

## Bachelor-Thesis

# Web Performance für den mobilen Endanwender



zur Erlangung des akademischen Grades  
Bachelor of Science

vorgelegt von:

Andreas Lorer

Wilhelmstraße 4  
88250 Weingarten  
15. April 2015

1. Gutachter: Prof. Klemens Ehret
2. Gutachter: Lukas Michna

---

## Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben. Die Arbeit wurde bisher, in gleicher oder ähnlicher Form, keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

---

Unterschrift

---

Ort, Datum

---

# Vorwort

## Zusammenfassung

Zeit wird als ein kostbares und begrenztes Gut empfunden, weswegen wir in sämtlichen Lebensbereichen auf deren Ersparnis und Nutzungsoptimierung bedacht sind. Dies gilt auch für aktivitäten im modernen Web. Allerdings laden auch im Jahr 2015 Webanwendungen und Webseiten immer noch sehr langsam. Diese Arbeit befasst sich mit den Ursachen und Gründen für diese Problematik und soll Lösungsstrategien aufzeigen, die dem Smartphone Anwender eine schnellere Online Erfahrung ermöglichen. Dafür werden Grundlagen erklärt, **Best Practices** erläutert und beschrieben, wie der Weg zur performanten Webanwendung aussehen kann.

Diese Methodiken zur Verbesserung der Performance wurden im Rahmen eines eigenen Projekts umgesetzt und erprobt. Dabei wurde ein automatisiertes Testverfahren verwendet und Daten wie Load Time, Speed Index, Render Start, Time to Visually Complete und andere Metriken über den Projektzeitlauf erfasst. Bei der Datenauswertung ergab sich eine signifikante Verringerung dieser Werte und damit verbunden eine drastische Reduzierung der Seitenladezeit von 7,6 auf unter 1,5 Sekunden.

Die hier beschriebenen Methodiken erlauben eine umfassende Steigerung der Nutzererfahrung für Webseiten im Internet. Die Arbeit schließt mit der Erkenntnis, dass die Voraussetzung für Web Performance weniger die Programmierung ist, sondern in erster Linie eine Veränderung der Unternehmenskultur stattfinden muss, um Web Performance richtig umsetzen zu können.

## Danksagung

Mein besonderer Dank für das Korrigieren und der inhaltlichen Überprüfung geht an:

Christina Negele

Michael Lorer

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Zielsetzung . . . . .	2
1.3 Eigene Leistung . . . . .	3
1.4 Ist-Zustand . . . . .	3
<b>2 Begriffe</b>	<b>4</b>
2.1 Pattern und Antipattern . . . . .	4
2.2 Content Delivery Network (CDN) . . . . .	4
2.3 Latenz . . . . .	5
2.4 Round Trip Time (RTT) . . . . .	5
2.5 Http/1.1 Protokoll . . . . .	5
2.6 TCP Three Way Handshake . . . . .	5
2.7 TCP Slow Start . . . . .	6
2.8 Above The Fold . . . . .	7
2.9 Perceived Performance . . . . .	8
<b>3 Die 1000 ms Barriere</b>	<b>9</b>
3.1 Touch Event . . . . .	9
3.2 Netzwerke . . . . .	10
3.2.1 Mobilfunknetz . . . . .	10
3.3 Der HTTP-Request . . . . .	11
3.4 Das Herunterladen einer 40 Kilobyte Datei . . . . .	12
3.5 Zusammengefasst . . . . .	13
3.6 Kritischer Rendering-Pfad . . . . .	17
3.6.1 Rendering . . . . .	17
3.6.2 Rendering-Pfad . . . . .	17
3.6.3 Critical Render Path . . . . .	18
3.6.4 Zusammengefasst . . . . .	19
3.7 Analyse des Wasserfalls . . . . .	20
<b>4 Entwicklung</b>	<b>23</b>
4.1 Tools . . . . .	23
4.1.1 Google Chrome Developer Tool . . . . .	23
4.1.2 Google Pagespeed Insight . . . . .	23
4.1.3 Google Closure Compiler . . . . .	24
4.1.4 Webpagetest . . . . .	24
4.1.5 Pingdom . . . . .	26
4.1.6 Speedcurve . . . . .	26
4.1.7 Google Spreadsheet . . . . .	27
4.1.8 Feed the Bot . . . . .	27
4.1.9 What Does My Site Cost? . . . . .	27
4.1.10 Critical Path CSS Generator . . . . .	28
4.1.11 Http Archive . . . . .	28
4.1.12 Perf Tooling Today . . . . .	28

4.1.13 Twitter . . . . .	28
4.2 Ausgangspunkt . . . . .	29
4.3 Best Practices . . . . .	31
4.3.1 Render Blocking Javascript / CSS . . . . .	31
4.3.2 CSS-Bereitstellung optimieren . . . . .	33
4.3.3 Ressourcen reduzieren . . . . .	33
4.3.4 Antwortzeit des Servers reduzieren . . . . .	34
4.3.5 Browser-Caching nutzen . . . . .	34
4.3.6 Komprimierung aktivieren . . . . .	38
4.3.7 „Keep-alive“ ermöglichen . . . . .	40
4.3.8 HTML5 Link Prefetching . . . . .	40
4.3.9 Domain Sharding . . . . .	40
4.4 Bilder optimieren . . . . .	41
4.4.1 Progressive Image Rendering . . . . .	41
4.4.2 Image Spriting . . . . .	42
4.4.3 Bild Komprimierung . . . . .	43
4.4.4 Responsive Images . . . . .	43
4.4.5 Adaptive Images . . . . .	45
4.5 Zusammengefasst . . . . .	45
<b>5 Workflow</b>	<b>46</b>
5.1 Nodejs & Node Package Manager . . . . .	46
5.1.1 Dependency Management . . . . .	47
5.1.2 Bower . . . . .	48
5.2 Gulp Task Manager . . . . .	49
5.3 Yeoman . . . . .	51
5.3.1 Eigene Generatoren erstellen . . . . .	53
5.4 Zusammengefasst . . . . .	54
<b>6 Ergebnis</b>	<b>56</b>
6.1 Wie wurde getestet? . . . . .	56
6.2 Datenauswertung . . . . .	57
<b>7 Der Weg zur Performance</b>	<b>61</b>
7.1 Hürden . . . . .	61
7.1.1 Projekt Manager . . . . .	62
7.1.2 Ästhetik anspruchsvolle Designer . . . . .	63
7.2 Performance Budget . . . . .	65
7.2.1 Budget Metriken . . . . .	65
7.2.2 Wie schnell, ist schnell genug? . . . . .	66
7.2.3 Arbeiten mit einem Performance Budget . . . . .	66
<b>8 Ausblick</b>	<b>69</b>
8.1 Http/2.0 . . . . .	69
8.1.1 Http/1.1 Optimierungen in Http/2 . . . . .	70
<b>9 Fazit</b>	<b>71</b>

<b>10 Anhang</b>	<b>72</b>
10.1 Webpagetest Teststandorte . . . . .	72
10.2 Argumentations Sammlung . . . . .	73
10.3 In Zusammenhang stehende Arbeiten . . . . .	75

# 1 Einleitung

Diese Arbeit beschäftigt sich umfassend mit dem Thema Web Performance unter dem Aspekt, dass immer mehr Anwender mittels Smartphone mit dem Web agieren. Sie soll die Frage klären, warum Webanwendungen auf einem Smartphone oftmals um ein Vielfaches langsamer Laden als via Desktop-PC, welche Auswirkungen sich daraus ergeben und welche Maßnahmen zur Optimierung ergriffen werden können. Daraus resultiert ein **Workflow**, der sich nach den „Best Practices“ richtet. Die Arbeit endet mit den kulturellen Veränderungen, die das Thema Web Performance für ein Unternehmen mit sich bringen kann und welche Herausforderungen es zu meistern gilt.

## 1.1 Motivation

Niemand mag es zu warten. Sei es auf Bus, Bahn oder an der Kasse im Supermarkt. Wir warten auch nicht gerne im Internet auf das Laden eines Videos, beim Besuch einer Webseite oder beim Einkaufen via App. Zu oft wird aus dem „nur mal eben diesen Begriff nachschlagen“ ein endloses Starren auf den weißen Bildschirm. Jeder kennt das.

Larry Page, CEO und Mitgründer von Google, sagt:

*„As a product manager you should know that speed is the number one feature.“* (Holzle 2010)

Niemand mag es zu warten, auch nicht auf eine Webanwendung. Die Studie „The Psychology of Web Performance“ zitierte bereits im Jahr 2008 folgende Ergebnisse:

*„Slow web pages lower perceived credibility and quality. Keep your page load times below tolerable attention thresholds, and users will experience less frustration, lower blood pressure, deeper flow states, higher conversion rates, and lower bailout rates. Faster websites are actually perceived to be more interesting and attractive.“* (WebSiteOptimization.com 2008)

Das Hauptvermarktungsargument für den Chrome Browser war damals, er sei schneller als die Konkurrenz. Tatsächlich ist für Google Geschwindigkeit alles. Deshalb hat Google im Jahr 2010 angekündigt, dass Geschwindigkeit in die Berechnung des **Google Page Rankings** mit einfließt.

*„Faster sites create happy users and we've seen in our internal studies that when a site responds slowly, visitors spend less time there. [...] Recent data shows that improving site speed also reduces operating costs. Like us, our users place a lot of value in speed — that's why we've decided to take site speed into account in our search rankings“* (Google 2010)

Aktuell (2015) geht Google sogar noch einen Schritt weiter und informiert tausende Webmaster per E-Mail über die schlechte Usability ihrer Websites für mobile Besucher und warnt ausdrücklich vor dementsprechend „angepassten Rankings“. (t3n 2015) Im Hinblick auf die Zukunft wird der Marktanteil an mobilen Internetnutzern noch weiter wachsen und die Optimierung der Ladezeiten gewinnt dadurch noch mehr an Bedeutung. Zwischen

2011 und 2014 stieg die Anzahl der Smartphonenuutzer von 18% auf 50% an. Dies ist ein Wachstum von 32% innerhalb von nur 3 Jahren.(TNS Infratest 2014)

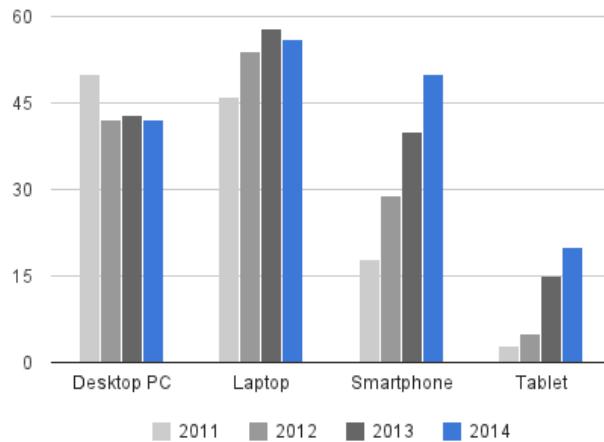


Abbildung 1: Gerätenutzung in der Gesamtbevölkerung (2011 – 2014)(TNS Infratest 2014)

Die Antwort auf diesen Trend läutete eine Ära ein, die wir heute unter dem Namen **Responsive Webdesign** kennen. „Responsive“ muss aber sehr viel mehr bedeuten, als nur eine angepasste Darstellung für eine bestimmte Art von Gerät. „*Two out of three mobile shoppers expect pages to load in 4 seconds or less.*“ (Radware 2013). Der Anwender erwartet also auf dem Smartphone ähnliche oder gleiche Ladezeiten wie er sie auch von der Nutzung des Desktop PC gewohnt ist. Diese Erwartungen werden von dem Großteil der Internetseiten nicht erfüllt. Der Inhalt einer Seite muss darum so aufbereitet werden, dass dieser auch auf Geräten mit langsamer Internetverbindung, hoher Latenz und einem begrenzten Datentarif, in einer für den Anwender annehmbaren Geschwindigkeit angezeigt werden kann.

## 1.2 Zielsetzung

Um gängige Methoden und Techniken der Ladezeit-Optimierung anzuwenden, wird das Projekt anhand der Website <http://andreaslorer.de/> durchgeführt. Das Ziel ist es, die Ladezeit der Website auf dem Smartphone, wie auch auf dem Desktop auf unter 1000 Millisekunden zu verringern. Mit Ladezeit ist dabei nicht die Zeit gemeint die benötigt wird um die Website komplett zu laden, sondern die Zeit bis eine erste visuelle Rückmeldung für den Anwender zu sehen ist. Diese, vom Anwender wahrgenommene Rückmeldung, nennt man auch „Perceived Performance“ und bedeutet, dass die Ladezeit als schneller empfunden wird, als es eigentlich laut Messwerten der Fall ist. Näheres dazu wird in Kapitel 2.9 beschrieben.

### 1.3 Eigene Leistung

Die Leistung besteht darin, einen Gesamtüberblick über die heutigen Best Practices zu geben. Die Arbeit soll dem Leser ein Gespür für Fehler in der Struktur von Webanwendungen geben, die für die Geschwindigkeit hinderlich sind. Es soll herausgefunden werden, was getan werden muss um die Ladezeit zu minimieren, wie ein moderner „Workflow“ aussehen kann, damit eine Webanwendung schon bei seiner Entstehung schnell lädt und im Projektverlauf schnell bleibt. Des Weiteren soll erklärt werden, welche Herausforderungen es zu meistern gilt um eine schnelle Webanwendung zu erreichen, welche Tools es gibt und welche Vor- oder Nachteile diese mit sich bringen.

Diese Arbeit befasst sich nicht mit der Geschwindigkeit von Datenbanken, SQL-Abfragen oder sonstigen Problemen, die durch einen Engpass ein schnelles Laden der Seite verhindern könnten.

### 1.4 Ist-Zustand

Die Webseite [www.andreaslorer.de](http://www.andreaslorer.de) ist auf einem Shared Hosting<sup>1</sup> aufgesetzt und die Antwortzeit des Servers beträgt < 200 Millisekunden. Dadurch, dass es keine Möglichkeit gibt Root Rechte<sup>2</sup> auf einem Shared Hosting zu bekommen, können so manche serverseitige Einstellungen nicht durchgeführt werden. Diese werden dann zwar aufgezeigt, kommen aber für dieses Projekt nicht zum Einsatz.

Die Website hat als Ausgangsbasis einen einfachen Aufbau. Sie besteht aus einer Bildergallerie basierend auf PHP und dem Bootstrap Framework.

---

<sup>1</sup>Bei Shared Hosting werden mehrere Websites von verschiedenen Website-Betreibern von dem gleichen Webserver gehostet. Bei Shared Hosting teilen sich in der Regel Hunderte andere Websites einen Server (ItWissen.info 2015)

<sup>2</sup>Standardmäßig existiert unter Linux immer ein Konto für den Benutzer „root“ mit der User-ID 0. Dies ist ein Systemaccount mit vollem Zugriff auf das gesamte System, und damit auch auf alle Dateien und Einstellungen aller Benutzer (wiki.ubuntuusers 2014)

## 2 Begriffe

### 2.1 Pattern und Antipattern

„Entwurfsmuster (englisch *design patterns*) sind bewährte Lösungsschablonen für wiederkehrende Entwurfsprobleme sowohl in der Architektur als auch in der Softwarearchitektur und -entwicklung. Sie stellen damit eine wiederverwendbare Vorlage zur Problemlösung dar, die in einem bestimmten Zusammenhang einsetzbar ist.“ (Wikipedia 2015)

„Während „Design Patterns“ in der Software-Entwicklung allgemein übliche und bekannte Ansätze sind, um Probleme zu lösen, sind Anti-Patterns Negativ-Beispiele – die zeigen, wie man es nicht macht – von bereits durchgeführten, gescheiterten Projekten, die dem erkennenden Mitarbeiter zielgerichtete Hinweise darauf geben, wie die Aufgabenstellung besser gelöst werden könnte. Als Synonym ist auch der Begriff Negativmuster im Gebrauch. Es ist tatsächlich möglich, daß das, was gestern noch als allgemein gangbarer Lösungsweg bezeichnet wurde, heute schon ein „Antipattern“ ist [...]“ (Stepken 2006)

### 2.2 Content Delivery Network (CDN)

Ein Content Delivery Network (CDN) oder auch Content Distribution Network genannt, ist ein Netz regional verteilter und über das Internet verbundener Server, mit dem Inhalte ausgeliefert werden. CDN-Knoten sind auf viele Orte verteilt, um Anfragen (Requests) von End-Nutzern nach Inhalten (Content) möglichst ökonomisch zu bedienen. Im Hintergrund (transparent) werden die Daten im Netz so vorgehalten (Caching), dass die jeweilige Auslieferung entweder möglichst schnell geht (Performance-Optimierung) oder möglichst wenig Bandbreite verbraucht (Kosten-Optimierung), oder beides zugleich. Durch eine nähere Platzierung des Servers zum Endverbraucher, verringert sich beispielsweise die Latenzzeit. Vor allem für globale Inhaltsangebote ist dadurch die Nutzung eines CDN von Vorteil. Große CDNs unterhalten tausende Knoten mit zehntausenden Servern. (Wikipedia 2015a, vgl.)



Abbildung 2: Schematische Darstellung eines CDN (Eigene Abbildung, Karte nach (Ritz 2014))

## 2.3 Latenz

Latenz bezeichnet die Verzögerung bis ein Paket von Sender A zu Empfänger B gelangt ist.

## 2.4 Round Trip Time (RTT)

„Round Trip Time“ wird im Deutschen „Paketumlaufzeit“ genannt. Es bezeichnet die Zeit die ein Datenpaket braucht um in einem Netzwerk von Sender A zu Empfänger B und wieder zurück zu gelangen. Bei einer Latenz von 100 ms würde die RTT folglich 200 ms betragen (Annahme: Hin- und Rückweg haben dieselbe Zeit). Wie später noch zu sehen ist, spielt die RTT für die Web Performance eine maßgebliche Rolle.

## 2.5 Http/1.1 Protokoll

Der für diese Arbeit wichtige Aspekt der HTTP/1.1 Spezifikation ist die Limitierung von Verbindungen pro Domain Name. Dabei weicht die Limitierung zwischen den Browsern ab und reicht von 6 (Google Chrome) bis 13 (Internet Explorer 11) parallelen TCP Verbindungen.<sup>3</sup>

## 2.6 TCP Three Way Handshake

TCP ist das meistgenutzte Verbindungsprotokoll im Internet. Auf diesem Protokoll wird der HTTP Request aufgebaut, der die eigentlichen Daten enthält. Bevor diese Daten zwischen Server und Browser ausgetauscht werden können, muss eine Verbindung aufgebaut werden. Abbildung 3 beschreibt den Prozess des Verbindungsauftausbs.<sup>4</sup> Dabei werden sogenannte SYN- und ACK-„Flags“ zwischen Server und Browser versendet.

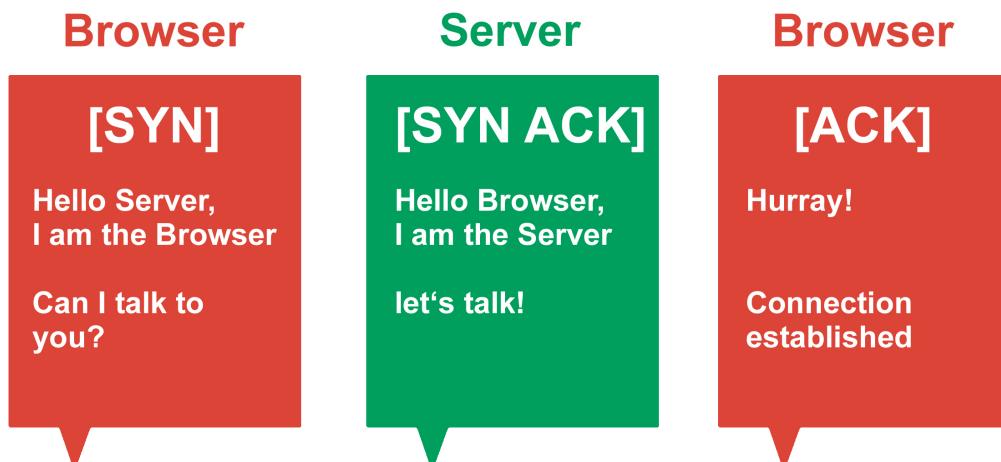


Abbildung 3: Three-Way-Handshake zum Aufbau einer TCP Verbindung zwischen Browser und Server (Eigene Abbildung)

---

<sup>3</sup>Eine Übersichtsliste ist hier zu finden <http://www.browserscope.org/?category=network>.

<sup>4</sup>Für ein tieferes Verständnis empfiehlt sich dieser Artikel: High Performance Browser Networking - Chapter 2: Building Blocks of TCP: <http://tinyurl.com/1jna35v>

## 2.7 TCP Slow Start

Ein Round Trip kann nicht beliebig viele Bytes transportieren, sondern ist durch die sogenannte „Congestion<sup>5</sup> Window Size“ limitiert. Der Überbegriff für dieses Verhalten nennt sich „Slow Start“

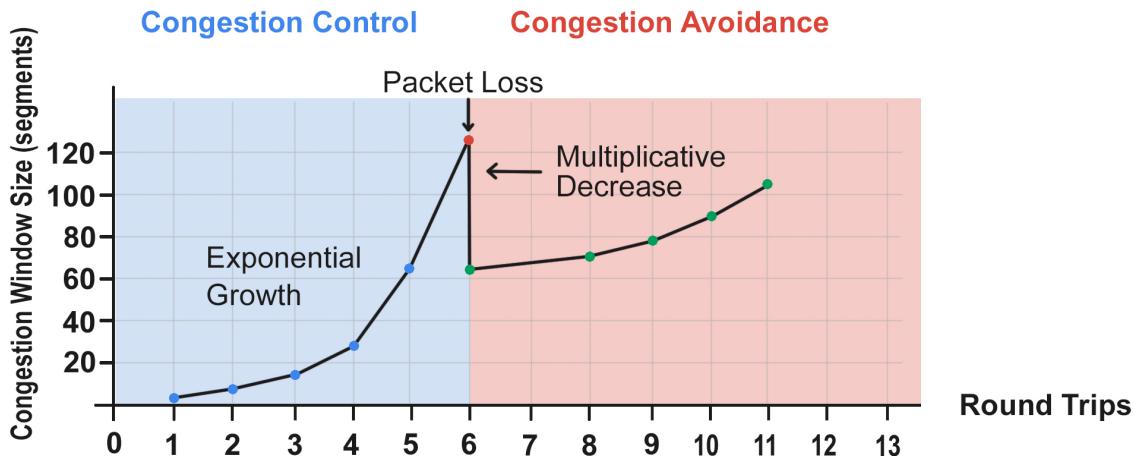


Abbildung 4: Congestion Control und Congestion Avoidance (Eigene Abbildung nach (Grigorik 2013c))

- Congestion Control: Nachdem eine neue Verbindung per TCP aufgebaut wurde, können weder Server noch Client wissen wie schnell die verfügbare Bandbreite ist, mit der Daten ausgetauscht werden können. Um das Netzwerk vor einem Datenstau zu schützen, wird mit einem sehr niedrigen Wert begonnen, der dann ansteigt bis das Limit erreicht ist. Dieses Verhalten nennt sich auch „Congestion Control“.
- Congestion Window Size: Diese Größe bestimmt, wieviel Bytes pro Segment gesendet werden dürfen, bis diese vom Empfänger per ACK (acknowledgement) bestätigt werden müssen. Die Größe der Segmente ist standardmäßig 1460 Bytes und die Rate bis zum ACK ist im April 2013 von 4 auf 10 Segmente erhöht worden.(Grigorik 2013c). In der Grafik wird noch von dem alten Wert mit nur 4 Segmenten pro Round Trip ausgegangen. Die Datenrate wächst exponentiell an, damit möglichst schnell die volle Bandbreite nutzbar ist.
- Congestion Avoidance bedeutet, dass sich die Datenrate wieder um ein Vielfaches verringert, falls es zu einem Paketverlust kommt. Da es besonders bei WLAN oder Mobilfunknetzen des Öfteren zu Paketverlusten kommen kann, ist dieser Aspekt besonders hervorzuheben, denn er verzögert das Erreichen der maximal möglichen Datenrate.

