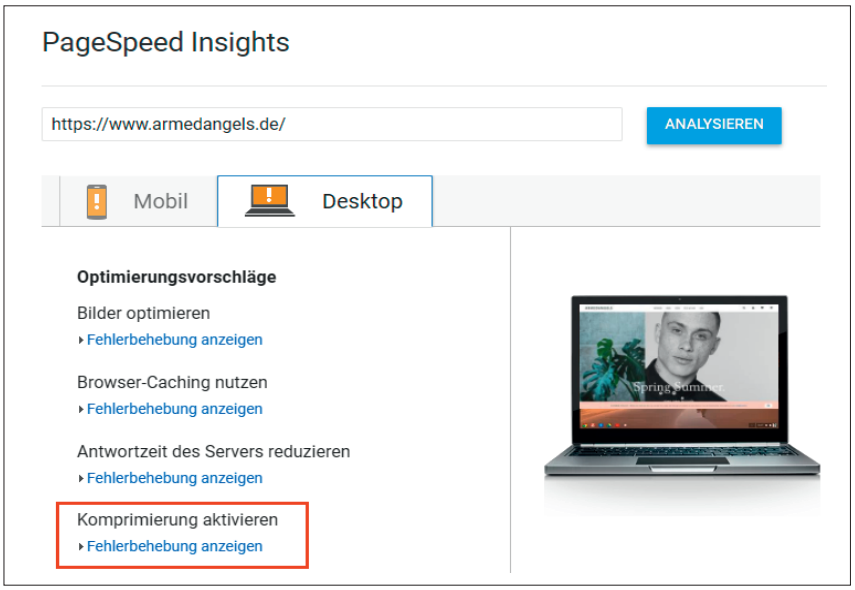

Abb. 18: Google PageSpeed Insights

Keepalive

„Keepalive" teilt dem Server mit, dass die Verbindung „am Leben" erhalten werden soll. Aus diesem Grund bezeichnet man sie auch als „dauerhafte oder persistente Verbindung". Es ist eine Methode, um die gleiche TCP-Verbindung für mehrere Anfragen zuzulassen, anstatt bei jeder Anfrage eine neue Verbindung zu öffnen. Dadurch können mehrere Dateien auf einmal übertragen werden, was einen wesentlichen Performance-Unterschied darstellt.

Zwar ist eine persistente Verbindung heutzutage eigentlich Standard. In Shared-Hosting-Umgebungen oder -Servern kann diese Einstellung jedoch ohne Wissen des Nutzers ausgeschaltet werden. Das wird häufig aus Performance-Gründen getan, wenn viele Seiten auf einer gemeinsamen Umgebung gehostet werden.

Ob Keepalive aktiviert ist oder nicht, findet man über die genannten Tools heraus oder über die Google-Chrome-Developer-Tool-Ansicht.

Sollte Keepalive nicht aktiv sein, kann dies innerhalb von wenigen Minuten behoben werden. Die Einstellung ist je nach Server anders und erfordert Zugriff auf die Server-Konfigurations-Datei. Sollte das nicht möglich sein, kann man die Einstellung auch in der .htaccess-Datei vornehmen.

Beispiel-Einstellung .htaccess-Datei

```
<ifModule mod_headers.c>
Header set Connection keep-
alive </ifModule>
```

Die Servereinstellungen für Gzip, Keepalive und das Caching sind absolute „Low Hanging Fruits", die man in kurzer Zeit anpassen kann. Die Umsetzung sollte man zusammen mit einem Entwickler oder ggf. auch mit dem Hosting-Anbieter in Angriff nehmen.

GZip

Gzip ist eine Methode, um Datenmengen zu verkleinern und sie somit schneller über ein Netzwerk zu übertragen. Gleichzeitig ist es auch ein Dateiformat. Metaphorisch könnte man sich einen Zip-Ordner vorstellen, der per E-Mail versendet wird. Durch Gzip-Komprimierung kann die Übertragungsmenge bis zu 90 % verkleinert werden, was die Ladezeit wesentlich verbessern kann. Die Aktivierung der Gzip-Komprimierung ist Standard. Wenn sie aus irgendeinem Grund nicht genutzt wird, ist die Webseite wahrscheinlich langsamer als die der Konkurrenz.

Um herauszufinden, ob es Ressourcen auf der eigenen Website gibt, die komprimiert übertragen werden könnten, kann man z. B. das PageSpeed-Insights-Tool von Google nutzen. Es meldet, dass man die „Komprimierung aktivieren" soll, und zeigt in der Fehlerbehebung die genauen Dateien an.

Die Einstellung kann in der Server-Konfigurationsdatei in wenigen Minuten vorgenommen werden und ist somit ein wesentlicher „Quick Win".

Beispiel-Einstellung Nginx-Server:

```
# Enable gzip compression.
# Default: off
gzip on;
```

Es können alle textbasierten Ressourcen komprimiert übertragen werden, z. B.:

- text/html
- text/css
- text/javascript
- application/javascript
- image/svg+xml

Fazit

Der Google-Algorithmus entwickelt sich stetig weiter. Eine professionelle Herangehensweise zeichnet sich dadurch aus, dem Algorithmus immer ein Stückchen voraus zu sein, sodass man bei Änderungen nicht stark betroffen wird. Dass Google moderne Webseiten schätzt und auch PageSpeed aktuell eine große Rolle spielt, merkt man an verschiedenen Aussagen, all ihren Ressourcen, die sie zu diesem Thema bereitstellen, bis hin zu eigenen Tools, die zur Evaluierung der Performance entwickelt wurden. Außerdem hat der PageSpeed eine Auswirkung auf das Nutzerverhalten, die Conversions und somit auch direkt auf den Umsatz. Ein Handeln im eigenen Interesse und im Interesse von Google ist auf jeden Fall sehr sinnvoll. Aus diesem Grund sollte man das Thema mobiler PageSpeed nicht verschlafen und seine Nutzer glücklich machen. Denn jeder liebt schnelle Webseiten (vor allen in der U-Bahn & Co.). ¶