<sup>5</sup>engl. congestion: Stauung, Überlastung, Anhäufung

Slow Start bedeutet also aus Sicht der Performance, dass bei einer neuen TCP Verbindung nicht die maximale Bandbreite zu Verfügung steht. Bei größeren Dateien wird zwar durch das exponentielle Wachstum das Maximum schnell erreicht, gerade aber bei kleineren Dateien mit wenigen Kilobyte ist dies oft nicht der Fall.

## 2.8 Above The Fold

Damit ist der auf einem Bildschirm sichtbare Bereich vor dem Scrollen gemeint. Diesem Bereich wird eine besondere Wichtigkeit zugesprochen.



Abbildung 5: Darstellung des sichtbaren Bereichs vor dem Scrollen

*„In an analysis of 57,453 eyetracking fixations, we found that there was a dramatic drop-off in user attention at the position of the page fold. Elements above the fold were seen more than elements below the fold: the 100 pixels just above the fold were viewed 102% more than the 100 pixels just below the fold.“*  
(Schade 2015)

Wichtige Informationen oder Navigationselemente sind meistens dort zu finden. Eine Webseite, die nach dem Paradigma des Responsive-Webdesign aufgebaut ist, kann dabei 3 oder mehrere Ansichten haben die alle einen unterschiedlichen „above the fold“ Bereich haben. Eine Anwendung kann aber auch unterschiedliche Seiten haben, auf dem der Anwender landen kann. Zum Beispiel wenn dieser an- oder abgemeldet ist. Paradebeispiel dafür sind Facebook oder Twitter.

Innerhalb des **above the fold** ist es wichtig, den Inhalt für den Seitenbesucher richtig zu strukturieren. Dafür ist es nötig zu wissen, welcher Inhalt denn für den Anwender am bedeutungsvollsten ist. Bei einer Online Zeitung mag das die neuste Schlagzeile sein, bei einem Portal wie Facebook die Timeline. Da das HTML Dokument vom Browser von oben nach unten gelesen wird, macht es folglich Sinn, die wichtigen Elemente vor den unwichtigen zu platzieren. Damit werden diese zuerst verarbeitet und können damit früher angezeigt werden.

## 2.9 Perceived Performance

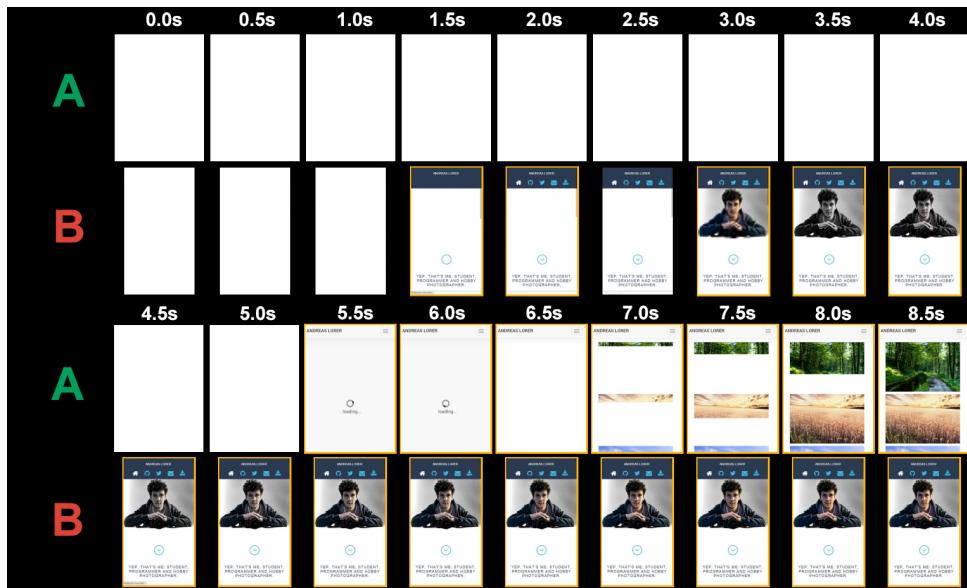


Abbildung 6: Zwei Seiten im Vergleich (Eigene Abbildung via webpagetest.org)

Abbildung 6 zeigt die Seiten A und B, mit nahezu identischer Ladezeit. Der Unterschied besteht darin, dass Seite B bereits nach 1.5 Sekunden dem Anwender eine erste visuelle Rückmeldung gibt, wohingegen Seite A ganze 5.5 Sekunden dafür benötigt. „Perceived Performance“ steht also für die Zeit bis ein erste visuelle Rückmeldung für den Anwender zu sehen ist und bedeutet, dass die Ladezeit als schneller empfunden wird, als es eigentlich laut Messwerten der Fall ist. Warum diese „Perceived Performance“ für eine Webanwendung so wichtig ist zeigen mehrere Studien, deren Daten in folgender Grafik aufbereitet sind.



Abbildung 7: Einfluss und Effekt einer langsamen Seite auf den Anwender (Eigene Abbildung nach Daten von: (Radware 2014, p. 8))

Bereits kleine Verbesser- oder Verschlechterungen der Ladezeit können einen großen Einfluss auf den Anwender haben. Yahoo hat herausgefunden, dass eine Seite, die um nur 400 Millisekunden schneller lädt, den Traffic um 9% erhöhte.(Stefanov 2008) 57% der Online Konsumenten haben eine Seite die länger als 3 Sekunden lädt bereits wieder verlassen. 78% der Anwender empfinden sogar Zorn oder Stress wenn eine Seite nicht lädt oder der Vorgang nicht ersichtlich ist. Perceived Performance ist also wichtiger als die Ladezeit selbst und Maßnahmen die sie verbessern sind besonders wichtig.

### 3 Die 1000 ms Barriere

Das Ziel dieser Arbeit die 1000 Millisekunden Barriere zu durchbrechen wurde nicht durch Zufall gewählt. Der Anwender nimmt die Geschwindigkeit einer Seite subjektiv wahr. Sie wird in der folgenden Grafik interpretiert:

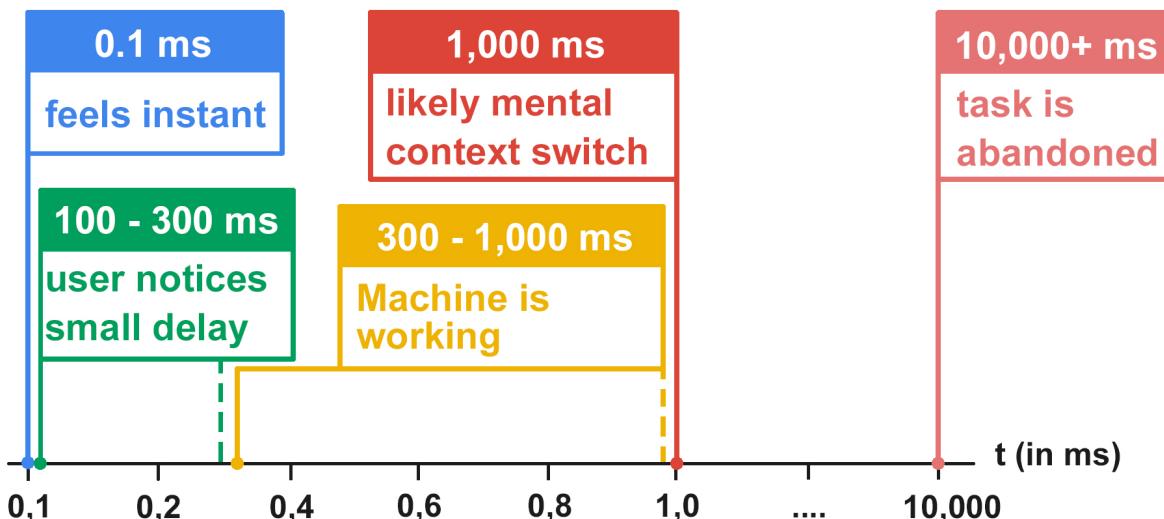


Abbildung 8: Zeit und Wahrnehmung durch den Anwender (Eigene Abbildung nach Daten von: (Grigorik 2013d))

Wie zu sehen ist, bleibt gerade einmal eine Sekunde Zeit, bevor das Gehirn uns sagt man solle doch einer anderen Aufgabe nachgehen, bis der Ladevorgang abgeschlossen ist. Der Anwender verlangt visuelle Rückmeldung, um „am Ball zu bleiben“. Auf vielen Webseiten sieht man deshalb Ladebalken oder sogenannte **Spinner**, die erkennbar machen, dass der Ladevorgang im Gange, aber noch nicht abgeschlossen ist.

Um das Ziel von einer einsekündigen Ladezeit bis zum ersten Render zu erreichen, ist es nötig zu verstehen an welcher Stelle die meiste Zeit verbracht wird. Bevor eine Seite mittels Smartphone vom Browser dargestellt wird, laufen eine ganze Reihe von Prozessen ab, die im Weiteren erklärt werden.

#### 3.1 Touch Event

Der Aufruf einer Seite über das Smartphone erfolgt über ein Touch Event auf einen Link, Button oder die Seite wird per URL aufgerufen. Hierbei können je nach Gerät zwischen 50 (iPhone 5) und 120 Millisekunden (Moto X - Android) zwischen der Berührung des Touch Screen und dem Registrieren des Events vergehen.(Takahashi 2013) Der Browser wartet allerdings nochmals bis zu 300 Millisekunden, denn er muss abwarten ob vielleicht noch ein zweiter Finger aufgelegt wird (Multitouch), oder ob der Anwender Scrollen oder Zoomen möchte.(Google 2011)

Dieses Verhalten lässt sich bei vielen Browern per Meta Tag abstellen:

```
1 <meta name="viewport" content="user-scalable=no">
```

Dies setzt natürlich voraus, dass die Webanwendung kein Zoomen benötigt, um sie zu bedienen. Gerade bei älteren Webseiten trifft das oftmals nicht zu, da sie keine für das Smartphone angepasste Ansicht haben (Responsive View). Eine vollständige Liste mit Meta Tags für die verschiedenen Browser ist der Fußnote zu entnehmen.<sup>6</sup>

## 3.2 Netzwerke

Warum gerade das Nutzen des Internets per Smartphone so langsam sein kann (und oftmals ist), liegt zu einem Großteil am Netzwerk. Eine Studie untersuchte die Top eine Millionen Webseiten des Internets auf ihre Ladezeiten. Dabei wurde eine Verbindung von 5 Mbit/s und 28ms RTT benutzt. Eine RTT von 28 ms ist sehr schnell, verglichen mit der Latenz des 3G Netzes (100 - 500 ms RTT: Abbildung 10). Diese Studie kam zu dem Ergebnis, dass fast 70% der Ladezeit nur durch Warten auf das Netzwerk verbracht wird:

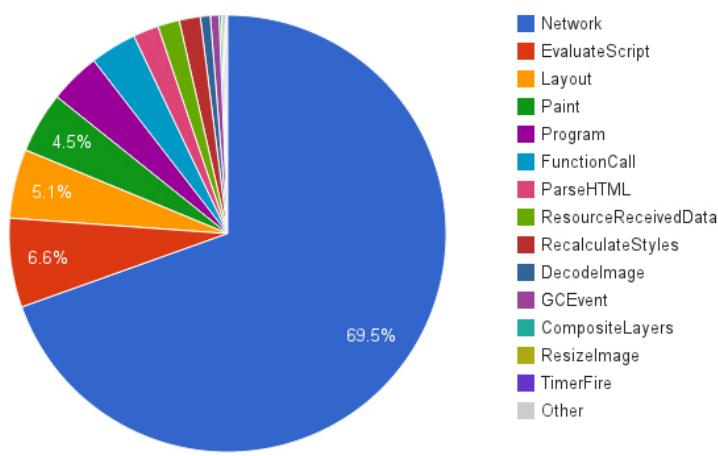


Abbildung 9: Untersuchung der Top 1 Millionen Alexa Seiten (Abbildung von: (Tonyg 2013))

Es macht also durchaus Sinn sich diesen Bereich näher anzusehen, um zu verstehen worauf Einfluss genommen werden kann und wo nicht.

### 3.2.1 Mobilfunknetz

Es gibt unterschiedliche Mobilfunkstandards, mit denen Anwender Zugriff auf das Internet erlangen. Selbst wenn einem Anwender 4G vom Mobilfunkanbieter versprochen werden, so ist die Netzabdeckung mit 4G noch nicht vollständig.

*„Bereits Mitte 2013 konnten laut dem Breitbandatlas der Bundesregierung 70 Prozent der deutschen Haushalte über LTE verfügen. E-Plus startete mit LTE im März 2014.“ (Bundesnetzagentur 2014)*

Das bedeutet, dass der 4G Anwender auf ein niedrigeres Netz wie zum Beispiel 3G ausweichen muss. Die verschiedenen Netzwerke unterscheiden sich entscheidend in ihrer Datenrate und vor allem in der Latenz. Die Tabelle in Abbildung 10 gibt eine Übersicht.

---

<sup>6</sup>Suppressing 300ms delay for touchscreen interactions: <http://tinyurl.com/psj5nxz>

Gernation	Data rate	Latency
2G	100 - 400 Kbit/s	300 - 1000 ms
3G	0,5 - 5 Mbit/s	100 - 500 ms
4G	1-50 Mbit/s	< 100 ms

Abbildung 10: Datenrate und Latenz (Eigene Abbildung nach (Grigorik 2013e))

Unser Smartphone ist nicht ständig mit dem „wireless service provider“ verbunden. Ist eine erste Verbindung nötig, so muss das Smartphone dem Sendeturm mitteilen, dass es Kommunizieren möchte. Der Anbieter muss die Anfrage authentifizieren, die Verbindung herstellen und dann die Anfrage in das Internet weiterleiten. Bis eine Authentifizierung erfolgt ist, kann je nach Anbieter und Mobilfunkstandard zwischen <100ms (LTE) und 2,5 Sekunden (3G) vergehen (Grigorik 2013a). Bereits hier ist zu sehen, dass es „worst case“ Szenarien gibt durch die es nicht möglich sein kann, dass eine Webanwendung in unter einer Sekunde eine Rückmeldung gibt. Gerade Mobilfunknetze unterliegen Stoßzeiten, die Funksignale von Smartphones können sich gegenseitig stören oder das Signal kann in gewissen Gegenden stärker oder schwächer sein.

### 3.3 Der HTTP-Request

Nachdem unser Mobilfunkanbieter uns mit dem Internet verbunden hat, kann die eigentliche Anfrage an den Server gestellt werden.

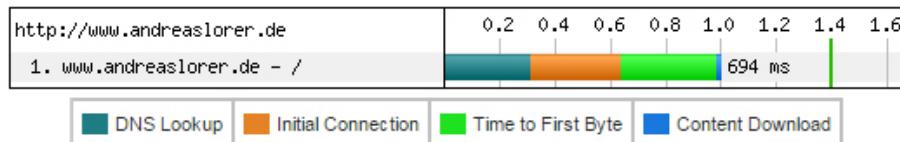


Abbildung 11: Anfrage der HTML-Datei von Irland aus (<http://webpagetest.org>)

- DNS Lookup: Um eine Verbindung mit dem Server herzustellen, benötigt das HTTP Protokoll die IP Adresse des Ziels. Dafür wird ein DNS Server damit beauftragt für den Namen „<http://andreaslorer.de>“ die zugehörige IP Adresse zurück zu geben.
- Initial Connection bezeichnet die Zeit die vergeht, bis eine neue Verbindung zum Server hergestellt wurde damit eine Kommunikation zwischen Browser und Server möglich ist. Hierbei findet der sogenannte TCP „Three-Way-Handshake“ statt, der dafür einen Round Trip benötigt.
- TTFB: Ist die Abkürzung für „Time To First Byte“. Dieser Begriff beschreibt die Zeit die vergeht bis das erste Byte vom Server den Browser erreicht. Der Server muss dabei die Antwort erst zusammenstellen, bevor er sie versenden kann. Dafür werden

unter Umständen Ressourcen aus der Datenbank abgefragt oder es müssen Berechnungen stattfinden. Diese Faktoren beeinflussen die TTFB und können optimiert werden (schnellerer Server, bessere Datenbankanbindung, Caching).

- Content Download: Die Zeit die benötigt wird, bis die Datei vom Server heruntergeladen wurde.
- Nachdem das HTML Dokument heruntergeladen wurde muss es vom Browser noch gelesen und interpretiert werden. Diese Zeit taucht im Diagramm nicht auf.

### 3.4 Das Herunterladen einer 40 Kilobyte Datei

Abbildung 12 zeigt schematisch wie eine 40kb Datei mittels einer neuen TCP Verbindung heruntergeladen wird. Sie soll verdeutlichen, wie vor allem die RTT eine entscheidende Rolle spielt und warum sie die Geschwindigkeit einer Seite viel höher beeinflusst als die Bandbreite. Damit soll mit dem Mythos „Schnelles Internet gleich schnelles Laden einer Webseite“ aufgeräumt werden.

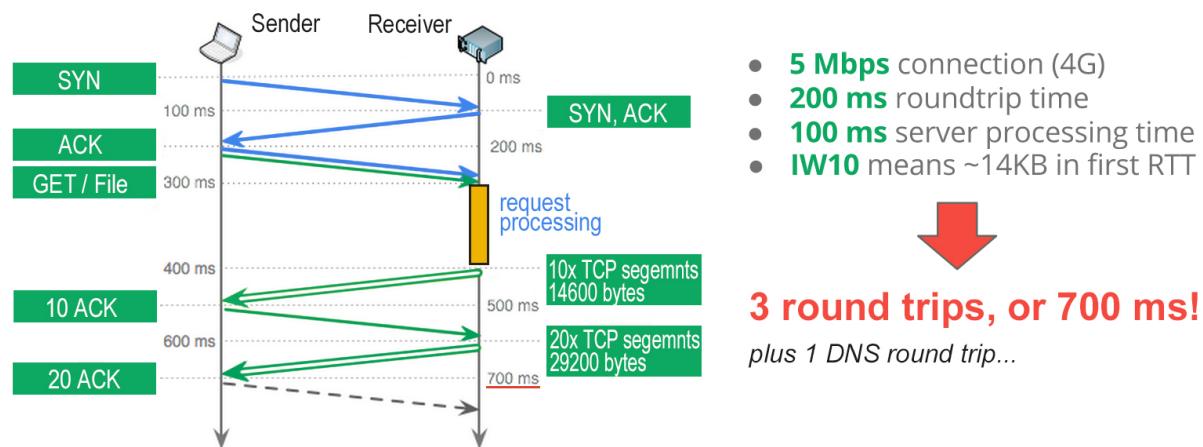


Abbildung 12: Herunterladen von 40kb mittels TCP (Abbildung nach (Grigorik 2013b))

Zuerst erfolgt der DNS Lookup, dann muss mittels TCP **three way handshake** eine Verbindung aufgebaut werden. Dies kostet bereits 2 Round Trips, was in diesem Beispiel 400 ms entspricht. Anschließend wird die Anfrage vom Server verarbeitet (**request processing**) und die Antwort vorbereitet. Durch den TCP slow start (siehe Punkt: 2.7) steht bei einer neuen Verbindung nicht die gesamte Bandbreite zur Verfügung. Deshalb kann die volle Datenmenge nicht auf einmal, sondern nur durch zusätzliche Round Trips heruntergeladen werden. Wenn die Performance einer Webanwendung verbessert werden soll macht es also Sinn in Round Trips zu denken. Wie viele Round Trips sind nötig, bis ich dem Browser Informationen übermittelt habe, sodass dieser etwas anzeigen kann? Idealerweise genau einer.

Abbildung 13 veranschaulicht die Kosten in Millisekunden, die auf jeder Ebene anfallen. Dieses Rechenbeispiel gibt eine ungefähre Vorstellung davon, wie viel noch von dem 1000

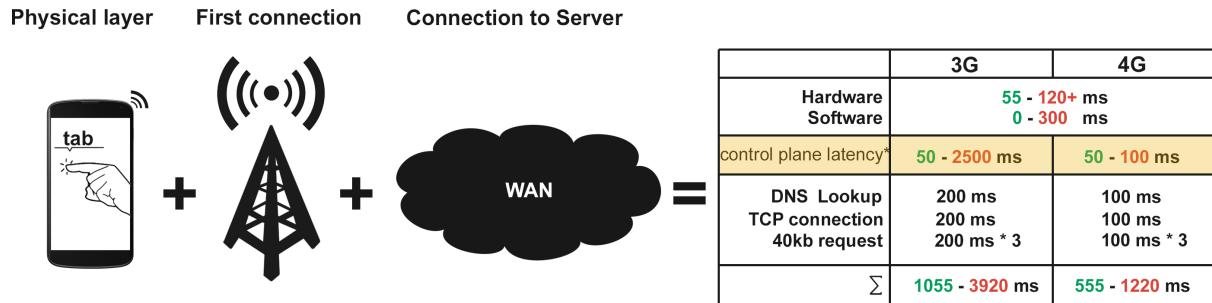


Abbildung 13: \*control plane latency: Wenn noch keinerlei Verbindung zu einem Sendeturm aufgebaut wurde, entstehen einmalige Authentifizierungskosten (Eigene Abbildung nach Daten von: (Takahashi 2013)(Grigorik 2013a, p. 7, 12))

Millisekunden Budget bleibt, wenn alleine die Netzwerkzeiten abgezogen werden. Für Nutzer mit 3G Netz ist es laut dieser These selbst im „best case“ Szenario nicht möglich, die 1000 ms Marke zu durchbrechen. Vor allem wenn man bedenkt, dass das 3G Netz eine Latenz von 100 bis 500 ms haben kann und hier schon ein optimistischer Wert von 200 ms verwendet wurde. Für Nutzer von 4G schaut es besser aus und es bleiben theoretisch 445 ms übrig.

Für die Performance Optimierung bedeutet es, dass in den ersten Kilobytes bereits genügend nützliche Informationen vorhanden sein sollten, damit der Browser mit dem Rendering beginnen kann, obwohl noch nicht alle Daten heruntergeladen sind. Noch spitzer formuliert: Der Browser sollte mit den ersten 14 kB (das ist die Menge an Daten die der erste Round Trip transportieren kann, siehe Abbildung 12) bereits den *above the fold* Bereich rendern können. Um das zu ermöglichen ist es nötig den kritischen Rendering-Pfad zu optimieren.

### 3.5 Zusammengefasst

Dieses Kapitel hat aufgezeigt, dass die Bandbreite nur eine untergeordnete Rolle spielt, wenn von schnellen Webanwendungen die Rede ist. Die Ladezeit wird dominiert von der Latenz und der damit verbundenen Round Trip Time. Diese entscheidet maßgeblich, wie schnell oder wie langsam der Ladevorgang ist. Folgendes ergibt sich auf der Ebene des Netzwerks:

- Die Ladezeit wird für mobile Anwender durch die Latenz bestimmt. 4G kann hier bereits Zeiten von <100 ms liefern, was das Erreichen der 1000 ms Barriere enorm erleichtert.
- Wahl eines guten Hostings: Die RTT als auch die **Server Response Time** sind je nach Anbieter unterschiedlich. Ein gutes Hosting kann hier unter Umständen bereits eine enorme Verbesserung bedeuten. Zur Not sollte gewechselt werden.

Wie in der Grafik zu sehen ist, sank die Response Zeit meines Hosting Providers von durchschnittlichen fast 400 Millisekunden auf 186 ms. Dies kann mehrere Gründe haben. So kann sich das Routing zum Server geändert haben, der Server kann ein

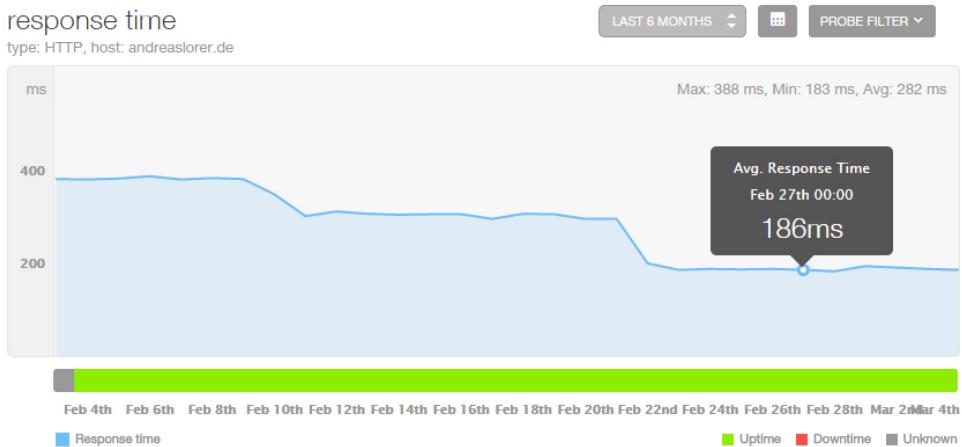


Abbildung 14: Verringerung der Response Time für <http://andreaslorer.de> (Abbildung nach pingdom.com)

Update erhalten haben oder die Maschine kann gewechselt worden sein. Was letzten Endes dazu geführt hat, kann in diesem Fall nicht genau gesagt werden.

- Vermeiden von Weiterleitungen: Abbildung 15 zeigt den Seitenaufruf von hasbro.com. Wie zu sehen ist, gibt es einen HTTP 301 (Wert in Klammer) Response zurück:

301 - Moved Permanently: „*Die angeforderte Ressource steht ab sofort unter der im „Location“-Header-Feld angegebenen Adresse bereit (auch Redirect genannt). Die alte Adresse ist nicht länger gültig.*“ (Wikipedia 2014)

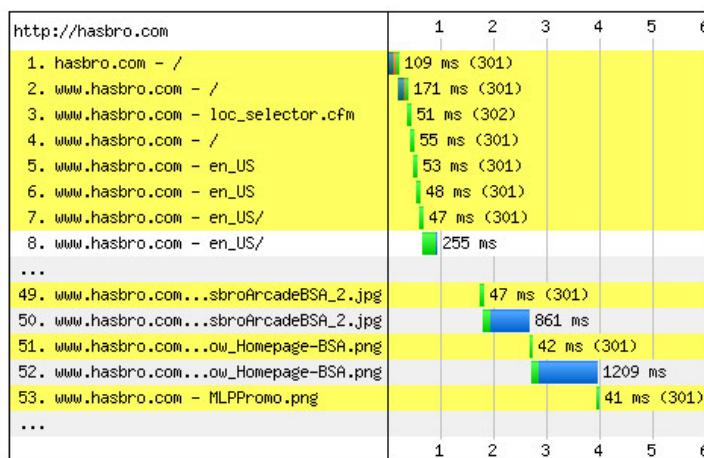


Abbildung 15: Testlauf von hasbro.com: „Dulles, VA USA - Thinkpad T430 - Chrome - Cable“ via [webpagetest.org](http://webpagetest.org))

Wenn ein Anwender **hasbro.com** eingibt (Abbildung 15) erfolgt der DNS Lookup und die TCP Verbindung wird aufgebaut. Der Browser erfährt dann, dass die Res-

source unter einer anderen Adresse zur Verfügung steht. Danach erfolgt die DNS Auflösung für www.hasbro.com bei dem der gleiche Vorgang nochmals von statthen geht. Es erfolgt die Weiterleitung auf die jeweilige Ländersprache. Nach rund 3,2 Sekunden konnte die erste Anfrage an die richtige Zieladresse aufgegeben werden. Aber auch Bilder werden auf dieser Seite weitergeleitet, wie in den Anfragen 49, 51 und 53 zu sehen ist. Ruft man sich die RTT von einem 3G Netz ins Gedächtnis, dürfte klar werden, was dies für den Smartphonenuutzer bedeuten kann.

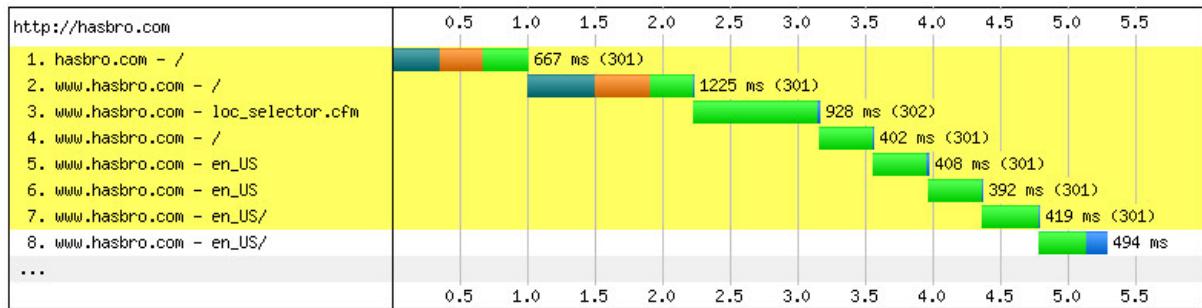


Abbildung 16: Testlauf mittels Smartphone von hasbro.com: „Dulles VA USA - Modell: MOTO G. 3G shaped 1.6Mbps / 300ms RTT“ Detailierter Test unter: [http://www.webpagetest.org/result/150310\\_AH\\_HVD/1/details/](http://www.webpagetest.org/result/150310_AH_HVD/1/details/)

Abbildung 16 zeigt den Aufruf von hasbro.com mittels Smartphone mit 3G Netz. 5.5 Sekunden dauert alleine der Verbindungsaubau zum HTML Dokument der Seite.

Die Weiterleitung von hasbro.com auf www.hasbro.com ist für die Suchmaschinen-optimierung<sup>7</sup> allerdings sinnvoll, denn sonst würde unter zwei Namen der gleiche Inhalt zu finden sein. Dies ist aus Sicht von Google „Duplicated Content“ und kann zu einer Abstrafung im Ranking führen.<sup>8</sup>

- Bei Anwendern die mittels 3G Verbindung im Internet sind bestehen wenig Möglichkeiten, eine Seite in unter einer Sekunde zu übermitteln. Die Zeit ab dem Touch Event und im Netzwerk kann bereits schon 900 ms betragen. Diese These deckt sich auch mit den Werten, die im Laufe des Projektes gesammelt wurden. Eine Auswertung davon findet in Kapitel 6 statt.
- Näheres positionieren der Bits: Durch die Benutzung eines CDN's lassen sich Bits und Bytes näher am Endanwender bereitstellen, was die Netzwerkzeiten verringert.
- Sage das Nutzerverhalten voraus: Wenn der Anwender in einer Einkaufs-App 3 Schritte zum Vollenden des Kaufvorgangs benötigt, dann lässt sich bei Schritt 1 bereits vorhersagen, was er für weitere Ressourcen im nächsten Schritt benötigt. Diese Ressourcen könnten bereits geladen werden. Das hat den Nachteil, dass wenn

<sup>7</sup>engl. SEO - Search engine optimization

<sup>8</sup>Mehr zu diesem Thema gibt es unter <http://www.sem-deutschland.de/seo-tipps/duplicate-content-definition/>

der Nutzer nie zu Schritt 2 oder 3 kommt, dann wurde das Internetvolumen des Anwenders (für die der Nutzer eventuell bezahlt) umsonst belastet.

- Senden von weniger Daten: Das schnellste Bit ist das, das nicht gesendet wird. Das Zusammenfügen und Verkleinern von Javascript und CSS Dateien verringert die Dateigröße. Zudem lassen sich die Daten per GZIP zwischen Browser und Server komprimieren. Wir wissen außerdem aus Abschnitt 2.5 (Http/1.1 Protokoll) bereits, dass die Anzahl von parallelen TCP Verbindungen limitiert ist und Abschnitt 2.7 hat den Einfluss des TCP slow start gezeigt. Wie das Verkleinern und Zusammenfügen in der Praxis umgesetzt werden kann, soll unter Punkt 5.2 konkretisiert werden.
- Damit nicht jede TCP-Verbindung neu aufgebaut werden muss, gibt es die Möglichkeit, die Verbindungen wiederzuverwenden. Dies nennt sich auch „Keep Alive“. Damit kann unter anderem der „three way handshake“ eingespart werden. Näheres dazu wird in Punkt 4.3.7 beschrieben.
- Stelle nützliche Bytes zu Verfügung: Wie in Abbildung 12 zu sehen ist, erfolgt mit dem ersten Round Trip eine Datenübertragung von 14 kB. Optimal ist es, wenn bereits mit dieser ersten Antwort genügend Informationen vorliegen, um etwas auf den Bildschirm des Anwenders zu zeichnen. Das setzt voraus, dass die HTML Datei 14 kB (nach Kompression) nicht überschreitet.

## 3.6 Kritischer Rendering-Pfad

Auf Englisch „Critical Render Path“ genannt, ist der wohl wichtigste Begriff wenn es um schnelle Ladezeiten geht. Durch die Optimierung des Rendering-Pfads kann die benötigte Zeit für das erste Rendern der Seite erheblich verkürzt werden. Das Verständnis des Rendering-Pfads ist zudem eine wesentliche Voraussetzung für die Erstellung von schnellen Webanwendungen und soll in diesem Abschnitt ausführlich erklärt werden. Dabei wird der Begriff in seine Teile zerlegt: Kritischer, Rendering und Rendering-Pfad.

### 3.6.1 Rendering

Der Browser liest das HTML Dokument und übersetzt es. Diesen Vorgang nennt man auch Parsen. „*Das Parsen der HTML- und CSS-Ressourcen und Ausführen von JavaScript beansprucht Zeit und Clientressourcen. Je nach Geschwindigkeit des Mobilgeräts und Komplexität der Seite kann dieser Prozess Hunderte von Millisekunden in Anspruch nehmen.*“ (Google 2014b) Bei der Optimierung von Webanwendungen ist besonders das Auftreten des ersten Zeichnens interessant (engl. „First Paint Event“). Je früher das First Paint Event auftritt, umso höher ist die Perceived Performance der Webanwendung.

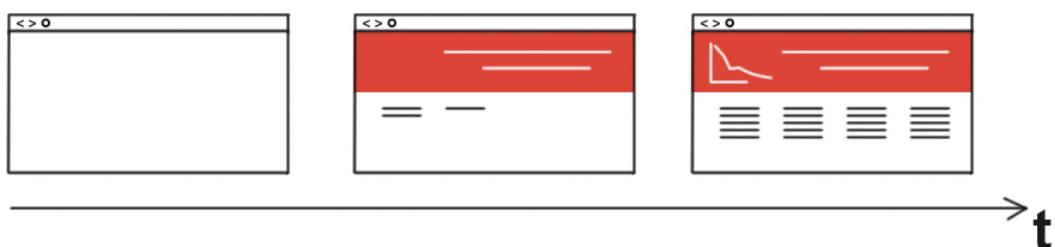


Abbildung 17: Der Render Prozess (Eigene Abbildung)

### 3.6.2 Rendering-Pfad

Der Rendering-Pfad setzt sich aus den für die Anwendung nötigen Ressourcen zusammen. Webanwendungen bestehen schließlich nicht nur aus einer HTML Datei, sondern aus mehreren Javascript und CSS Dateien.

Gegeben ist das folgende Beispiel einer simplen HTML Datei.

```
1      <!DOCTYPE html>
2      <meta charset="utf-8">
3      <title>Web Performance for mobile users</title>
4
5      <link href="assets/styles.css" rel="stylesheet" />
6      <script src="assets/script.js"></script>
7
8      <p> Hello world! </p>
```

Listing 1: Beispiel Code

Nachdem diese HTML Datei heruntergeladen wurde, beginnt der Browser sie von oben nach unten zu parsen. Dabei stößt er in Zeile 5 auf einen Link-Tag, der ihn anweist diese

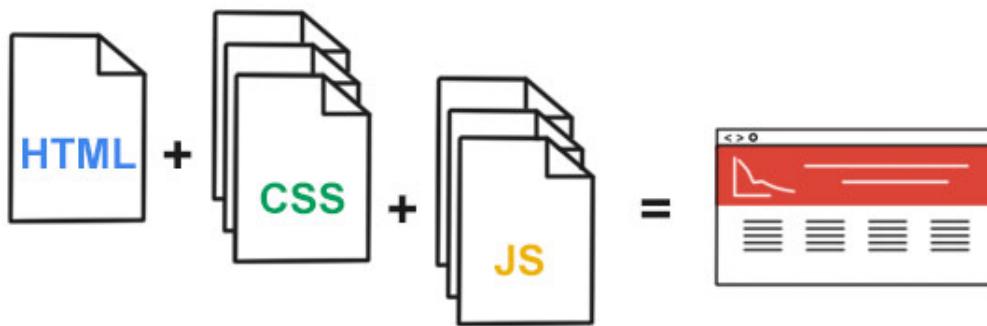


Abbildung 18: Ressourcen die für das Rendern nötig sind (Eigene Abbildung)

Datei herunterzuladen. In Zeile 6 findet der Browser einen `Script`-Tag. Auch diese Datei muss heruntergeladen, interpretiert und ausgeführt werden, denn jede Javascript Datei kann den DOM-Baum<sup>9</sup> oder das CSS manipulieren. So können per Javascript sowohl Elemente dem DOM-Baum hinzugefügt, als auch weggenommen werden oder Elemente können eine Änderung ihrer CSS Attribute erhalten. Dieser Umstand verbietet es dem Browser, mit dem Rendering zu beginnen, da bis zur Ausführung der Javascript Dateien noch Manipulationen erfolgen können. Bevor also das „Hello world“ in Zeile 8 angezeigt werden kann, ist das Rendern blockiert. Dieses Verhalten nennt sich auch „Render Blocking“ und wird sowohl von Javascript als auch von CSS Dateien ausgelöst. Folglich spricht man hierbei auch von „Render Blocking Javascript“ und „Render Blocking CSS“. Erst wenn diese blockierenden Ressourcen geladen und interpretiert wurden, kann der Browser den Render Tree erstellen, das Layout festlegen und schlussendlich mit dem Rendern beginnen.

### 3.6.3 Critical Render Path

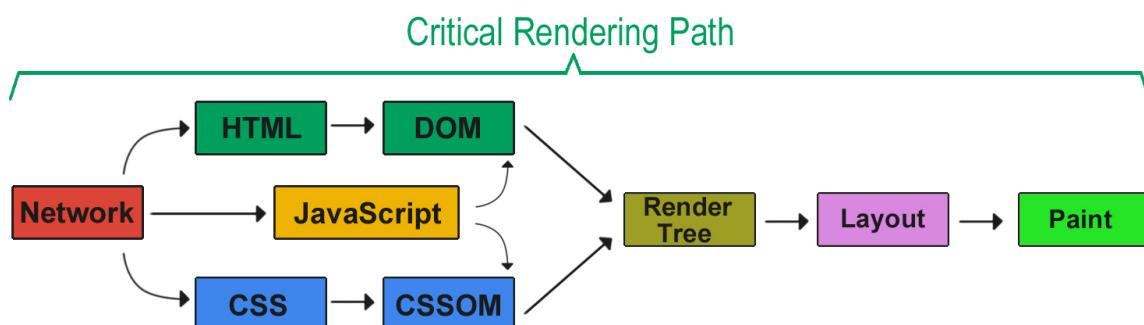


Abbildung 19: Der Kritische Rendering-Pfad (Eigene Abbildung nach ([santana14](#) ))

Der **Critical Render Path** besteht aus genau den Javascript und CSS Dateien, die für das Rendern des *above the fold* (Punkt: 2.8) von Nöten sind. Um dies umzusetzen ist

<sup>9</sup>Der DOM-Baum: <http://wiki.selfhtml.org/wiki/JavaScript/Objekte/DOM>

es nötig die Ressourcen in zwei Teile zu zerlegen: Für das Rendering **absolut** notwendig und nicht notwendige Ressourcen. Alle Dateien, die für das **erste** Rendern nicht notwendig sind, sollten so lange mit dem Laden verzögert werden, bis der above the fold der Anwendung geladen ist. Wie genau so eine Umsetzung aussieht, wird in Punkt 4.3.1 noch ausführlich gezeigt.

### 3.6.4 Zusammengefasst

Folgende Pattern lassen sich, bedingt durch den **Kritischen Rendering-Pfad**, für die Erstellung von Webanwendungen ableiten.

- CSS Dateien möglichst weit oben im „<head>“ Bereich platzieren und Javascript vor dem Schließen des „</body> tags“. Da Javascript und CSS so lange blockieren, bis sie heruntergeladen wurden, kann mit dem Parsen des gesamten Dokumentes nicht fortgefahren werden. (Bart 2014)  
Das Platzieren von Javascript am Ende des Dokuments hat allerdings den Nachteil, dass sich der Zeitpunkt des Herunterladens verzögert. Deshalb ist es ratsam, genau die Javascript Dateien in den „<head>“ zu verlagern, die **kritisch** für das Rendern des **above the fold** Bereichs sind und den Rest vor den schließenden „</body> tag“.
- Zusammenfügen von Dateien: Je weniger einzelne Dateien, umso weniger wird das Rendern der Seite blockiert.
- Aufteilen von Ressourcen in 2 Gruppen: Für das Rendering kritisch und unkritisch. Das gilt sowohl für CSS als auch für Javascript-Dateien. Unkritische Dateien werden so lange verzögert, bis der above the fold der Seite geladen wurde.
- Inlining von CSS im HTML Dokument: Durch das Einbetten von CSS direkt in das HTML Dokument wird das CSS bereits mit der ersten Serverantwort mitgesendet. Dadurch muss der Browser die Datei nicht anfordern und herunterladen, sondern kann gleich mit dem Parsen beginnen.
- Die Herausforderung besteht darin in der eigenen Webanwendung die für das Rendern kritischen Ressourcen zu erkennen und aufzuteilen, ohne die Funktionalität der Anwendung in Mitleidenschaft zu ziehen. Dinge die fast immer verzögert geladen werden können sind zum Beispiel Social Media Buttons (Facebook, Google+, Twitter ect.), Widgets oder Tracking Codes wie Google Analytics.

### 3.7 Analyse des Wasserfalls

Für ein besseres Verständnis des Kritischen Rendering-Pfads soll ein praktisches Beispiel einer unoptimierten Seite helfen. In Abbildung 20 ist der Ausschnitt eines Wasserfallmodells dargestellt.

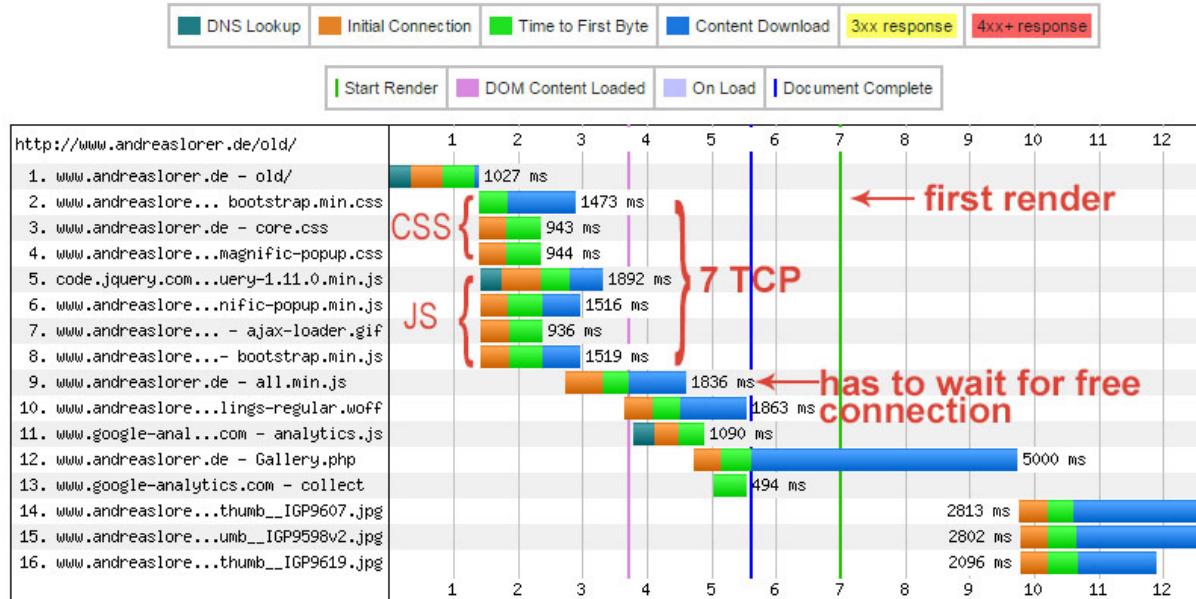


Abbildung 20: Testlauf von: „Dulles VA USA - Modell: MOTO.G. 3G shaped 1.6Mbps / 300ms RTT“ Ganzer Test: [http://www.webpagetest.org/result/150308\\_A1\\_2W4/8/details/](http://www.webpagetest.org/result/150308_A1_2W4/8/details/)

Sie zeigt das typische Verhalten des Browsers: Es werden zuerst CSS, Javascript und anschließend Bild-Dateien heruntergeladen. Hierbei fällt auf, dass er nicht mit allen Dateien gleichzeitig beginnen kann, sondern (wie in Punkt 2.5: HTTP/1.1 beschrieben) nur 6 TCP Verbindungen pro Host Name aufbauen darf.<sup>10</sup> Je weniger einzelne Dateien die Webseite benötigt, umso weniger bilden sich Warteschlangen für eine frei werdende TCP Verbindung (der Wasserfall wird flacher).

Auch ist wieder die Latenz als dominierender Faktor zu sehen. Es fällt auf, dass es nur einen relativ langen blauen Balken gibt (Anfrage Nr. 12 Gallery.php), bei dem für längere Zeit etwas heruntergeladen wird. Das Herunterladen der meisten Inhalte dauert überwiegend gerade so lange, wie die benötigte Zeit um eine Verbindung herzustellen.

Bei Sekunde 7 (senkrechte grüne Linie) tritt das Start Render Event ein: Der Browser hat mit dem Rendern begonnen. Das Ziel des Kritischen Rendering-Pfads ist es, diese senkrechte grüne Linie möglichst weit nach links zu schieben, also die Zeit bis zum ersten Rendern zu verringern.

Zum Vergleich nun in Abbildung 21 das Wasserfallmodell einer optimierten Seite:

<sup>10</sup>Es sind hier 7 Verbindungen, da die Datei „code.jquery“ von einer Google Domäne kommt und deshalb als neuer Host Name zählt.

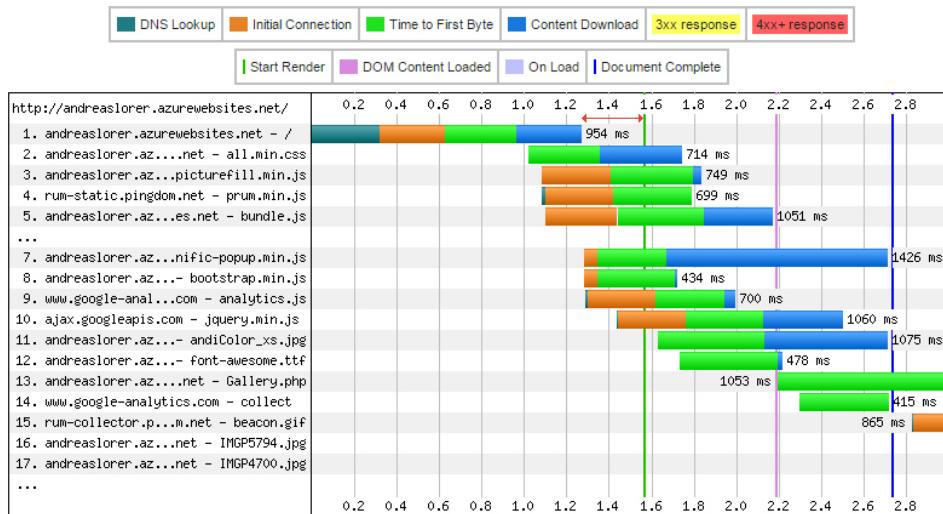


Abbildung 21: Selbe Testbedingungen wie bei Abbildung 20. Ganzer Test: [http://www.webpagetest.org/result/150308\\_5V\\_JSD/6/details/](http://www.webpagetest.org/result/150308_5V_JSD/6/details/)

Wie zu sehen ist, fällt die senkrechte grüne Linie bereits bei rund 1.6 Sekunden (5.4 Sekunden schneller). Das CSS und Javascript wird zu diesem Zeitpunkt noch heruntergeladen. Daraus lässt sich schlussfolgern, dass in dieser optimierten Version kein **Render Blocking Javascript / CSS** mehr vorhanden ist. Dadurch ist der Browser nicht blockiert und kann bereits früh mit dem Rendern der Seite beginnen. Dieses Diagramm lässt sich noch viel weiter interpretieren und belegt die Hauptaussage dieses Kapitels „**Brechen der 1000 ms Barriere**“:

Request Nummer 1 in Abbildung 22 zeigt genau, was in Kapitel 3.2 beschrieben wurde. Es spiegelt sehr schön die 300 ms RTT des 3G Netzes wieder:

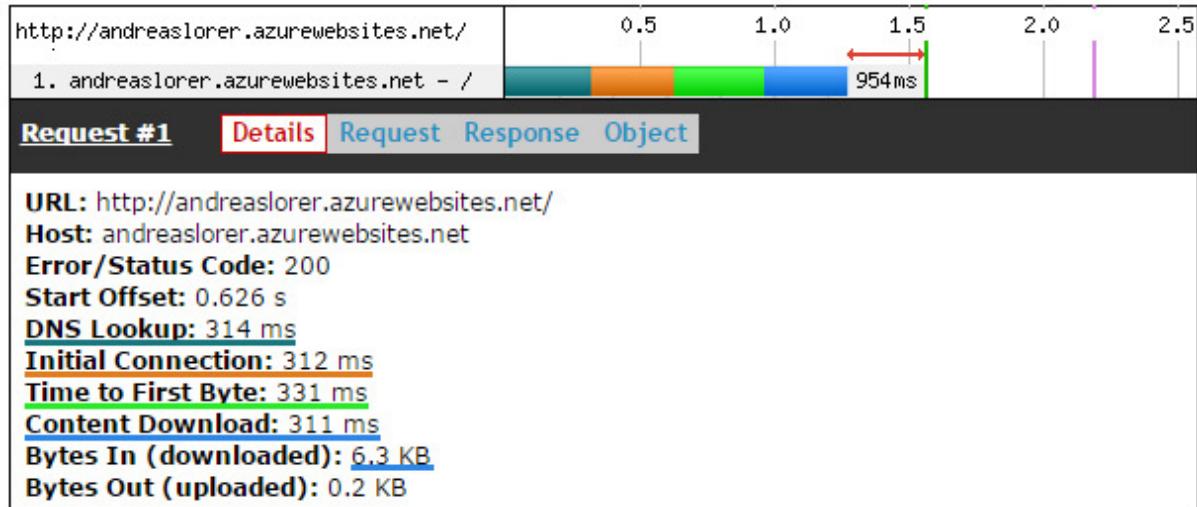


Abbildung 22: Request Nr. 1 im Detail (Abbildung nach webpagetest.org)

Mit gerundeten Werten ergibt sich:

- DNS Lookup: 1 RTT = 300 ms
- Initial Connection (TCP - 3 Way Handshake): 1 RTT = 300 ms
- TTFB: Server Processing Time<sup>11</sup>: = 300 ms
- Content Download<sup>12</sup>: 1 RTT = 300 ms
- Zeit fürs Parsen, Ausführen von Scripts ect.: ca. 500 ms <sup>13</sup>:

Obwohl hier nur eine 6.3kb Datei heruntergeladen wurde und keine 40kb wie in Beispiel 13, ist es ohne eine geringere Latenz (4G) nicht möglich, unter die 1000 ms Barriere zu kommen. Leider stellt der Serviceanbieter von [Webpagetest.org](#) keine Testumgebung mit 4G zu Verfügung. Ein Beweis für das Erreichen eines „First Render“ mittels Smartphone in unter 1000 ms Sekunde steht folglich aus. Mittels Kabelverbindung sind Werte um die 200 ms zu erreichen; dies wird per Datenauswertung in Kapitel 6 gezeigt.

---

<sup>11</sup>die Antwort muss erst vom Server generiert werden

<sup>12</sup>Das HTML Dokument beträgt 6.3 kB (siehe in Abbildung 22 Eintrag: Bytes In (downloaded)), TCP kann mit dem ersten Round Trip rund 14 kB transportieren. Das HTML Dokument kann also in einem Round Trip geliefert werden.

<sup>13</sup>Dies ist in der Abbildung mittels rotem Pfeil (<- ->) markiert

## 4 Entwicklung

Dieses Kapitel soll den Entwicklungsprozess konkretisieren und den Optimierungsprozess einer Webanwendung aufzeigen. Es soll erläutert werden, wie bestimmte Probleme gelöst werden, welche Fragen sich stellten und welche Antworten darauf gefunden wurden. Dafür werden die entsprechenden Tools und Hilfsmittel die zu Verwendung kamen vorgestellt und entsprechend erklärt. Dies soll ein Bewusstsein dafür schaffen, was möglich ist und wie eine technische Umsetzung aussehen kann.

### 4.1 Tools

Dies ist eine Auflistung an Tools und nützlichen Seiten, die entweder im Projekt verwendet oder für wertvoll befunden wurden und deshalb hier ihren Platz finden, damit jeder für sich entscheiden kann, ob ihr Einsatz sinnvoll sein könnte.

#### 4.1.1 Google Chrome Developer Tool

Dieses Tool ist über die Taste F12 im Chrome Browser zu finden. Nützliche Funktionen sind:

- **Device Emulation**<sup>14</sup>: Damit lassen sich sowohl verschiedene Geräte wie Smartphones oder Tablets, als auch das Touchverhalten simulieren.
- In der Device Emulation lässt sich auch die Netzwerkgeschwindigkeit simulieren.
- Netzwerk: Hier lässt sich das Wasserfallmodell nachvollziehen. Auch lässt sich hier das Caching des Browsers abschalten, während das Developer Tool geöffnet ist.
- Audits: Unter diesem Reiter bekommt man erste Informationen, welche Verbesserungen sich für diese Seite aus dem Gesichtspunkt der Performance ergeben. So wird zum Beispiel aufgezeigt, wie viele CSS Selektoren auf dieser Seite gar keine Verwendung finden (gerade bei CSS-Frameworks wie Bootstrap kann es sein, dass rund 90% der Selektoren keine Verwendung haben).

#### 4.1.2 Google Pagespeed Insight

Pagespeed Insight ist ein Analysetool für Webanwendungen. Per URL Eingabe wird die Anwendung aufgerufen und gegen die „Best Practices“ von Google getestet<sup>15</sup>. Dabei wird eine Wertung von 1 (schlecht) bis 100 (gut) vergeben. Mobile und Desktop Version werden voneinander unabhängig bewertet. Findet das Tool einen Verstoß, so gibt es Hilfestellungen wie zum Beispiel weiterführende Links oder Hinweise zur Behebung des Problems. Für die Verbesserung der Perfomance ist dieses Tool eines der besten Anlaufziele um die Probleme zu finden. Pagespeed Insight gibt es auch als Plugin für das Google Chrome Developer Tool.<sup>16</sup>

---

<sup>14</sup>Bei geöffnetem Tool (F12): strg + shift + M oder klick auf das Smartphone Symbol

<sup>15</sup>Beispiel für einen Test: <http://tinyurl.com/nvxksks>

<sup>16</sup>Plugin - Pagespeed Insight: <http://tinyurl.com/mv8fcx8>

### 4.1.3 Google Closure Compiler

Das ist ein einfaches Tool von Google<sup>17</sup> mit der Aufgabe Javascript zu verkleinern. Dieser Vorgang nennt sich „minify“ und ist auch für HTML und CSS möglich. Ein Beispiel:

Listing 2: Input

```

1  /**
2   * urlEncodes an object to send it via post
3   * @param {Object} object Object with key value pairs
4   * @return {String} string in format key=value&foo=bar
5   */
6  var urlEncode = function (object) {
7      var encodedString = '';
8      for (var prop in object) {
9          if (object.hasOwnProperty(prop)) {
10              if (encodedString.length > 0) {
11                  encodedString += '&';
12              }
13              encodedString += encodeURIComponent(prop + '=' + object[prop]);
14          }
15      }
16      return encodedString;
17 };

```

Wird zu:

Listing 3: Output

```

1  var urlEncode=function(c){var a="",b;for(b in c)c.hasOwnProperty(b)&&
2  (0<a.length&&(a+="&"),a+=encodeURIComponent(b+"="+c[b]));return a};

```

Wie zu sehen ist werden nicht nur alle Kommentare, Leerzeichen und Zeilenumbrüche entfernt, sondern auch Variablennamen werden auf 1 Zeichen reduziert um weitere Bytes zu sparen. Die Funktionalität bleibt dabei gewährleistet. Dieser Vorgang ist auch unter dem Namen „uglify“ bekannt.

### 4.1.4 Webpagetest

Webpagetest.org ist das wohl umfangreichste und beste Webseiten Analysetool das im Internet zu finden ist. Es ist ein kostenloser Service, der hauptsächlich von Patrick Meenan entwickelt wurde. Das Tool ist leicht zu bedienen aber schwer zu beherrschen („easy to use, hard to master“). Es gibt zahllose Einstellungen und undokumentierte Funktionen auf die man nur in Vorträgen oder Foren stößt. Ein Buch, das sich nur mit diesem Tool beschäftigt ist beim Verlag O'Reilly.<sup>18</sup> erschienen. Die Features für Webpagetest sind vielseitig:

- Webpagetest hat die wohl genaueste Erfassung von Netzwerkzeiten und spiegelt damit realitätsgerecht die Ladezeiten einer Seite wieder.
- Es liefert ein umfassendes Spektrum an Daten und Diagrammen, was ausführliche Analysen zulässt.

<sup>17</sup> <http://closure-compiler.appspot.com/>

<sup>18</sup> Buch - Using WebPagetest: <http://shop.oreilly.com/product/0636920033592.do>

- Der Speed Index ist eine von diesem Tool eigene Maßeinheit zum bestimmen der **Perceived Performance** einer Seite.

*„The Speed Index metric was added to WebPagetest in April, 2012 and measures how quickly the page contents are visually populated (where lower numbers are better). It is particularly useful for comparing experiences of pages against each other (before/after optimizing, my site vs competitor, etc) and should be used in combination with the other metrics (load time, start render, etc) to better understand a site’s performance.“*(webpagetest.org 2015, vgl.)

- Es lassen sich Webanwendungen mittels eines in der Realität existierenden Geräts testen. So kann vom Standort Dulles VA ein MOTO G zum Testen einer Seite verwendet werden. Dieses Gerät ruft dann auch wirklich die eingegebene URL auf und die darunterliegende Schicht misst die Zeit. Abbildung 23 zeigt den Teststandort Dulles VA.<sup>19</sup>

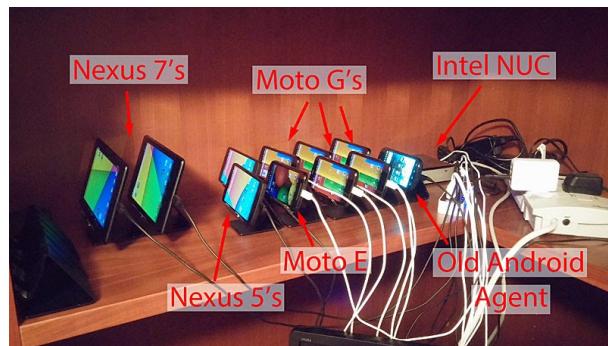


Abbildung 23: Webpagetest Android Geräte Farm (Abbildung von (Meenan 2014))

- Man kann Tests direkt miteinander vergleichen. Das ist möglich, indem diese URL eingegeben wird: [www.webpagetest.org/video/compare.php?tests=](http://www.webpagetest.org/video/compare.php?tests=) und nach dem „=“ Zeichen die Test ID, beispielsweise „150310\_8E\_GRH“, eingegeben wird. Mit einem Komma getrennt wird eine 2. oder 3. ID angefügt. Die Tests werden dann in einer Vergleichsansicht dargestellt.
- Filmstrip Ansicht: Damit lässt sich visuell erkennen wann welches Element gerendert wird.
- Videoerstellung: Aus der Filmstrip Ansicht lässt sich ein Video erstellen. Das ist vor allem interessant, wenn mit der Vergleichsmethode mehrere Tests geladen sind. Der Ladevorgang der Testläufe wird dann in einem Video parallel abgespielt. Vor allem für Präsentationen oder vorher- / nachher-Vergleiche ist dies nützlich.
- Test History: Durch eine Registrierung auf der Seite wird ein eigenes Testprofil angelegt, in dem alle Test-ID's gespeichert werden.

<sup>19</sup>Einen detaillierten Einblick vom Gründer und Entwickler Patrick Meenan gibt es hier: <http://tinyurl.com/o4b3rxh>

- Testen von verschiedenen Standpunkten: Webpagetest ermöglicht es die eigene Seite von ganz verschiedenen geographischen Standpunkten aus aufzurufen. Dadurch lässt sich ein Eindruck gewinnen, wie schnell die Seite aus dem Ausland aufrufbar ist und wie stark die Abweichung sein kann.
- API: Webpagetest hat eine offene API (Schnittstelle) durch die das Tool von außerhalb erreichbar ist. So lässt sich ein Test beispielsweise in Google-Spreadsheets aufrufen und das Ergebnis direkt in eine Tabelle schreiben (mehr dazu in Kapitel: 6.1 „Wie wurde getestet?“). Diese Schnittstelle limitiert allerdings die Anzahl an Tests pro Tag auf 200. Für mehr muss man sich eine eigene private Instanz erstellen.
- Private Instanz: Da Webpagetest Open Source is, gibt es die Möglichkeit eine eigene private Instanz aufzusetzen. Dies kann sowohl per Amazon Cloud oder auf einem eigenen Server geschehen. Damit lassen sich dann beliebig viele Tests ausführen.

#### 4.1.5 Pingdom

<http://tools.pingdom.com/fpt/> ist eine Alternative zu Webpagetest. Auch damit lässt sich eine URL nach Performanceproblemen analysieren. Die Ergebnisse sind nicht so genau wie mit Webpagetest und auch ein Testen mit Smartphones fehlt. Bei einer kostenlosen Anmeldung erhält man allerdings ein System zur Überwachung der eigenen Webanwendung. Bei Ausfall oder zu hoher Last kann eine SMS versendet werden, um den Admin auf diesen Umstand hinzuweisen. Durch Einbetten eines Scripts auf der eigenen Seite lässt sich die Response Zeit aufzeichnen (siehe Abbildung 14). Dieses Tracking nennt man auch „real user monitoring“ und ist zum Beispiel auch durch Google Analytics in solcher Form möglich.

#### 4.1.6 Speedcurve

Speedcurve ist ein kommerzielles Tool basierend auf Webpagetest. Es liefert einen „life monitoring“ Service mit dem sich Webanwendungen vergleichen lassen. So kann man zum Beispiel die eigene Webanwendung dauerhaft und über einen längeren Zeitraum mit der Konkurrenz vergleichen. Dies ist vor allem nützlich um nicht nur eine schnelle Seite beim Go-Live zu haben, sondern auch schnell zu bleiben und dies validieren zu können.

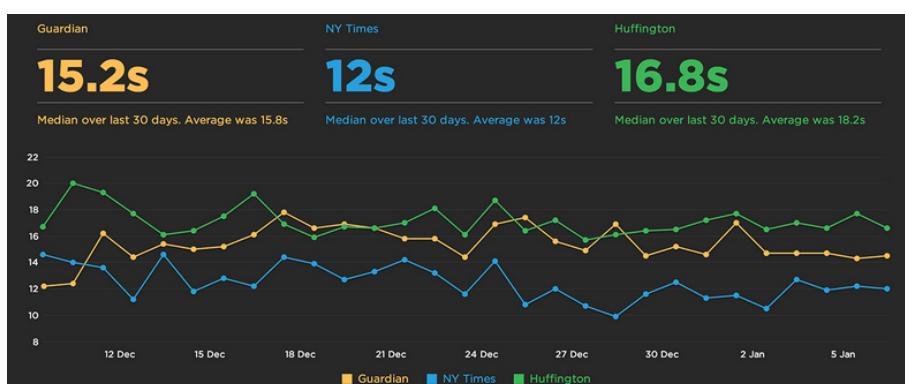


Abbildung 24: Speedcurve Life Monitoring (Abbildung von <http://speedcurve.com/>)

#### 4.1.7 Google Spreadsheet

Google Spreadsheet bietet ähnliche Funktionen wie Microsofts Excel. In Tabellen können Werte eingetragen und Berechnungen ausgeführt werden. Der große Vorteil an Google Spreadhseet besteht in der Möglichkeit, dass es einen Skript Editor gibt, mit dem sich Programme schreiben lassen. So sind zum Beispiel API Abfragen möglich, dessen Ergebnisse dann direkt in die Tabelle geschrieben werden können.

#### 4.1.8 Feed the Bot

<http://www.feedthebot.com/pagespeed/> bietet umfassende Artikel zu SEO und Web Performance. Wenn man sich mit dem Thema Web Performance beschäftigen möchte, ist dies eine erstklassige Anlaufstelle.

#### 4.1.9 What Does My Site Cost?

„Was kostet es eigentlich meinen Seitenbesucher, wenn sein Datenvolumen für diesen Monat aufgebraucht ist und er pro verbrauchtes MB zur Kasse gebeten wird?“ Diese Frage versucht diese Webanwendung zu klären und visuell darzustellen.

<http://whatdoesmysitecost.com/> benutzt die Webpagetest Schnittstelle, um eine eingegebene URL zu analysieren und berechnet aus den billigsten Anbieteren pro Land einen Preis für den Aufruf der Seite mittels Smartphone:

*„Prices were collected from the operator with the largest marketshare in the country and the for the least expensive plan with a (minimum) data allowance of 500 MB over (a minimum of) 30 days. Prices include taxes. Because these numbers are based on the least expensive plan, they are **best case** scenarios.“(whatdoesmysitecost.com 2015)*



Abbildung 25: Find out how much it costs for someone to use your site on mobile networks around the world.(whatdoesmysitecost.com 2015)

In Deutschland kostet also der Seitenaufruf von [www.hs-weingarten.de](http://www.hs-weingarten.de) rund 20 Cent. Dieses Tool stellt auf sehr schöne Art und Weise dar, dass schlechte Web Performance nicht nur den Anwender verärgert, sondern zusätzlich auch noch bares Geld kosten kann.

#### 4.1.10 Critical Path CSS Generator

Im Kapitel 3 „Brechen der 1000 ms Barriere“ wurde empfohlen, man solle das CSS des above the fold direkt in das HTML als inline CSS einbetten. <http://jonassebastianohlsson.com/criticalpathcssgenerator/> erstellt aus einer gegebenen URL und dem dazugehörigen CSS genau den CSS-Code, der für den above the fold Bereich nötig ist. Das Ergebnis lässt sich dann bequem in das eigene HTML Dokument integrieren.

Dieser Generator funktioniert allerdings nur dann zuverlässig, wenn sowohl die Smartphones, als auch Desktop Darstellung identisches CSS haben. Bootstrap zum Beispiel manipuliert die Navigation auf der Smartphone Ansicht per Javascript und fügt dabei Elemente ein. Diese hinzugefügten Elemente kennt dieser Generator natürlich nicht und kann sie folglich auch nicht beachten.

#### 4.1.11 Http Archive

<http://httparchive.org/> ist ein Archiv der populärsten Seiten des Internets und bietet eine Vielzahl an statistischen Auswertungen, Trends und Daten.

*„[HTTP Archive] is a permanent repository of web performance information such as size of pages, failed requests, and technologies utilized. This performance information allows us to see trends in how the Web is built and provides a common data set from which to conduct web performance research.“ (Archive 2015)*

#### 4.1.12 Perf Tooling Today

<http://perf-tooling.today/> ist wohl die umfassendste Sammlung an Web Performance Tools und Material im Internet. Es hat eine Liste von 105 Tools, 51 Artikel, 27 Videos und 14 Slidedecks (Stand: 12.03.15).

#### 4.1.13 Twitter

Twitter bietet die Möglichkeit, am Puls der Zeit zu sein und unter dem Hashtag #webperf und #perfatters erhält man neuste Erkenntnisse, Tools oder Links, die sonst vielleicht unentdeckt bleiben.

## 4.2 Ausgangspunkt

Im folgenden Abschnitt soll der Prozess beschrieben werden um von einer langsamen Webanwendung zu einer schnellen zu gelangen. Von Beginn an war es wichtig, den Verbesserungsablauf zu dokumentieren und in konkreten Daten zu erfassen. Wie bereits unter Punkt 4.1.7 beschrieben, bietet Google Spreadsheets die Möglichkeit, Skripte zu schreiben und die Ergebnisse direkt in eine Tabelle auszugeben. Diesen Umstand hat sich **Andy Davies** zu Nutzen gemacht und ein Programm<sup>20</sup> geschrieben (MIT License), das es ermöglicht Webpagetest innerhalb von Google Spreadsheet<sup>21</sup> aufzurufen. Damit wurden während der Entwicklungsphase täglich Tests aufgezeichnet.<sup>22</sup> Die Auswertung dieser Messdaten erfolgt in Kapitel 6.

Da nur in Dulles VA eine Testinstanz mit echten physikalischen Smartphones zur Verfügung steht, wurde mittels der **Microsoft Azure Cloud** die selbe Seite auch in den USA gehostet, um die Latenz zwischen USA und Europa zu umgehen. Dadurch lässt sich exakter bestimmen wie schnell ein Smartphone mit 3G Netz die Seite aufrufen kann. Leider steht aktuell keine Testinstanz mit 4G zur Verfügung.

Als Ausgangspunkt dient die Seite <http://andreaslorer.de/old/>. Zu Beginn des Optimierungsprozesses gab es folgenden Ausgangspunkt (Daten via Developer Tool & Webpagetest):

Desktop: <sup>23</sup>

- 42 requests: 30 Images, 5 JS, 3 CSS, 4 other
- 1000 kB Seitengröße
- Speed Index: **3584**
- Start Render: **1399** ms
- Load Time: 1926 ms
- TTFB: 690 ms

Mobile: <sup>24</sup>

- 17 requests: 4 Images, 5 JS, 3 CSS, 4 other
- 363 kB Seitengröße
- Speed Index: **10642**
- Start Render **6968** ms

---

<sup>20</sup>WebPageTest Bulk Tester via GitHub: <https://github.com/andydavies/WPT-Bulk-Tester>

<sup>21</sup>Das Google Dokument ist hier zu finden: <http://tinyurl.com/nv4pge5>

<sup>22</sup>Die gesamten Daten sind hier zu finden: <http://tinyurl.com/15usz79>

<sup>23</sup>Webpagetest: [http://www.webpagetest.org/result/150312\\_Z1\\_18QD/](http://www.webpagetest.org/result/150312_Z1_18QD/)

<sup>24</sup>Webpagetest: [http://www.webpagetest.org/result/150308\\_A1\\_2W4/](http://www.webpagetest.org/result/150308_A1_2W4/)

- Load Time: 5587 ms
- TTFB: 1292 ms

Diese Werte sind nicht zufriedenstellend und für dieses Projekt wurden eine Start Render Zeit von weniger als einer Sekunde und ein Speed Index von unter 1000, sowohl für Mobile- als auch Desktopgeräte, angestrebt.

Der erste Schritt war es die Seitengröße zu verringern. Aus diesem Grund wurde das Framework gewechselt und die Seite neu aufgebaut. Bootstrap ist zwar ein sehr populäres Framework, hat aber gerade für kleine Seiten sehr viele Komponenten, die keine Verwendung finden (auch als Overhead bezeichnet). Bootstrap lässt sich zwar per „Customize“ Funktion so zusammenstellen, dass nur die Komponenten zur Verfügung gestellt werden die für das eigene Projekt von Nöten sind, es ist aber dennoch ein Framework mit relativ großem Volumen (30 bis 90 kB). Die Alternativen zu Bootstrap sind vielzählig. Die Entscheidung für dieses Projekt fiel auf <http://purecss.io/>. Dieses Framework von Yahoo ist komprimiert gerade einmal **4 kB** groß, vollkommen responsive und kommt mit den wichtigsten Komponenten wie einer Navigations Bar, Buttons, Tabellen, Menüs und Form Elementen. Je nach gewählten Komponenten benötigt es kein Javascript und kein JQuery. Dadurch lassen sich weitere Kilobytes als auch Requests einsparen.

Da Bootstrap seine eigene Icon-Font liefert musste hierzu eine Alternative gefunden werden. „Font Awesome“<sup>25</sup> bietet dabei eine der umfangreichsten Icon Sammlungen im Web an und ist unter der [Open Font License](#) komplett frei benutzbar (auch kommerziell). Font Awesome ist mit seinen 519 Icons allerdings nicht gerade ein Leichtgewicht und kann bis zu 100 kB groß sein. Da auf der Seite <http://andreaslorer.de> weniger als 20 Icons zum Einsatz kommen, ist der Überschuss folglich enorm. Deshalb gibt es eine Webanwendung namens <http://fontello.com>. Damit lassen sich aus einer Vielzahl an Icons genau diejenigen wählen, die für die eigene Seite benötigt werden. Auch das Wählen aus verschiedenen Icon-Sammlungen ist möglich. Heruntergeladen wird anschließend eine ZIP-Datei. Das Resultat: Die neue Version der Seite benötigt nur noch 5.6 kB für die Icons. Verglichen mit: Bootstrap 43 kB, Font-Awesome 97 kB.

Als nächstes wird die Webanwendung mittels [Pagespeed Insight](#) 4.1.2 analysiert. Das Ergebnis liefert Anhaltspunkte, welche weiteren Änderungen umgesetzt werden sollte. Im Folgenden soll erläutert werden, welche Verbesserungen möglich sind und wie eine mögliche Umsetzung in der Praxis aussieht.

---

<sup>25</sup>Font-Awesome: <http://fortawesome.github.io/Font-Awesome/>

## 4.3 Best Practices

### 4.3.1 Render Blocking Javascript / CSS

Bereits unter Punkt 3.6.2 „Rendering-Pfad“ ist das blockierende Verhalten von Javascript und CSS angesprochen worden. Grundvoraussetzung für diesen Punkt ist, dass das Javascript der Webanwendung in ihre für das Rendern kritische und unkritische Teile zerlegt wurde.

Der Browser stellt bereits von Haus aus zwei Attribute bereit, mit denen sich Skripte asynchron herunterladen lassen. Diese Attribute heißen „async“ und „defer“ und werden von jedem Browertyp unterstützt.(caniuse.com 2015) Sie erlauben es, dass der Browser nicht auf das Herunterladen der Dateien warten muss, sondern mit dem Parsen des Dokuments fortfahren darf. Async wird direkt nach dem herunterladen ausgeführt und dafür muss das Parsen pausiert werden. Defer hingegen unterscheidet sich von async in zwei Punkten: 1. Das Skripte wird nach Ende des Parsens ausgeführt. 2. Mit defer verzögert geladene Skripte werden in genau der Reihenfolge ausgeführt, wie sie im HTML Dokument vorliegen.

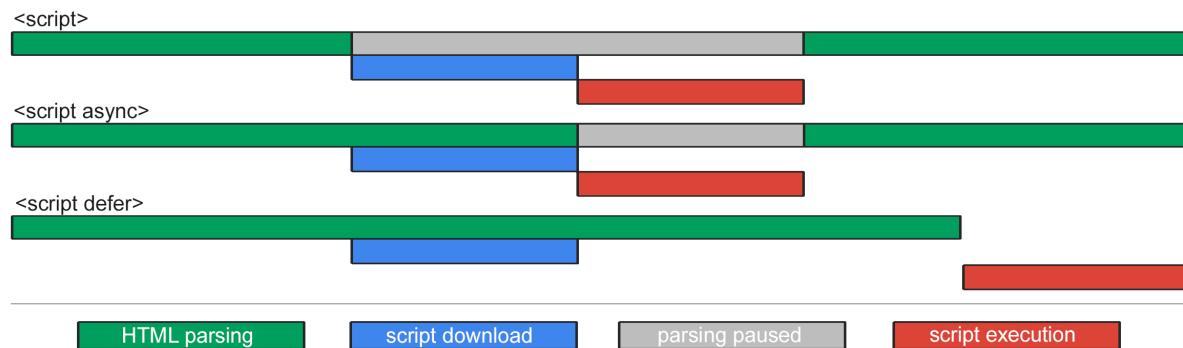


Abbildung 26: Script-Tags mit verschiedenen Attributen (Abbildung nach (growingwiththeweb.com 2014))

Diese Methode erlaubt es Skripte parallel herunterzuladen, ohne dass der Renderprozess warten muss. Was damit nicht erreicht werden kann ist das herunterladen so lange zu verzögern, bis die für diese Seite primär wichtigen Ressourcen zuerst geladen wurden. Im Vergleich dazu kann mit Hilfe des Javascripts in Listing 4 die Datei „defer.js“ komplett mit dem Laden verzögert werden, bis der Ladeprozess der Seite abgeschlossen ist.

```

1  function loadJS( src, cb ){ // the function to asynchronous load js
2    var ref = window.document.getElementsByTagName( "script" )[ 0 ];
3    var script = window.document.createElement( "script" );
4    script.src = src;
5    script.async = true;
6    ref.parentNode.insertBefore( script, ref );
7    if (cb && typeof(cb) === "function")
8      script.onload = cb;
9    return script;
10 }
11 loadJS( "path/to/script.js" ); // the function call to load your script

```

Listing 4: Javascript nach (Group 2015)

Dieses Skript hat einen Nachteil: Es kann nicht mehrere Skripte laden, die voneinander abhängen und deren Reihenfolge wichtig ist, um die Funktionalität zu gewährleisten.

*„loadJS does nothing to manage execution order of requested scripts, so we do not advise using it to load multiple javascript files that depend on one another. It simply fetches a script and executes it as soon as possible. You can certainly use loadJS to load multiple scripts as long as those scripts are designed to execute independently of any other scripts being loaded by loadJS.“ (Group 2015)*

So gibt es Skripte (Skript A), die von anderen Frameworks wie zum Beispiel JQuery (Skript B) abhängen. Das bedeutet: Wenn Skript A schneller heruntergeladen und ausgeführt wird als Skript B, dann kann A bereits Funktionen von B aufruft, die noch nicht zur Verfügung stehen. Daraus resultiert ein Fehlschlagen des Skripts und somit können Teile der Webanwendung nicht mehr wie beabsichtigt funktionieren.

Darum gibt es Skripte, die genau diese Funktionalität bereitstellen können. Skript A und B werden gleichzeitig heruntergeladen, A wird aber erst genau dann ausgeführt, wenn B zur Verfügung steht.

<http://headjs.com/> kann das erreichen. Durch das Herunterladen und Einfügen in das HTML Dokument, kann per Funktionsaufruf die Abhängigkeit festgelegt werden:

```
1      // Load up some script A and then script B
2      head.load("jquery.js", function() {
3          // Call a function when done
4          console.log("Done loading jquery");
5          head.load('defer.js');
6      );
```

Listing 5: Headjs dependency loading (Listing nach <http://headjs.com/>)

Headjs hat den Nachteil, dass es auch noch andere Funktionalitäten außer dem Laden von Javascript ermöglicht. Dies wird aber nicht benötigt und deshalb ist Headjs mit 2.1 kB doch zu groß, um es `Inline` in ein HTML Dokument einzubetten. Eine bessere Alternative ist jQl.<sup>26</sup> Die Verwendung ist sehr simpel:

```
1      <script type="text/javascript">
2          // first include jQl inline (missing here cause its too long) then
3          // call
4          jQl.loadjQ('jquery.js');
5          jQl.loadjQdep('defer.js');
```

Listing 6: jQl asynchronous jQuery-Loader

Dieses Skript sagt im Grunde: Lade beide Dateien gleichzeitig herunter, beachte aber die Abhängigkeit (`loadjQdep` steht für: load dependency) von `defer.js` gegenüber JQuery. Ist `defer.js` früher heruntergeladen als JQuery, so wird gewartet und anschließend werden die

<sup>26</sup>jQl an asynchronous jQuery Loader: <http://www.yterium.net/jQl-an-asynchronous-jQuery-Loader>

Skripte in der richtigen Reihenfolge ausgeführt.

Für die Webseite <http://andreaslorer.de> wurde sowohl defer, als auch das Skript jQ1 verwendet. „<script defer src='critical.js'></script>“ wird dabei in den „<head>“ Bereich des HTML Dokuments platziert, damit es möglichst früh erkannt wird und der Download bereits beginnen kann. Das Skript aus Listing 6 befindet sich vor dem „</body>“-Tag.

Wie Javascript blockiert auch CSS das Rendern der Seite. In Listing 7 ist ein Skript der Filament Group zu sehen. Dieses Skript ermöglicht es CSS verzögert zu laden.

```
1      <script>
2          // https://github.com/filamentgroup/loadCSS/blob/master/loadCSS.js
3          // [c]2014 @scottjehl, Filament Group, Inc.
4          // Licensed MIT
5          function loadCSS(e,a,g,h){function f(){for(var a,c=0;c<d.length;c++)d[c].href&&
6              -1<d[c].href.indexOf(e)&&(a!=0);a?b.media=g||"all":setTimeout(f)}
7              var b=window.document.createElement("link");a=a||
8                  window.document.getElementsByTagName("script")[0];
9              var d=window.document.styleSheets;b.rel="stylesheet";b.href=e;b.media="only x";
10                 b.onload=h||function(){};a.parentNode.insertBefore(b,a);f();return b
11             };
12             loadCSS( "path/to/css" );
13         </script>
14         <!-- fallback for disabled javascript -->
15         <noscript><link href="path/to/css"></noscript>
```

Listing 7: Lädt eine CSS Datei asynchron

Mehr als dieses Skript und der entsprechende Funktionsaufruf: „loadCSS(...)“ ist nicht notwendig. Dabei ist in Zeile 15 auch eine Fallback-Option vorhanden die in Kraft tritt, wenn der Anwender Javascript im Browser deaktiviert hat. Somit ist selbst ohne Javascript die Funktionalität gewährleistet.

#### 4.3.2 CSS-Bereitstellung optimieren

Wenn externe Ressourcen klein sind, können diese direkt in das HTML Dokument **Inline** platziert werden. Das trifft vor allem auf das CSS zu, dass für die korrekte Darstellung des above the fold nötig ist. Dabei sollte darauf geachtet werden, dass das komprimierte HTML Dokument nicht die 14 kB Marke überschreitet. Dadurch kann es im ersten round trip geliefert werden. CSS Dateien die groß sind, sollten per **Link-Tag** eingebunden und mittels Skript verzögert geladen werden. Das CSS für den above the fold Bereich sollte **Inline** im „<head>“ Bereich der Seite stehen.

#### 4.3.3 Ressourcen reduzieren

Das schnellste Byte ist das, das nicht gesendet wird und der schnellste Request ist der, der nicht gestellt wird. Deshalb gibt es drei Maßnahmen, die für eine Webanwendung umgesetzt werden sollten:

- „Minify“: Kommentare, Leerzeichen oder Zeilenumbrüche sind für die Funktionalität nicht notwendig. Bei der Verkleinerung des HTML- CSS oder Javascript Codes

werden diese entfernt und dadurch die Dateigröße verkleinert. Wie unter Punkt 4.1.2 angesprochen, gibt es Pagespeed Insight auch als Chrome-Erweiterung. Damit ist es möglich, eine reduzierte Version des HTML Dokuments zu erzeugen. Für Javascript ist der Closure Compiler (4.1.3) das richtige Werkzeug. CSS lässt sich per <http://cssminifier.com/> verkleinern.

- „Uglify“: Dabei werden Variablennamen auf nur wenige Zeichen reduziert. Aber auch der Code wird teilweise umgeschrieben, wenn für einen verwendeten Ausdruck beispielsweise eine kürzere Schreibweise existiert.

```
1 // input:  
2 if(foo){  
3     bar();  
4 }  
5 else{  
6     boo();  
7 }  
8 // output:  
9 foo?bar():boo();
```

Listing 8: Beispiel: Uglify eines Ausdrucks

Input und Output sind identisch in ihrem Ausdruck, die Zeichenanzahl wurde aber von 27 auf 16 verringert.

- „Concatenate“: Damit ist das Zusammenfügen von einzelnen Dateien zu einer Einzigen gemeint. Dadurch lassen sich zusätzliche Requests einsparen. Dies hat den Vorteil, dass sowohl der TCP slow start nur einmal eintritt als auch nur eine TCP Verbindung aufgebaut werden muss. Zusätzlich werden weniger TCP Verbindungen belegt.

#### 4.3.4 Antwortzeit des Servers reduzieren

Die Zeit zur Antwort des Servers lässt sich zum Beispiel mit Webpagtest oder Pingdom herausfinden. Ein Server sollte auf eine Response Zeit von unter 200 ms kommen. „*Es gibt Dutzende potenzielle Faktoren, die die Antwortzeit Ihres Servers beeinträchtigen können: eine langsame Anwendungslogik, langsame Datenbankabfragen, langsames Routing, Frameworks, Bibliotheken, CPU-Ressourcenmangel oder Speicherplatzmangel. Berücksichtigen Sie zur Verkürzung der Antwortzeit Ihres Servers alle diese Faktoren.*“ (Google 2015). Bereits unter Punkt: 3.5 wurde nahegelegt, ein gutes Hosting zu wählen. Besser man wechselt das Hosting als der Kunde den Service.

#### 4.3.5 Browser-Caching nutzen

Fehlendes Browser-Caching (das lokale Speichern von Daten) wird von Pagespeed Insight bemängelt, wenn der Server bei seiner Antwort keinen expliziten **Caching-Header** versendet. Durch das Speichern von statischen Ressourcen wie Javascript, Stylesheets und Bildern kann Zeit eingespart werden, wenn der Besucher die Webanwendung ein weiteres Mal aufruft. Generell sollten alle statischen Ressourcen, außer das HTML Dokument

selbst, gechached werden. Um auf dem Server (Apache) das Caching von statischen Ressourcen zu ermöglichen, ist ein Eintrag in die `htaccess` Datei des Servers nötig. Folgender Eintrag sollte dort platziert werden:

```
1 <IfModule mod_expires.c>
2   ExpiresActive On
3   ExpiresByType image/jpg "access 1 year"
4   ExpiresByType image/jpeg "access 1 year"
5   ExpiresByType image/gif "access 1 year"
6   ExpiresByType image/png "access 1 year"
7   ExpiresByType text/css "access 1 year"
8   ExpiresByType text/woff "access 1 year"
9   ExpiresByType application/pdf "access 1 year"
10  ExpiresByType text/x-javascript "access 1 year"
11  ExpiresByType application/x-shockwave-flash "access 1 year"
12  ExpiresByType image/x-icon "access 1 year"
13  ExpiresDefault "access 1 month"
14 </IfModule>
```

Listing 9: Aktivieren von Browser Caching in Apache (Listing nach: (Sexton 2015b))

Listing 9 hat zwei Aufgaben. Erstens: Es setzt die Ablaufzeit für alle statischen Ressourcen auf 1 Jahr und erfüllt damit den von Google empfohlenen Wert. Längere Speicherzeiten sind dagegen nicht empfohlen, da dies gegen die RFC-Richtlinien verstößen würde (Google 2014a). Zweitens: Es wird mit dem HTTP Request ein Header mitgesendet. Dieser ermöglicht es dem Browser seine lokal gespeicherten Ressourcen zu managen. Er besteht aus folgenden Teilen und es ist jeweils nur **eine** der Optionen nötig.

- Last-Modified: date
- ETag: ID

Diese beiden Header ermöglichen es dem Browser zu überprüfen, ob sich die gecachten Ressourcen geändert haben oder noch identisch sind. Last-Modified ist dabei das Datum der letzten Änderung und der ETag-Header ist ein automatisch generierter Wert, der die Datei eindeutig identifiziert.

Beim erneuten Laden einer Seite werden diese Header zurück an den Server gesendet und verglichen. Wenn die Datei auf dem Server geändert wurde, stimmen die Werte nicht überein und der Server schickt eine entsprechende Antwort zurück.

- Cache-Control: max-age=value
- Expires: date

Mit diesen Headern ist es möglich Serveranfragen komplett zu vermeiden. Sämtliche vom Browser ausgegebenen HTTP-Anfragen werden zuerst an den Browsecache weitergeleitet um zu überprüfen, ob eine gültige Antwort im Cachespeicher vorliegt, die der Anfrage entspricht. Liegt eine Übereinstimmung vor wird die Antwort aus dem Cache ausgelesen, wodurch sowohl die Netzwerklatenz als auch die durch die Übertragung anfallenden Datenkosten umgangen werden.(Grigorik 2014b, vgl.) Das bedeutet, dass die Latenz bei Smartphones für gecachte Dateien komplett vermieden werden kann. Gültige Ressourcen werden erst gar nicht angefragt, sondern gleich aus dem Cache geladen. Ungültige oder

abgelaufene Ressourcen werden dagegen vom Server geholt. Ohne diesen Header muss der Browser für jede in seinem Cache befindliche Ressource den Server anfragen. Dafür ist jedes mal ein Round Trip nötig.



Abbildung 27: Gecachte Ressourcen müssen nicht mehr abgefragt werden

Der in Abbildung 27 rot unterstrichene Wert zeigt, dass 59 ms im Netzwerk verbraucht wurde (Kabel Verbindung). Der Server antwortete mit: „304 - Not-Modified“. Grün zeigt mit seinen 0 ms, dass keinerlei Kommunikation mit dem Server nötig ist, sondern die Datei, in diesem Fall ein Bild, direkt aus dem Cache geholt wird.

Fazit: Ohne Cache-Control lässt sich durch die Browser eigene Caching Funktionalität das erneute Herunterladen der Datei vermeiden. Mit Cache-Control kann sowohl das Herunterladen, als auch der gesamte Verbindungsauflauf zum Server vermieden werden.

Was aber, wenn sich zum Beispiel eine CSS-Datei geändert hat? Dann würden nun Besucher mit leerem Cache eine andere Darstellung erhalten wie Besucher mit der gecachten Version. Dafür gibt es mehrere Lösungsansätze.

1. Die HTML Datei sollte nicht gecached werden, da sonst Änderungen nicht mehr den Anwender erreichen können.
2. Eine für die Datei angemessene max-age: Dateien die sich oft ändern dürfen auch entsprechend niedrige max-age Werte haben. Dadurch wird die Datei zeitnah ablaufen und für alle Anwender neu angefordert.
3. Ressourcen können mit einer ID versehen werden: `styles.css` wird in `styles.v1.0.1.css` umbenannt. Bei jeder Änderung wird dann diese Versionsnummer hochgezählt. Dadurch ändert sich der Dateiname und wird vom Browser als eine neue, noch nicht im Cache enthaltene Ressource erkannt, die er herunterladen muss.
4. Alternativ zur ID ist auch ein **Fingerprint** möglich. Dabei wird eine Zahl aus der Datei generiert. Ändert sich die Datei, so ändert sich auch der Fingerprint. Dieser Fingerprint wird auch wiederum dem Dateiname angefügt. Das kann so aussehen: `styles.82s0dfa.css`.<sup>27</sup>

Browser-Caching ist eine mächtige Funktionalität, die sich jeder zu Nutzen machen sollte. Sie ist zudem ganz einfach mit nur einem Eintrag in die `htaccess` Datei realisierbar. Allerdings hat eine Studie von Yahoo ergeben, dass 40-60% der Besucher beim Seitenaufruf einen leeren Cache haben und rund 20% aller aufgerufenen Seiten wurden mit einem leeren Cache aufgerufen.

<sup>27</sup>Dieses Verfahren lässt sich auch automatisieren, ich verweise auf folgenden Artikel: <https://adactio.com/journal/8504>

*„[...] I don't know about you, but these results came to us as a big surprise. It says that even if your assets are optimized for maximum caching, there are a significant number of users that will always have an empty cache.“ (Yahoo 2007)*

Folglich macht es Sinn die Geschwindigkeit der Seite für die sogenannten „first users“ zu optimieren und nicht von einem gecachten Zustand der Seite auszugehen. Feed-The-Bot stellt ein Tool<sup>28</sup> zu Verfügung, mit dessen Hilfe sich überprüfen lässt, ob die eigene Webseite „Browser-Caching“ richtig einsetzt. Abbildung Nummer 28 zeigt links eine Seite mit aktiviertem Browser-Caching und rechts eine Seite ohne.

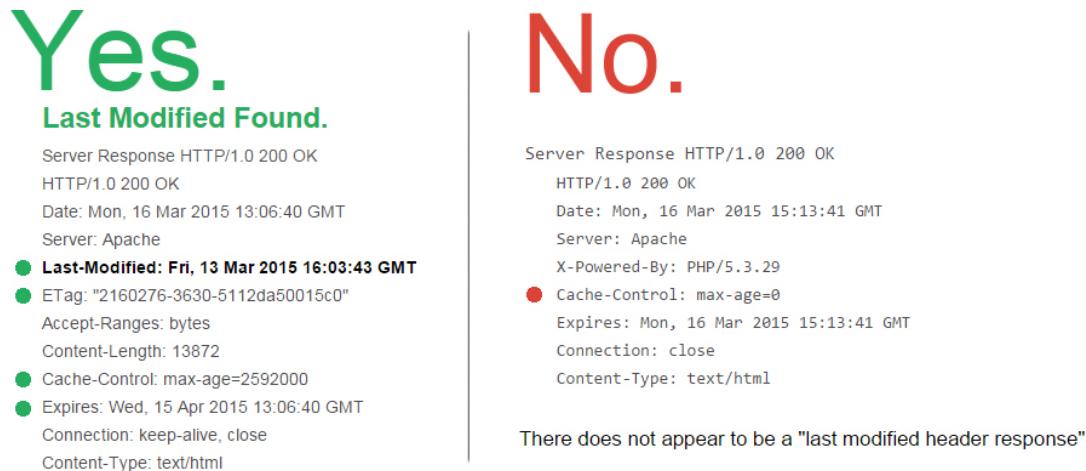


Abbildung 28: Beispiel: Eine Webseite mit und ohne „Cache Control“. (Eigene Abbildung nach feedthebot.com)

---

<sup>28</sup>Browser-Caching Tool: <http://www.feedthebot.com/tools/if-modified/>

#### 4.3.6 Komprimierung aktivieren

Nachdem durch das in Punkt 4.3.3 beschriebene Verfahren die Ressourcen soweit wie möglich verkleinert wurden, können diese vor dem Versenden komprimiert werden. Dies nennt man auch Datenkomprimierung und stellt ein ganz eigenes Forschungsgebiet dar. Deshalb soll nur der Grundgedanke erklärt werden.

Gegeben sei folgende Textnachricht:

```
1 # Lorem ipsum dolor sit amet, consectetur adipisicing elit. Debitis
   temporibus incidunt id.
2 # Cumque molestiae est praesentium magnam, fugit ipsa.
3 format: text/plain
4 date: 21.03.15
5 AAAZZBBBBEEMMM EEETTTAAAAA
```

Diese Nachricht ist 200 Zeichen lang und kann durch einfache Regeln verkürzt werden. Zuerst werden die Kommentare, die mit `#` gekennzeichnet sind, entfernt. Sie sind für die Bedeutung der Nachricht nicht relevant. Das Datum könnte in eine ID konvertiert werden: 210315. Die Nutzdaten der Nachricht wird nach Wiederholungen durchsucht. Daraus ergibt sich dann: 3A2Z4B3E3M 3E3T4A (Grigorik 2014a). Die Neue Nachricht:

```
1 format: text/plain
2 date: 210315
3 3A2Z4B3E3M 3E3T4A
```

Die Zeichenanzahl wurde von 200 Zeichen auf 47 Zeichen reduziert. Das ist eine Reduktion von 76,5%!

Das gängigste Komprimierungsprogramm im Web ist GZIP und wird von allen modernen Browsern unterstützt. Es arbeitet nach dem „Deflate-Algorithmus“ und komprimiert Daten verlustlos.<sup>29</sup> Je länger eine Textdatei ist, umso drastischer kann sich die Komprimierung auswirken. GZIP ist meistens standardmäßig aktiviert. Falls nicht, kann GZIP mittels `htaccess` Eintrag von Listing 10 in Apache aktiviert werden.<sup>30</sup> Einträge in die `htaccess` Datei sollten nur dann eingesetzt werden, wenn man zum Beispiel ein Shared Hosting verwendet und dadurch keinen direkten Zugang zur Serverkonfiguration hat. Grund dafür ist, dass die Konfiguration von Apache mittels `htaccess` den Server verlangt.(Apache.org 2015)

```
1 <ifModule mod_gzip.c>
2 mod_gzip_on Yes
3 mod_gzip_dechunk Yes
4 mod_gzip_item_include file \.(html?|txt|css|js|php|pl)$
5 mod_gzip_item_include handler ^cgi-script$
6 mod_gzip_item_include mime ^text/.*
7 mod_gzip_item_include mime ^application/x-javascript.*
8 mod_gzip_item_exclude mime ^image/.*
9 mod_gzip_item_exclude rspheader ^Content-Encoding:.*gzip.*
10 </ifModule>
```

Listing 10: GZIP htaccess Eintrag

<sup>29</sup>Eine ausführlichere Beschreibung über den GZIP Algorithmus und der textbasierte Dokumentkomprimierung ist hier zu finden: <http://www.infinitepartitions.com/art001.html>. Für ein tiefere Verständnis bietet sich dieses Video von Google an: <http://tinyurl.com/mfxt5zt>

<sup>30</sup>Für andere Server wie zum Beispiel Nginx oder Node.js gibt es auf GitHub fertige Konfigurationen <https://github.com/h5bp/server-configs>

Mittels <http://www.feedthebot.com/tools/gzip/> lässt sich überprüfen, ob der eigene Server GZIP erlaubt.

## Gzip compression test

Gzip working? Yes reducing file size by 65.6%

Page size: 13,872 - Compressed size: 4,771 - Saving: 9,101 bytes

Abbildung 29: Testen von GZIP anhand von andreaslorer.de (Eigene Abbildung nach: feedTheBot.com)

Diesem Tool sollte aber nicht einfach nur blind vertraut werden. Es gibt Ressourcen welche explizit als Komprimierbar gekennzeichnet werden müssen. So sollte mittels „Chrome Developer Tool“ (oder vergleichbare Tools wie z.B. Firebug in Firefox) nachgesehen werden, ob alle Ressourcen auch tatsächlich mittels GZIP komprimiert werden. Abbildung 30 zeigt, dass die Datei: „Gallery.php“ trotz aktiviertem GZIP keine Komprimierung verwendete.



Abbildung 30: Die Serverantwort im JSON-Format ohne und mit GZIP

Der unterstrichene Wert ist dabei die Datengröße vor GZIP und der farblich hinterlegte Wert nach GZIP. Wie zu sehen ist, konnte die zu übertragende Datengröße um fast das 20-Fache verringert werden. Entsprechend drastisch sank auch die Zeit für das Herunterladen der Datei (blauer Balken). Wie konnte das erreicht werden? Um GZIP für json-Dateiformate zu ermöglichen ist nur dieser Eintrag in der entsprechenden PHP-Datei (hier in „Gallery.php“) nötig:

```
1     header('Content-Type: text/javascript');
2     header('Accept-Encoding: gzip');
3     ob_start('ob_gzhandler');
```

Manchmal sind es sehr kleine Dinge, die eine überaus große Wirkung haben können und die oftmals übersehen werden.

#### 4.3.7 „Keep-alive“ ermöglichen

Eine weitere Einstellung die in Apache vorgenommen werden kann ist „keep-alive“ und hat folgende Bedeutung:

*„Webpages are often a collection of many files and if a new connection (the brief communication) has to be opened for each and everyone of those files it could take significantly longer to display that webpage.*

*More officially the definition of HTTP keep-alive would be something like: a method to allow the same tcp connection for HTTP conversation instead of opening new one with each new request.“ (Sexton 2015a)*

```
1      ## keep-alive
2      <ifModule mod_headers.c>
3          Header set Connection keep alive
4      </ifModule>
```

Listing 11: htaccess Eintrag nach (Sexton 2015a)

Dieser Eintrag in der htaccess Datei fügt den `keep alive header` bei Serveranfragen hinzu.<sup>31</sup> Mittels Webpagetest kann überprüft werden, ob `keep-alive` auf dem Server aktiviert ist. Das Wiederverwenden von Verbindungen ist überaus wichtig, denn dadurch spart man sich den kompletten Verbindungsaufbau (three way handshake) ein.

#### 4.3.8 HTML5 Link Prefetching

Mit HTML5 gibt es ein neues Link-Attribut namens „prefetch“. Damit lassen sich Ressourcen herunterladen, die auf der aktuellen Seite noch nicht benötigt werden. Zum Beispiel in einem Bestellprozess lässt sich bereits bei Schritt 1 sagen, was auf der Seite von Schritt 2 benötigt wird. Dadurch lässt sich die Latenz und die Zeit zum Herunterladen in den Hintergrund verlagern, ohne das der Anwender davon etwas mitbekommt. Natürlich ist dies mit Bedacht einzusetzen und es macht wenig Sinn, große Mengen an Daten zu laden, die der Anwender vielleicht gar nie verwendet. Prefetch ist ganz einfach über den Link-Tag zu verwenden:

```
1      <!-- load a single ressource -->
2      <link rel="prefetch" href="http://your-resource-comes-here.jpg" />
3      <!-- or a full page -->
4      <link rel="prefetch" href="http://your-site/your-sub-site" />
```

Listing 12: Prefetching via Link-Tag

#### 4.3.9 Domain Sharding

Bei Domain Sharding werden die Seiten Ressourcen (Bilder, Javascript, CSS), auf verschiedene Domain Namen verteilt. Der Vorteil besteht darin, dass HTTP/1.1 die Anzahl an parallelen Downloads pro Domain Name limitiert, durch Domain Sharding kann künstlich die Anzahl an parallelen Verbindungen erhöht werden. Der Nachteil besteht in zusätzlichen DNS Lookups. Für Seiten mit vielen Ressourcen kann sich diese Methode aber durchaus lohnen.

<sup>31</sup>Bei Shared Hostings kann es trotz dieser Einstellung oft nicht erreicht werden, `keep-alive` zu aktivieren.

## 4.4 Bilder optimieren

„Ein Bild sagt mehr als Tausend Worte.“ Auch im modernen Web haben immer mehr und immer größere Bilder Einzug gehalten. Sie werden eingesetzt um eine Botschaft zu übermitteln, Emotionen zu erzeugen oder als **Eyecatcher**. In Abbildung 31 ist die durchschnittliche Seitengröße der Top 1000 Seiten<sup>32</sup> des Webs abgebildet. So beträgt die durchschnittliche Seitengröße (Abbildung 31 rechts) zum Zeitpunkt März 2015 rund 1843 kB. Davon sind 65% (1112 kB) Bildmaterial. Das linke Diagramm zeigt einen Aufwärtstrend der Seitengröße in Kilobyte über die Zeitspanne von einem Jahr. Dabei repräsentiert der Graph in Lila die schlechtesten 90% der Internetseiten, Grün die 10% der besten Seiten und Gelb stellt den Median dar. Wie zu sehen ist, werden die schlechtesten Seiten weiterhin schlechter, indem sie weiter an Kilobytes zulegen, auch der Median und die besten 10% sind über das Jahr hinweg leicht angestiegen.

Mit dem Optimieren von Bildern können sehr viele Kilobytes gespart werden.

*„Most sites fail to leverage best practices for optimizing images.*

*Despite the fact that images represent one of the single greatest performance challenges (and opportunities), 34% of pages failed to properly implement image compression, and 76% failed to take advantage of progressive image rendering.“ (Radware 2014, p. 4)*

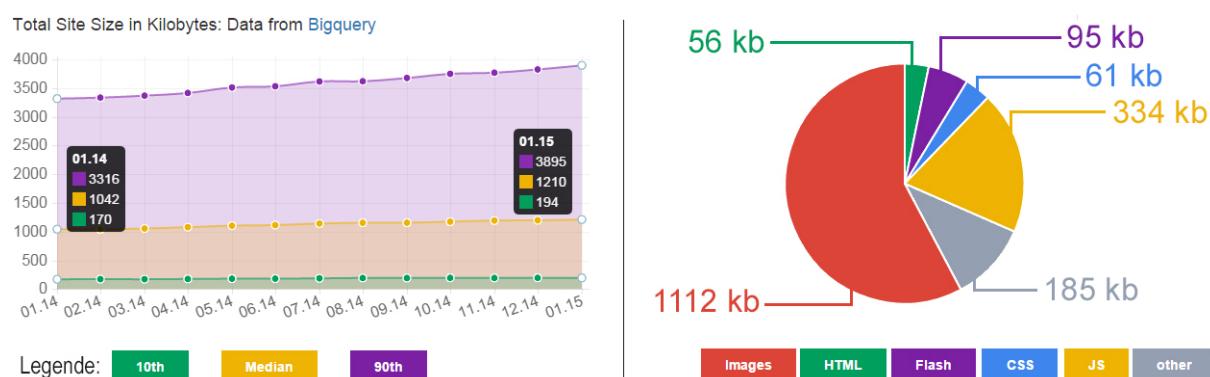


Abbildung 31: Seitenanalyse der populärsten Seiten des Webs (Eigene Abbildung - Daten via bigqueri.es: Gesamte Auswertung <http://tinyurl.com/o8vawxd> )

### 4.4.1 Progressive Image Rendering

Eigentlich eine sehr alte Technik ist das „Progressive Image Rendering“. Es kam längere Zeit außer Mode, Bilder als Progressive zu speichern. Der Trend hat sich aber wieder geändert und Progressive Images gilt als „Best Practice“ im Web. Sowohl Webpagetest als auch Google schlagen vor, Bilder Progressive an den Anwender auszuliefern. Testet man eine Seite via Webpagetest.org, so bekommt man unter dem Reiter „Compress Images“ beispielsweise solch eine Auswertung:

<sup>32</sup>Top 1000 Seiten nach dem Ranking von Alexa: (Alexa gehört zu Amazon und liefert umfassende Analysen über das Web <http://www.alexa.com/>)

Use Progressive JPEGs: 100/100

124.8 kB of a possible 124.8 kB (100%) were from progressive JPEG images

Abbildung 32 zeigt einmal den normalen Ladevorgang eines Bildes (unten) gegenüber dem „Progressive Rendering“ (oben).

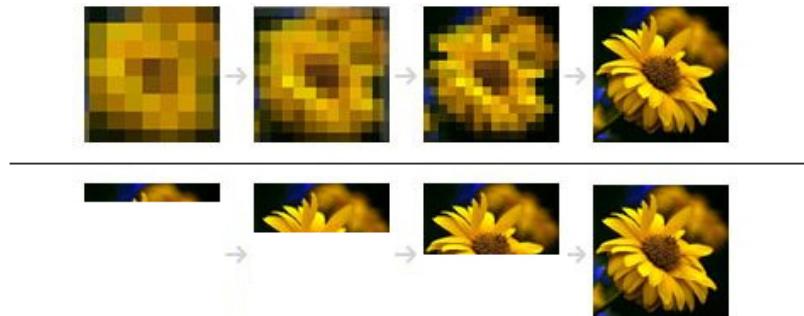


Abbildung 32: Gegenüberstellung von Progressive und normalem Laden

Nicht nur sind Progressive Images fast immer ein paar Kilobyte kleiner, sie erhöhen auch die **Perceived Performance** für den Anwender. Wo bei normalen Bildern lange Zeit eine weiße Lücke klafft, ist bei Progressive Images bereits sehr viel früher das volle Bild zu sehen. Das PNG Äquivalent für Progressive-Jpeg's ist „Interlaced“. Bilder sollten zum Beispiel in Photoshop über den Reiter: Datei -> für Web speichern... entweder Progressive oder als Interlaced gespeichert werden.

#### 4.4.2 Image Spriting

„Image Spriting“ bezeichnet das Kombinieren von vielen kleinen Bilddateien zu einer einzigen großen Datei. Mittels CSS lässt sich aus dem Bild dann das gewünschte Icon auswählen.



Abbildung 33: Image Sprite von verschiedenen Icons

Der Vorteil dieser Methode ist, dass nur ein Request benötigt wird um alle Icons die für die Seite benötigt werden, herunterzuladen. Der Nachteil ist, dass durch die größere Datei der Download länger dauern kann und sich so die Zeit für das erste Rendern verringert. Auch bei einer einzigen Änderung muss die ganze Datei (die im besten Fall bereits gecached wurde) neu heruntergeladen werden.

#### 4.4.3 Bild Komprimierung

Bei der Komprimierung von Bildern unterscheidet man zwischen zwei Arten: „Lossless“ und „Lossy“.

- Lossless bezeichnet die Komprimierung eines Bildes, ohne dabei die Qualität des Bildes zu verändern. So werden bei Lossless zum Beispiel die Metadaten eines Bildes entfernt, die von einer Kamera bei der Aufnahme hinzugefügt werden.
- Lossy ist die Reduzierung der Bildgröße auf Kosten von Qualität. Für die meisten Bilder ist diese Reduzierung aber durchaus legitim und oftmals sind die Einbußen mit bloßem Auge nicht zu erkennen. Lossy Komprimierung kann aber durchaus bis zu rund 40% der Bildgröße reduzieren.

Google hat ein eigenes Bildformat namens „Webp“ entwickelt und soll bis zu 30% der Bildgröße bei gleichbleibender Qualität einsparen. Allerdings ist dieses Format nur im Chrome Browser und Opera (der auf Chrome basiert) unterstützt.(caniuse.com 2015) Eine bessere Alternative ist moz-jpeg. Das ist ein von Mozilla entwickelter JPEG Bild Encoder, um die Bildgröße ohne Qualitätseinbußen zu verringern. Dabei bleibt das .jpg Format erhalten und ist somit überall einsetzbar.

*„For the short term Mozilla has developed MozJPEG — a modernized JPEG encoder that offers better compression while remaining fully standard-compliant, so it's compatible with all browsers, operating systems and native apps, and you can use it today without waiting for the whole world to upgrade“ (Lesiński 2014)*

Allerdings ist die Verwendung nicht ganz so einfach wie in diesem Zitat dargestellt und es ist das eigenständige Compilieren von C-Code<sup>33</sup> nötig, um dies auf dem eigenen Betriebssystem zu verwenden.

Glücklicherweise gibt es eine Webanwendung<sup>34</sup> mit der ganz einfach per „drag and drop“ Bilder mittels diesem Encoder komprimiert werden können. Je nach Bild lassen sich so mehrere Hundert Kilobyte einsparen (abhängig von der Qualitätseinstellung und Größe des Bildes).

Natürlich bietet auch Photoshop oder IrfanView (kostenlos) umfangreiche Möglichkeiten die Größe von Bildern zu verringern. Moz-jpeg schafft dies allerdings mit einem qualitativ besseren Ergebnis bei zudem kleinerer Bildgröße. Eine Verwendung sollte auf jeden Fall in Betracht gezogen werden.

#### 4.4.4 Responsive Images

„Responsive Images“ ist eine auf das Gerät des Endanwenders angepasste Auslieferung von Bildern. In vielen Webanwendungen sieht man Folgendes:

Das Bild wurde nicht nur viel zu Groß gewählt, es wird auch auf jedem Gerät, egal ob Tablet, Smartphone oder Desktop die selbe Anzahl an Bytes über die Leitung gesendet.

---

<sup>33</sup>Das Repository ist hier zu finden: <https://github.com/mozilla/mozjpeg> und eine Anleitung gibt es hier: <http://calendar.perfplanet.com/2014/mozjpeg-3-0/>

<sup>34</sup>Webanwendung zur Verwendung von moz jpeg: <https://imageoptim.com/mozjpeg>



Abbildung 34: Downsampling auf GitHub (Eigene Abbildung via Chrome Developer Tool)

Diese Methode nennt man **Downsampling**. Das bedeutet, dass nicht das Bild selbst verkleinert wird, sondern der Browser das Bild herunterlädt und dann auf die entsprechende Größe skaliert. Das kostet nicht nur Rechenleistung, sondern vor allem sehr viel unnötige Bandbreite und sollte unter allen Umständen vermieden werden.

Seit HTML 5 gibt es ein neues HTML Attribut namens **srcset**. Dieses Attribut ist leider noch nicht in allen Browsern implementiert, hat aber immerhin bereits eine globale Abdeckung von rund 50% (caniuse.com 2015). Für dieses Problem gibt es allerdings Abhilfe. Es gibt ein Polyfill<sup>35</sup> namens „Picturefill“<sup>36</sup>, dass genau diese Funktion für alle Browser zur Verfügung stellen kann. Nötig ist dazu nur das Herunterladen und Einbinden des Skripts in das HTML Dokument. Dadurch wird folgendes Listing möglich:

```

1   <picture id="hero-image">
2     <source srcset="someImg_lg.jpg" media="(min-width: 1367px)">
3     <source srcset="someImg_md.jpg" media="(min-width: 768px)">
4     <source srcset="someImg_xs.jpg" media="(min-width: 300px)">
5     <img srcset="fallback_md.jpg" alt="Some alt text">
6   </picture>
```

Listing 13: Srcset in Verwendung

Das bedeutet, dass Smartphones mit einer Breite von  $> 300\text{px} < 768\text{px}$  das Bild „someImg\_xs“ bekommen. Tablets mit  $768\text{px} < 1367\text{px}$  bekommen das mittlere Bild und alles was über 1367px ist bekommen die größte Version zu Verfügung gestellt. Falls der Anwender Javascript im Browser deaktiviert, ist es möglich, ein Fallback-Bild zu setzen (Listing 13: Zeile 5). Man kann sogar einen Schritt weiter gehen und auch das Bildformat entsprechend dem aufrufenden Gerät anpassen:

```

1   <picture id="hero-image">
2     <source srcset="someImg_lg.webp" type="image/webp" media="(min-width:
1367px)">
3     <source srcset="someImg_lg.jpg" media="(min-width: 1367px)">
4     <source srcset="someImg_md.webp" type="image/webp" media="(min-width:
768px)">
5     <source srcset="someImg_md.jpg" media="(min-width: 768px)">
```

<sup>35</sup> „Ein Polyfill [...] ist ein - meist in Javascript geschriebender - Code-Baustein, der in älteren Browsern eine neuere, von diesen nicht unterstützte Funktion mittels eines Workarounds nachrüsten soll. Beispielsweise sind Features von HTML5 in älteren Browsern nicht verfügbar. Webseiten können diese Funktionen trotzdem verwenden, wenn sie ein entsprechendes Polyfill mitliefern.“ (Wikipedia 2015c)

<sup>36</sup> Das offizielle Picturefill Projekt ist hier zu finden: <http://scottjehl.github.io/picturefill/>

```
6      <source srcset="someImg_xs.webp" type="image/webp" media="(min-width:
7          300px)">
8      <source srcset="someImg_xs.jpg" media="(min-width: 300px)">
9      <img srcset="fallback_md.jpg" alt="Some alt text">
10     </picture>
```

Listing 14: Srcset mit webp

Wie zu sehen ist, wurden 3 Zeilen eingefügt, die sich lediglich im Attribut (type="image/webp") unterscheiden. Das bedeutet, dass Anwender mit Chrome Browser das Chrome eigene Bildformat **webp** bekommen und alle anderen das normale jpg Format. Voraussetzung ist natürlich, all diese Bilder in ihren verschiedenen Auflösungen und Formaten anzulegen und bereit zu stellen, was einen Mehraufwand bedeutet. Picturefill ermöglicht es Down-sampling zu vermeiden und eine auf das Gerät angepasste Version des Bildes auszuliefern. Dadurch sinkt die Anzahl von Bytes die übertragen werden müssen enorm.

#### 4.4.5 Adaptive Images

Adaptive Images ist ein PHP-Skript<sup>37</sup>, das mit Hilfe einer htaccess Datei Bilder auf dem Server automatisch auf die verschiedenen Gerätegrößen zuschneidet. Ruft ein Anwender die Seite auf, prüft das Skript die Größe des Bildschirms und liefert anschließend das für ihn passende Bild aus.

### 4.5 Zusammengefasst

Durch Anwendung der Best Practices ist es möglich die Ladezeit drastisch zu verringern. Viele Einstellungen sind serverseitig einfach umzusetzen, anderes erfordert einen erheblichen Mehraufwand und mag sich für bestimmte Projekte nicht lohnen.

Für alle Seitenbetreiber die einen Root Zugriff auf ihren Server haben gibt es eine Apache oder auch Nginx Erweiterung von Google namens „Pagespeed Module“.<sup>38</sup> Dieses Modul kann einem die gesamte Handarbeit abnehmen (wie das Zusammenfügen von Scripts ect.) und führt viele der Best Practices automatisiert aus. Es lässt sich umfassend konfigurieren und bietet die Möglichkeit gewisse Regeln ein oder ausgeschaltet zu lassen. Mittels Webpagetest lässt sich eine Seite gegen dieses Modul testen. Dadurch kann man sehen wie groß die Ladezeitverringerung wäre, allein durch die Aktivierung dieser serverseitigen Erweiterung. Einen solchen Test lässt sich über diese URL ausführen: <http://www.webpagetest.org/compare>

<sup>37</sup>Mehr über dieses Skript ist hier zu finden: <http://adaptive-images.com/>

<sup>38</sup>Zu Pagespeed Module: <https://developers.google.com/speed/pagespeed/module>

## 5 Workflow

Im folgenden Abschnitt soll erläutert werden, wie ein moderner Workflow aussehen kann. Dabei sollen viele Aufgaben die oft noch per Hand erfolgen, automatisiert werden.

### 5.1 Nodejs & Node Package Manager

Nodejs ist eine auf Google Chromes Javascript Runtime aufbauende Plattform und ist unter <http://nodejs.org> kostenlos zu bekommen. Nodejs liefert einen eigenen Paket Manager namens „Node Package Manager“ kurz: „npm“.

Npm erlaubt das Installieren von Paketen. Ein Paket ist ein Programm, ein Framework (wie zum Beispiel Bootstrap) oder kann eine beliebige andere Art von Ressource sein, die über die npm-Umgebung zur Verfügung gestellt wird. Pakete können sowohl Global auf dem eigenen Rechner, als auch nur für ein bestimmtes Projekt installiert werden. Der Vorteil von einem Paket Manager wie npm liegt in der Zeitsparnis. Es lassen sich diese Pakete direkt per Befehl in die Kommandozeile eingeben. Dadurch spart man sich das Aufrufen der Webseite im Browser, als auch das Suchen, Herunterladen und Entpacken der gewünschten Datei.

Ein Beispiel: Das Paket „tmi - too many images“ ist ein kleines Programm, dass eine gegebene URL nach ihrer totalen Bildgröße analysiert und diese mit der durchschnittlichen Größe des Webs vergleicht. Es lässt sich mittels npm-Befehl über die Kommandozeile installieren. Danach lässt es sich ganz einfach per Befehl aufrufen:

```
C:\Users\Andi\Documents\workspace\bachelorthesis (master)
λ npm install -g tmi

C:\Users\Andi\Documents\workspace\bachelorthesis (master)
λ tmi andreaslorer.de

Your image weight
36.15 kB
Median mobile site image weight
17 kB
Median desktop site image weight
47 kB

On Mobile
-78 kB compared to sites in the 25th percentile
-403 kB compared to sites in the 50th percentile
-958 kB compared to sites in the 75th percentile

On Desktop
-233 kB compared to sites in the 25th percentile
-817 kB compared to sites in the 50th percentile
-1.67 MB compared to sites in the 75th percentile

Thanks for keeping the web fast <3
```

Es wäre aber auch möglich das unter Punkt 4.1.2 vorgestellte Google Pagespeed Insight, mittels „npm install -g psi“ (-g steht für: Global) zu installieren und per „psi someURL.de“

aufzurufen. Es gibt bereits mehr als 134,082 Pakete (Stand 21.03.2015), die verschiedenste Aufgaben erledigen können. Das Spektrum an Paketen ist sehr groß und reicht von sinnlosen Aufgaben (wie dem zufälligen generieren von Katzennamen<sup>39</sup>) bis hin zu hoch komplexen Programmen wie „sitespeed.io“<sup>40</sup>, das eine rekursive Performance-Analyse einer ganzen Webanwendung ermöglicht.

### 5.1.1 Dependency Management

Dependency Management bedeutet, dass ein Projekt oftmals auch noch von anderen Ressourcen abhängt. Das kann eine Library oder ein Framework wie Bootstrap sein. Dieses Framework seinerseits kann seinerseits wieder von externen Ressourcen abhängen. Im Falle von Bootstrap hängt dieses von JQuery ab. Der klassische Weg, wie solche externe Abhängigkeiten im Projekt geregelt werden, sieht wie folgt aus:

- Zu Beginn des Projekts werden Bibliotheken und Frameworks heruntergeladen.
- Es folgt das Entpacken und das Verschieben in das richtige Projektverzeichnis.
- Erscheint beispielsweise eine neue Version des Frameworks, beginnt dieser Prozess von neuem.

Dependency Manager (wie z.B. npm oder Bower) schaffen hier Abhilfe und vereinfachen diese Schritte. Durch Ausführen des Befehls: „npm init“ lässt sich eine neue Abhängigkeitsstruktur für ein Projekt anlegen. Dabei wird eine Datei mit dem Namen: „package.json“ und ein Ordner mit dem Namen „node\_modules“ erzeugt. In dieser Datei werden nun sowohl die Beschreibung, die Version, als auch alle Abhängigkeiten gespeichert. Will man nun beispielsweise „JQuery“ installieren, erfolgt das ganz einfach über das Kommando:

```
1 npm install -save jquery
```

Die `-save` Option gibt die Anweisung, dass folgender Eintrag in die package.json Datei aufgenommen werden soll:

```
1 {
2   "dependencies": {
3     "jquery": "^2.1.3"
4   }
5 }
```

`^2.1.3` bedeutet dabei, dass für dieses Projekt mindestens eine JQuery Version größer als 2.1.3 vorliegen muss. Der große Vorteil besteht nicht nun darin, dass weder ein Seitenaufruf getätigt, noch die Dateien entpackt und verschoben werden müssen. Zudem lassen sich über den Befehl „npm update“ alle Pakete auf die neueste Version bringen, ohne das der Vorgang des erneuten Herunterladen und Entpacken wiederholt werden muss. Die package.json Datei sollte in die Versionskontrolle des Projekts hinzugefügt werden. Der von npm angelegte Ordner: „node\_modules“ sollte dabei unbedingt per `.gitignore` von

<sup>39</sup>Cat-names: <https://www.npmjs.com/package/cat-names>

<sup>40</sup>Sitespeed.io: <http://www.sitespeed.io/>

der Versionierung ausgeschlossen werden. Lädt ein Teammitglied das Repository herunter, so muss nur noch „npm install“ aufgerufen werden und alle Abhängigkeiten, mit den für dieses Projekt verwendeten Versionsnummern, werden automatisch heruntergeladen und installiert. Damit ist jedes Teammitglied auf dem selben Stand und verwendet die selben Versionen. Neue Abhängigkeiten lassen sich so auch ganz einfach an alle Mitglieder verteilen.

### 5.1.2 Bower

Wie npm, so ist auch Bower ein Paket Manager. Um genau zu sein basiert Bower auf npm und lässt sich mittels „npm install -save bower“ für das Projekt installieren. Der Vorteil von Bower besteht darin, dass über eine `.bowerrc` Datei angeben werden kann, in welchem Ordner die Pakete installiert werden sollen. Dies ist bei npm nicht möglich.

Es lohnt sich Frameworks und Libraries wie z.B. `Bootstrap` oder `Underscore` mittels Bower zu installieren und andere Programme, wie zum Beispiel Gulp oder andere Nodejs Module, per npm.

Bower lässt sich, ähnlich wie npm, über das Kommando: „bower init“ Initialisieren. Danach lassen sich per Befehl „bower install -save bootstrap“ das Bootstrap Framework installieren.

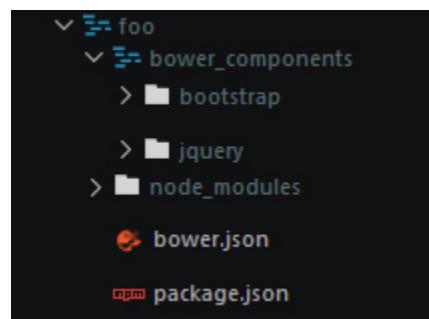


Abbildung 35: Projektstruktur mit npm und Bower

Wie in Abbildung 35 zu sehen ist, wurde nicht nur Bootstrap, sondern auch JQuery von Bower installiert. Das liegt daran, dass Bootstrap als interne Abhängigkeit wiederrum auch Abhängigkeiten besitzt. Diese werden bei der Installation von Bootstrap gleich mit installiert. Entfernt man Bootstrap wieder, so würde auch JQuery entfernt werden. Würde Bootstrap in einer neuen Version erscheinen, die von einer höheren JQuery Version abhängt, so würde Bower automatisch auch JQuery auf die nötige Version aktualisieren.

## 5.2 Gulp Task Manager

Warum und für was braucht es überhaupt einen Task Manager?

*If you aren't using productivity tools or task automation, you are working **too hard**. [...] Automation isn't about being lazy. It's about being **efficient**.* (Osmani 2014b, p. 18,78)

Ein Task Manager übernimmt immer wiederkehrende Arbeiten. Dazu zählt zum Beispiel die Aufgabe „minify, uglify und concatenating“ wie in Punkte: 4.3.3 bereits beschrieben. Aber auch das Übersetzen von „Sass“<sup>41</sup>, oder das Optimieren und Verkleinern von Bildern lässt sich als Task beschreiben und automatisieren. Die zwei bekanntesten Task Manager heißen **Gulp** und **Grunt**. Hier soll das Arbeiten mittels Gulp beschrieben werden, denn er ist sehr viel einfacher zu benutzen als Grunt.

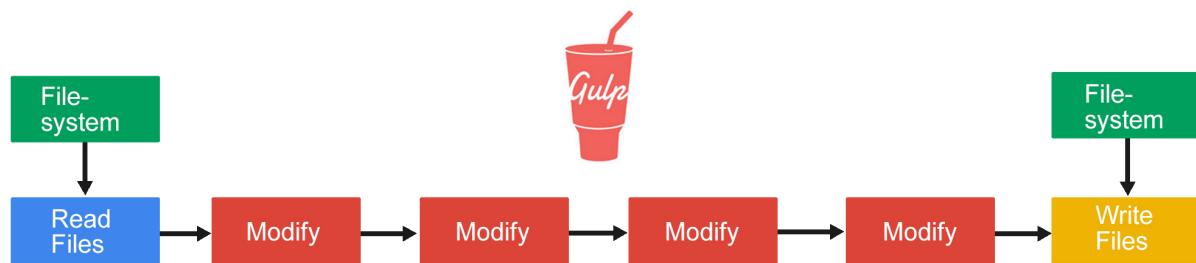


Abbildung 36: Gulp arbeitet nach dem „file stream“ Prinzip (Eigene Abbildung nach (Osmani 2014b, p. 85))

Gulp besteht nur aus 4 Kommandos:

1. `gulp.task`: Definiert einen Namen für einen Gulp Task. Dadurch wird dieser Task über die Kommandozeile benutzbar. Für Anwender, die wenig mit der Kommandozeile arbeiten wollen, gibt es auch eine Google Chrome Browsererweiterung namens Gulp-devtools. Damit lassen sich die verschiedenen Tasks ganz einfach über eine Benutzeroberfläche anklicken.
2. `gulp.src`: Greift auf eine oder mehrere Dateien zu. Auch Ordner können angegeben werden.
3. `gulp.dest`: Schreibt die Datei in den angegebenen Ort. Durch die Verwendung des \*-Zeichens, lässt sich eine Wildcard<sup>42</sup> vergeben. Mittels „images/\*.jpg“ lassen sich zum Beispiel alle jpg-Dateien aus dem Ordner „images“ auswählen.
4. `pipe`: Mittels „pipe“ lassen sich die Dateien (auch „file streams“ genannt) modifizieren. Dabei lässt sich pipe beliebig oft aufrufen. Die Aufgabe: „minify, uglify und concatenating“ könnte dabei so erfolgen:

<sup>41</sup>Sass is the most mature, stable, and powerful professional grade CSS extension language in the world. <http://sass-lang.com>

<sup>42</sup>Wildcard bezeichnet im Computer-Bereich einen Platzhalter für andere Zeichen

```
1 // require the gulp modules needed:  
2 var gulp = require('gulp');  
3 var uglify = require('gulp-uglify');  
4 var concat = require('gulp-concat');  
5  
6 // javascript task: concat, minify, uglify all javascript files in folder:  
7 gulp.task('javascript', function () {  
8     gulp.src(['site/assets/libs/*.js', 'site/assets/js/*.js'])  
9         .pipe(concat('bundle.min.js'))  
10        .pipe(uglify())  
11        .pipe(gulp.dest('dist/assets/js/'));  
12});
```

Dieser Task mit dem Namen „javascript“ holt nun alle Javascript Dateien aus dem Ordner „libs“ und „js“, fügt diese zu einer einzigen Datei zusammen und benennt sie „bundle.min.js“. Danach erfolgt das „uglify“ (beinhaltet das „minify“). Zum Schluss wird die Datei in den Ordner „dist/assets/js“ geschrieben. Die Ausgangsdateien werden dabei nicht modifiziert. Es sind auch so genannte „watch“ Tasks möglich, die bei einer Dateiänderung ausgeführt werden können. So kann der Browser immer dann neu geladen werden, wenn sich eine CSS Datei geändert hat. Dadurch wird ein manuelles neu Laden, um Änderungen zu betrachten, hinfällig.

Grunt hat gegenüber Gulp den Vorteil, dass es älter ist als sein Konkurrent. Dadurch gibt es viele Pakete, die nur mittels Grunt zur Verfügung stehen. Beispielsweise das Paket „grunt-responsive-images“. Damit lassen sich aus einem gegebenen Bild automatisch verschiedene Bildgrößen herausrechnen und abspeichern. Dies ist besonders hilfreich, wenn man „responsive images“ wie in Punkt: 4.4.4 beschrieben, verwenden möchte. Einen Blick auf Grunt kann sich also durchaus lohnen.

### 5.3 Yeoman

Ein Projekt wird oftmals manuell oder mittels eines Programms, wie zum Beispiel PHP-Storm oder Visual Studio erstellt. Es wird ein passender Projekttyp gewählt und das Programm erzeugt dann eine fertige Ordnerstruktur mit den wichtigsten Dateien. Zum Beispiel so:

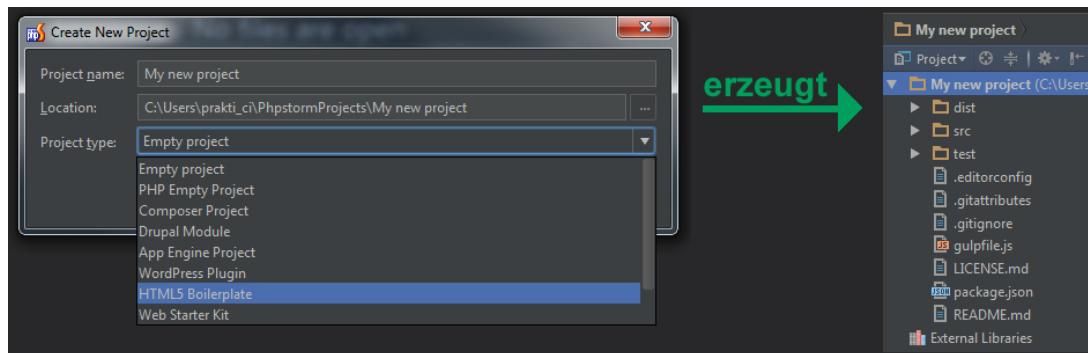


Abbildung 37: Projekt erstellen via Php-Storm (Eigene Abbildung)

<http://yeoman.io/> ist ein Tool für das schnelle Erstellen eines Grundgerüsts für ein Projekt.

*„Yeoman helps you to kickstart new projects, prescribing best practices and tools to help you stay productive.“*

*To do so, we provide a generator ecosystem. A generator is basically a plugin that can be run with the ‘yo’ command to scaffold complete projects or useful parts.*

*Through our official Generators, we promote the „Yeoman workflow“. [...] We take care of providing everything needed to get started without any of the normal headaches associated with a manual setup.“ (<http://yeoman.io>)*

Es stellt sogenannte „Generators“ zur Verfügung, die genau diese Funktionalität der Projekterstellung abbilden. Dabei wird das Ziel verfolgt „Best Practices“ für das jeweilige Projekt umzusetzen. Der Vorteil von Yeoman besteht darin, dass es eine enorme Vielzahl an Generatoren zur Verfügung stellt. So gibt es Generatoren für die Erstellung von Web-Apps, Chrome Browser Erweiterungen, Wordpress Blogs, Firefox OS Apps und noch tausend<sup>43</sup> weitere. Yeoman steht als npm Paket zur Verfügung und lässt sich mittels „npm install -g yo“ mit Hilfe der Kommandozeile installieren. Via „npm install -g generator-someName“ lassen sich dann beliebige Generatoren dazu installieren. Yeoman bringt dabei die zuvor vorgestellten Tools wie Gulp / Grunt und Bower zusammen. Durch das Ausführen wird das gesamte Projekt angelegt:

- Es werden alle npm Pakete (Bower, Gulp usw.) die nötig sind automatisch installiert.
- Die Ordnerstruktur mit einer CSS und Javascript Datei wird angelegt und im HTML Dokument eingebunden.

<sup>43</sup>Eine volle Liste an verfügbaren Generatoren ist hier zu finden: <http://yeoman.io/generators/>

- Nach der Installation stehen die wichtigsten Gulp Tasks zur Verfügung, ohne dass man sie selber schreiben muss.
- Durch Gulp besteht zum Beispiel die Möglichkeit, einen Webserver mittels Kommando „gulp serve“ zu starten.
- Es bestehen zusätzliche Funktionalitäten wie das automatische Aktualisieren des Browsers nach einer Änderung zur Verfügung.
- Mittels Eingabeaufforderung lässt sich auswählen welche Module (Sass, Bootstrap, Modernizr) enthalten sein sollen und welche nicht.

Das Ausführen von „yo gulp-webapp“:

```
C:\Users\prakti_ci\Documents\myProject
λ yo gulp-webapp

  _[--(o)--]_ | 'Allo 'allo! Out of the
  ( -`U` - ) | box I include HTML5
  / A \       Boilerplate, jquery, and
  | ~ |       a gulpfile.js to build
  .-. .-' Y '- your app.

? What more would you like?
>(•) Sass
  (•) Bootstrap
  ( ) Modernizr
```

Abbildung 38: Starten eines neuen Projekts mittels generator-gulp-webapp (Eigene Abbildung)

würde innerhalb von wenigen Minuten ein Projekt anlegen, das bereits fertig mit Bootstrap, JQuery und einem HTML Dokument konfiguriert ist. Mittels „gulp serve“ kann das direkt im Browser betrachtet werden:

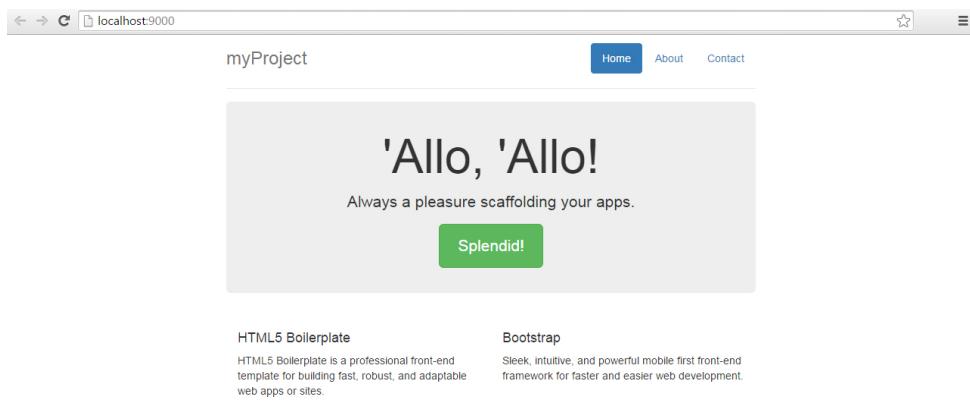


Abbildung 39: Ergebnis des Kommandos: „yo gulp-webapp“ (Eigene Abbildung)

### 5.3.1 Eigene Generatoren erstellen

Yeoman bietet jedem die Möglichkeit, einen eigenen Generator zu erstellen. Auch dafür gibt es wiederum einen Generator, den „generator-generator“. Damit lässt sich eine persönliche Projektstruktur erstellen, die den eigenen Projekten gerecht wird. Dadurch lassen sich beispielsweise Skripte zum verzögerten Laden von Javascript, von Beginn an in das eigene Projekt integrieren. Eigene Generatoren können ganz einfach veröffentlicht werden. Es muss lediglich ein Benutzer mittels „npm adduser“ angelegt werden und dann kann mittels „npm publish“ der eigene Generator als npm Paket zur Verfügung gestellt werden. Auch Updates lassen sich mittels „npm publish“ binnen Sekunden einspielen. Mein persönlicher Generator ist zum Beispiel unter: <https://www.npmjs.com/package/generator-4dev> zu finden und kann von jedem über „npm install -g generator-4dev“ installiert und anschließend mittels „yo 4dev“ aufgerufen werden.

Für ein Unternehmen könnte dies zum Beispiel bedeuten, mittels einem eigenen Generator eine einheitliche Projektstruktur einzuführen, der die Best Practices des Unternehmens abbildet. Alternativ könnte auch ein Konsens gefunden werden, welcher der bereits existierenden Generatoren für die jeweils eigenen Projekte einzusetzen ist.

Eine ausführliche Anleitung und Dokumentation zur Erstellung von Generatoren ist der offiziellen Seite: <http://yeoman.io/authoring/index.html> zu entnehmen.

## 5.4 Zusammengefasst

Gulp, Bower und Yeoman arbeiten perfekt zusammen und ermöglichen einen modernen Workflow. Mittels `npm` und `Bower` lassen sich Pakete ganz einfach installieren, deinstallieren und aktualisieren, während gleichzeitig eine einheitliche Entwicklungsumgebung bereitgestellt wird. Außerdem bieten viele Frameworks die Möglichkeit, mittels Bower, nur einzelne Module zu installieren. Dadurch ist es möglich nur das Nötigste in die Seite einzubinden.

Gulp ermöglicht eine umfassende Automatisierung von Aufgaben, die gerade für die Web Performance sehr wichtig sind. Folgende Projektstruktur erweist sich als sinnvoll:

```
1   .
2   |_ site/
3   | |_ bower_components/
4   | |_ index.html
5   | |_ assets/
6   |   |_ images/...
7   |   |_ js/
8   |   | |_ script_A.js
9   |   | |_ script_B.js
10  |   | |_ script_C.js
11  |
12  |   |_ css/
13  |   | |_ style_A.css
14  |   | |_ style_B.css
15  |   | |_ style_C.css
16  |   |_ ...
17  |
18 |_ dist/
19 | |_ bower_components/
20 | |_ index.html
21 | |_ assets/
22 |   |_ images/...
23 |   |_ js/
24 |   | |_ script_all.js
25 |
26 |   |_ css/
27 |   | |_ style_all.css
28 |   |_ ...
29 |
30 |_ node_modules/
31 | |_ Gulp
32 | |_ Bower
33 | |_ ..
34 |
35 |_ gulpfile.js
36 |_ package.json
37 |_ bower.json
38 |_ .gitignore
39 |_ ...
```

Listing 15: Projektstruktur

„./site“ ist dabei der Ordner, in dem die Entwicklung stattfindet. Der Ordner „./dist“ beinhaltet die für die Veröffentlichung angepasste Version von optimierten Bildern, verkleiner-

te und zusammengefügte CSS und Javascript Dateien und eine Kopie der restlichen, für die Webanwendung wichtigen Elemente. Diese werden automatisch mittels Gulp erzeugt (meistens durch einen sogenannten „build Task“) und lassen sich selbst bei kleinsten Änderungen an der Seite, mit nur einem Kommando neu erstellen. Durch diese Arbeitsweise ist es möglich, dass das Projekt von Beginn an einen für die Veröffentlichung optimalen Stand hat. Gulp bietet für fast alle Aufgaben eine Automatisierung und es lohnt sich ein Blick in die zahllosen Pakete<sup>44</sup>, die von einer immer weiter wachsenden Community bereitgestellt werden.

Durch Beachten des kritischen Rendering-Pfads (3.6), das Einsetzen der unter Kapitel 4.1 vorgestellten Tools und den in dieser Arbeit beschriebenen Optimierungsmaßnahmen, lässt sich eine für den Endanwender akzeptable Ladezeit erreichen.

---

<sup>44</sup>Gulp Plugin Suchmaschiene: <http://gulpjs.com/plugins/>

## 6 Ergebnis

Bereits zu Beginn des Projekts war es wichtig den Projektfortschritt messbar zu machen. Dafür wurde eine Testumgebung aufgebaut, mit deren Hilfe die Seite auf ihrer Geschwindigkeit getestet werden kann. Ganz entscheidend war dabei die Webpagetest API in Verbindung mit Google Spreadsheets. Damit lassen sich regelmäßig automatisierte Tests durchführen und die Daten werden nach erfolgreichem Test automatisch in einer Spreadsheet Tabelle gespeichert. Die über den Zeitraum der Arbeit hinweg gesammelten Daten sind hier zu finden: <http://tinyurl.com/15usz79>. Diese Daten wurden anschließend mittels <http://Chartjs.org> in Diagrammen aufbereitet und alle Diagramme sind auch online abrufbar unter: <http://bithugger.github.io/bachelorthesis/>

### 6.1 Wie wurde getestet?

Damit mittels Google Spreadsheets die Webpagetest API verwendet werden kann, ist es nötig, einen sogenannten API-Key anzufordern. Ein solcher Key ist kostenlos unter der Adresse: <http://www.webpagetest.org/getkey.php> zu erhalten und bietet die Möglichkeit täglich 200 Seitenaufrufe zu tätigen. Als Seitenaufruf zählt sowohl die „first view“ als auch „repeat view“. Die Tests sind 30 Tage abrufbar und gespeichert.

Für das Testen der Seite kann aus einer Vielzahl an Teststandorten gewählt werden. Damit lässt sich nachvollziehen wie beispielsweise die Ladezeiten aus der USA oder Asien sind. Je nach Zielgruppe sollten Tests von verschiedenen Standorten in betracht gezogen werden.<sup>45</sup> Für dieses Projekt wurden ausschließlich Teststandorte aus den USA und Europa gewählt.

Als Testparameter wurde eine Anzahl von 9 Tests pro Testlauf gewählt. Dabei wurde sowohl der „first view“ als auch „repeat view“ aufgezeichnet. Von den 9 Testläufen wurde der Median als Ergebnis des Testlaufs verwendet. Über den Zeitraum der Arbeit wurde die Seite 1089 Tests unterzogen.

Das größte Problem bestand in der möglichst genauen Messzeiterfassung für die Ladezeiten mittels Smartphone. Webpagetest stellt nur einen Smartphones Teststandort in Dulles USA zu Verfügung. Da die Latenz zwischen dem Hosting in Deutschland und dem Seitenaufruf in der USA sehr viel größer ist, als wenn dieser direkt aus Deutschland erfolgt, wurde nach einer Lösung gesucht diese Messungen exakter zu gestalten. Die Lösung dafür ist ein zweites Hosting mit der selben Seite in den USA zu erstellen. Dafür wurde die „Microsoft Azure Cloud“ verwendet. Eine kostenlose Testversion ermöglicht es ganz einfach auf verschiedensten Kontinenten eine Webseite zu hosten. Die Seite wurde für die Tests online geschaltet und nach den Testläufen wieder offline genommen.

---

<sup>45</sup>Eine volle Liste der zur Verfügung stehenden Teststandorte ist im Anhang unter Punkt: 10.1 zu finden.

## 6.2 Datenauswertung

Folgende Daten wurden bei jedem Testlauf erfasst: Speed Index, TTFB (ms), Render start (ms), Visually complete (ms), Dom Content loaded (ms), Site fully loaded (ms), Requests, Bytes in Document.

Das Diagramm in Abbildung 40 zeigt eine Übersicht der verschiedenen Messwerte über den Zeitraum des Projekts. Die Y-Achse bildet die Zeit in Millisekunden ab und die X-Achse ist das Datum der Messung. Innerhalb von 36 Tagen haben sich alle Werte signifikant verbessert.

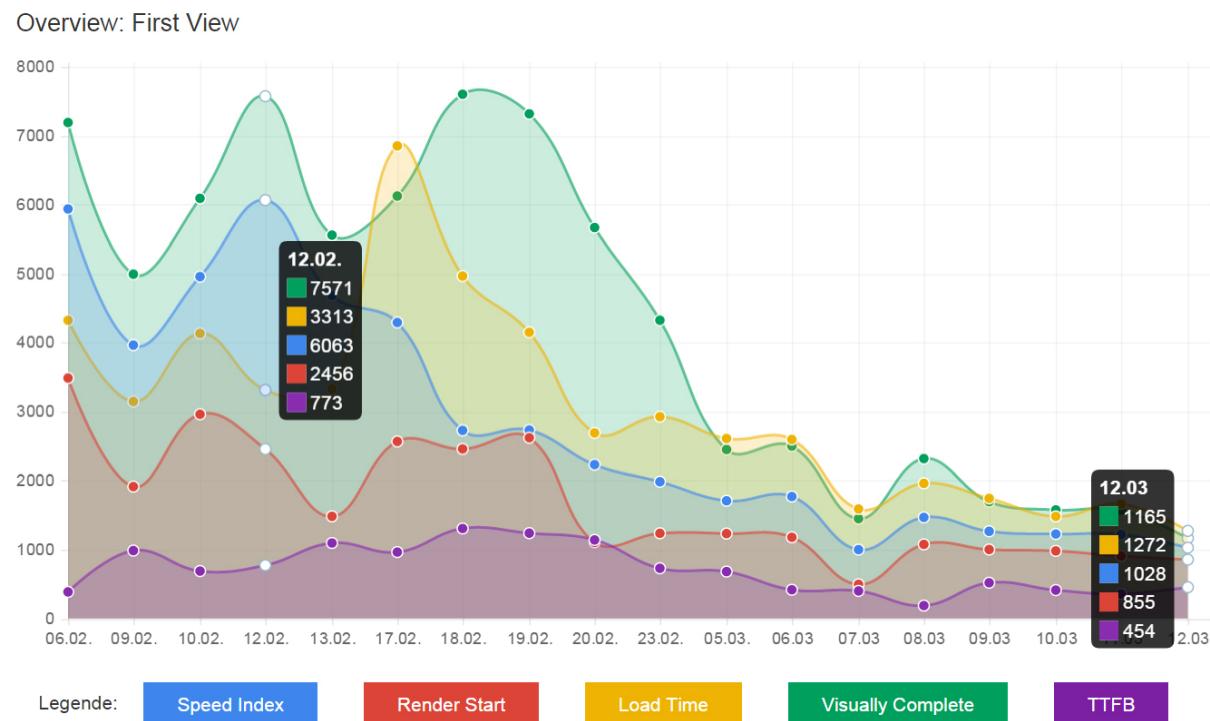


Abbildung 40: Datenauswertung - Überblick

- Speed Index: Ein Speed Index von < 1000 Punkten wird als „schnell genug“ angesehen. Der Median des Speed Indexes konnte von 6063 Zählern auf 1028 Zähler verringert werden.
- Render Start: Während dieser Wert bereits bei Projektbeginn mit 2,4 Sekunden für das erste Rendern recht gut war, konnte auch hier fast das dreifache der Zeit (287%) eingespart werden. Dieser Wert wird durch dieses Diagramm nicht drastisch genug ausgedrückt. So konnte die alte Version der Seite auf dem Smartphone durchaus einen **Render start** von ganzen 10 Sekunden haben. Dass dieser Wert im Diagramm vergleichbar gering ausfällt liegt daran, dass auch viele Tests mittels Desktop PC und Kabelverbindung eingeflossen sind. Die jetzige Renderzeit auf dem Smartphone beträgt ungefähr 1,4 Sekunden.

- Load Time bestimmt wie lange ein Anwender warten muss, bis eine Interaktion mit der Seite möglich ist. Dieser Wert konnte um volle 2 Sekunden verringert werden. Dies wurde vor allem durch eine Verringerung der Seitengröße erreicht.
- Visually Complete: Der höchste Messwert mit rund 7,6 Sekunden konnte auf 1,2 Sekunden verringert werden. Der Hauptgrund dafür ist die Priorisierung des Inhalts „above the fold“. Bei der alten Version erfolgte keine Priorisierung welche Bilder zuerst und welche zuletzt geladen werden sollen. Dadurch konnte es sein, dass Bilder die sehr weit unten platziert waren, zuerst geladen wurden, was die „Visually Complete“ Zeit erhöhte. In der neuen Version wird alles was unterhalb des „above the fold“ ist verzögert geladen.
- TTFB<sup>46</sup>: Ist die Zeit die vergeht, bis das erste Byte der Serverantwort den Browser erreicht. Eine Zeit von <200 ms sind dabei annehmbar, ein größerer Wert kann eine Vielzahl von Ursachen haben.

Die Ursache kann am eigenen Serversetup liegen (zu leistungsschwach, falsche Serverkonfiguration, langsame Verbindung zur Datenbank, langwierige Berechnungen etc.) und man kann darauf Einfluss nehmen oder aber das Problem liegt außerhalb des eigenen Einflussbereichs und es gibt Probleme beim Hosting Provider, das Routing zum Server ist Überlastet oder sonstige Netzwerkprobleme, deren Diagnose sich als sehr schwierig herausstellen kann. Die Wahl des richtigen Hosting Anbieters (zur Not wechseln) als auch Servers, mit genug Leistung für das entsprechende Anwendungsgebiet, sind entscheidend.

Die Seitengröße konnte in der Mobilen und Desktop Variante um rund 3/4 reduziert werden.

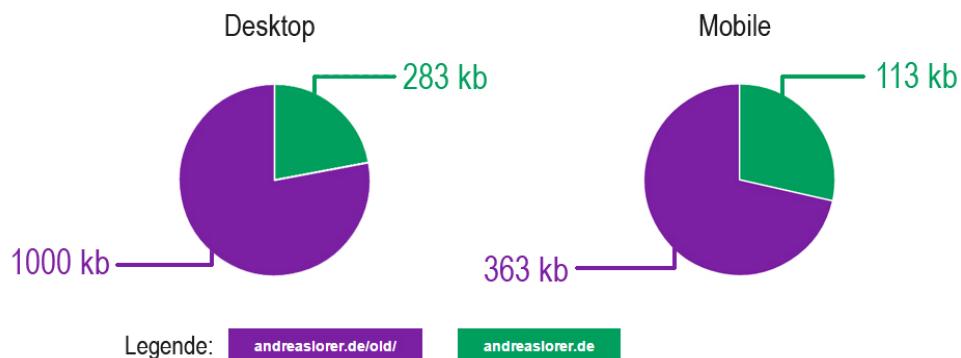


Abbildung 41: Seitengröße in Kilobyte

Dadurch sinkt für den Anwender nicht nur die Ladezeit, sondern auch sein Datenvolumen wird weniger in Anspruch genommen. Laut <http://whatdoesmysitecost.com/site/andreaslorer.de> kostet ein Seitenaufruf zwischen 0,01\$ und 0,04\$. Die Reduzierung wurde durch folgende Änderungen erreicht:

- Die „Best Practices“ wurden umgesetzt und der Server wie oben beschrieben entsprechend konfiguriert.

<sup>46</sup>Time To First Byte

- Bilder „below the fold“ werden erst dann geladen, wenn der Anwender dort hin scrollt.
- Bilder wurden umfassend komprimiert und **Progressive** abgespeichert.
- Das Framework wurde von Bootstrap auf **Pure.css** gewechselt.
- Verwendung von „Responsive Images“.
- Es werden mittels **Fontello** nur die Icons geladen, die auch in der Seite eine Verwendung finden.
- JQuery wurde als Abhängigkeit für die Bildergallery entfernt. Dadurch kann das herunterladen von JQuery soweit verzögert werden, bis alle wichtigen Teile der Seite geladen wurden.
- Scripts und sonstige Ressourcen wurden verkleinert und zusammengefasst. Abbildung 42 und 43 zeigen die Anzahl an Requests vor (links) und nach (rechts) der Optimierung für die Desktop- und Smartphone-Variante.

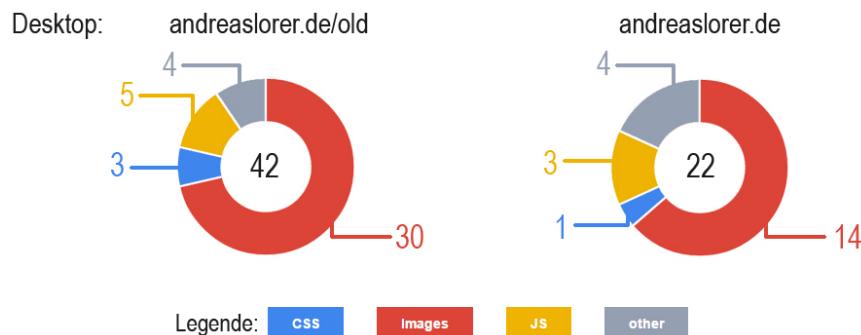


Abbildung 42: Anzahl an Requests via Desktop

Die Anzahl an Requests wurde in der Desktop Variante um die Hälfte verringert. In der Mobilen Variante konnten weitere Requests eingespart werden und es wird nur noch der Hintergrund und das erste Bild der Bildergallery geladen.

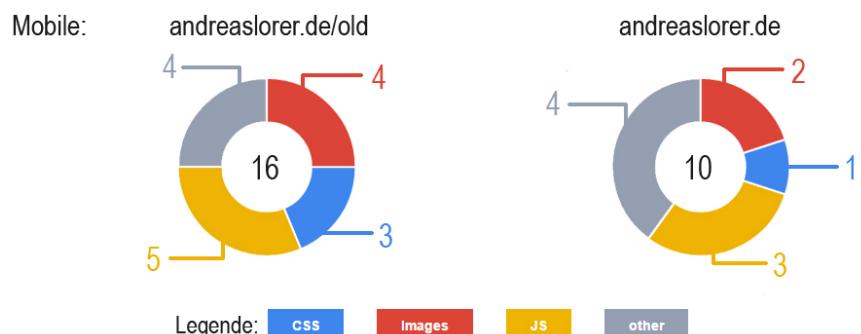


Abbildung 43: Anzahl an Requests via Mobile

Beim Seitenaufruf mittels Desktop-PC und Kabelverbindung können „first Render“ Zeiten von unter 189 ms und ein Speed Index von 468 erreicht werden.<sup>47</sup> Auf Smartphones mit 3G Netz und 300 ms RTT sehen die Werte etwas schlechter aus und so kann je nach Tag oder Uhrzeit das Ergebnis entsprechend anders ausfallen. Hier sind Werte von 1,3 Sekunden bis zum „first Render“ und einem Speed Index von 1948 möglich.<sup>48</sup>. Ein visueller Vergleich zwischen der alten und neuen Seite ist in Form eines Videos unter der URL: <http://tinyurl.com/o8xoy7m> zu sehen.

---

<sup>47</sup> Test Desktop: [http://www.webpagetest.org/result/150324\\_EW\\_105M/5/details/](http://www.webpagetest.org/result/150324_EW_105M/5/details/)

<sup>48</sup> Test Mobile: [http://www.webpagetest.org/result/150308\\_5V\\_JSD/4/details/](http://www.webpagetest.org/result/150308_5V_JSD/4/details/)

## 7 Der Weg zur Performance

Damit Webanwendungen überhaupt eine gute Performance bieten können gibt es eine fundamentale Bedingung: Sowohl das ganze Team als auch das ganze Unternehmen muss hinter dem Gedanken „Speed is feature number one“ (Holzle 2010) stehen. Diese Voraussetzung zu erfüllen und alle in ein „Boot“ zu bekommen, kann bereits sehr schwierig sein, ist aber für den Erfolg einer schnellen Webanwendung unabdingbar.

*„Performance more often comes down to a cultural challenge, rather than simply a technical one.“ (Kovalcin 2015, p. 13)*

*„If other designers and developers who shape the site aren't educated on performance, how can they make the best decisions about user experience? How can they weigh the balance between aesthetics and page speed? If they aren't empowered to make improvements, any performance champions will simply be playing cleanup after other people's work. Spending your time cleaning up other people's work (especially when it's preventable) is a one-way ticket to burnout.“ (Hogan 2014b)*

Damit dies gelingt, beschäftigt sich dieser Abschnitt mit der Frage wie der Weg zur Web Performance aussehen kann und welche Hürden es zu meistern gibt.

### 7.1 Hürden

Wenn man an das Thema Web Performance denkt so scheint es, dass dies ausschließlich für die Entwickler von Belangen ist.

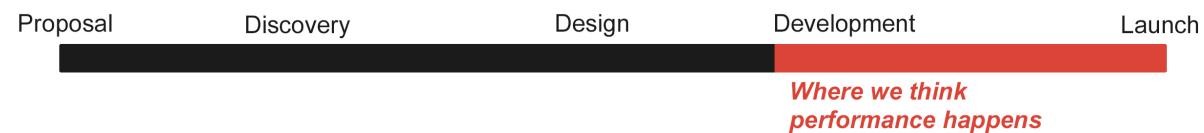


Abbildung 44: (Eigene Abbildung nach (Kovalcin 2015, p. 10))

Genau das Gegenteil ist allerdings der Fall. Wenn eine Webanwendung klassischerweise in die Entwicklungsphase geht, sind bereits die meisten Weichen gestellt. Das Design ist ausgearbeitet und kann mehr oder weniger mächtig ausfallen. Das Budget für das Projekt wurde vereinbart und der Umfang wurde vertraglich festgelegt. Das Projektmanagement muss darauf achten, dass die Wünsche und Erwartungen des Kunden erfüllt werden und die Entwicklung sich auf diesen Aspekt konzentriert.



Abbildung 45: (Eigene Abbildung nach (Kovalcin 2015, p. 11))

Deshalb beginnt das Thema Web Performance bereits ganz am Anfang. Der Kunde muss mit in das Boot geholt werden und es muss verdeutlicht werden, warum sich schnelle Ladezeiten lohnen, was für negative Auswirkungen langsame und was für positive Auswirkungen schnelle Ladezeiten auf den Endanwender haben. Der Kunde sieht oftmals nicht den Mehrwert an einer schnellen Webanwendung und möchte für etwas, das er nicht sehen kann auch kein Geld investieren. Argumente wie:

*„We surveyed 3000 users about 17 key product drivers. They rated speed 2nd most important only after easy to find content.“* (Patrick Hamann 2014, p. 8)

*„User load time expectations are 2 seconds or less and decreasing“* (Bixby 2013)

oder die in dieser Arbeit genannten Argumente können bei der Überzeugungsarbeit helfen.<sup>49</sup> Aktuelle Reports oder Zahlen und Fakten, die den direkten Zusammenhang zwischen Ladezeit und dem daraus resultierenden Profit verdeutlichen, sind entsprechend aufbereitet sehr hilfreich.

Der Kunde könnte auch argumentieren, dass er gar nicht so viele Mobile Anwender hat und es sich deshalb nicht lohnt. Dies war damals die selbe Argumentationsweise, die gegen „Responsive Webdesign“ sprach. *„Amongst the top 10,000 websites, almost 8% of websites went responsive within the span of a single year – an incredible growth!“* (Guypo.com 2014). Diese Argumentation hielt nicht lange und heute spricht jede Firma von Responsive, will eine Responsive Website oder hat zumindest darüber nachgedacht. (Guypo.com 2014) Web Performance ist ein Zukunftstrend, der mit der steigenden Anzahl an Smartphone Nutzern einher geht und damit früher oder später für jeden relevant ist. Ein guter „sales pitch“ könnte sein:

*„We'll provide you with a **fast**, responsive, immersive online experience.“* (Kovalcin 2015, p. 32)

Dem Kunden muss Performance als visuell greifbares Erlebnis präsentiert werden. Dabei sind nicht nur Diagramme, sondern auch Tools wie Webpagetest sehr nützlich. Damit können Vergleichsvideos erstellt werden. Denkbar wäre zum Beispiel ein direkter Vergleich mit den Konkurrenzseiten zu zeigen. Auch ein Vorher- / Nachervergleich eines erfolgreichen Projekts könnte dem Kunden präsentiert werden. Zum Launch kann dann ein Vergleich mit pre- und post-Performance aufgezeigt werden.

### 7.1.1 Projekt Manager

Für das Team der Projekt Manager ergeben sich laut Katie Kovalcin folgende Hürde: (Kovalcin 2015, p. 43)

- Verständnis für die Bedeutung von Web Performance.
- Stellvertreter und Verfechter von Web Performance gegenüber Kunden.
- Leistet Hilfestellung bei der Verwaltung eines Performance Budgets.

---

<sup>49</sup>Eine Sammlung mit Argumenten ist im Anhang unter Punkt 10.2 zu finden.

Zuerst müssen die Projekt Manager verstehen, warum Web Performance wichtig für das Projekt ist, um den Kunden entsprechend führen und beraten zu können. Des Weiteren muss jemand dafür Sorge tragen, dass das Performance Budget (dazu später mehr) eingehalten wird. Neue Anforderungen und Funktionen müssen dementsprechend bewertet und mit dem Kunden diskutiert werden.

### 7.1.2 Ästhetik anspruchsvolle Designer

„Ästhetik anspruchsvolle Designer“ können oft nicht abschätzen, was ihre Entscheidungen für einen Einfluss auf die Webanwendung haben. Die Voraussetzung dafür ist zu wissen, wie das Web funktioniert. Warum Webanwendungen langsam sind und was dazu führt. Erst dann lassen sich Entscheidungen treffen, die sowohl den Endanwender und sein Online Erlebnis, als auch den ästhetischen Anspruch zufrieden stellen. Wikipedia beschreibt den Begriff Design so:

*„Insbesondere umfasst Design auch die Auseinandersetzung des Designers mit der Funktion eines Objekts sowie mit dessen Interaktion mit einem Benutzer“* (Wikipedia 2015b)

Design ist also mehr als nur die visuelle Aufbereitung von Informationen, es muss in erster Linie dem Anwender dienen. Webseiten, die zu groß sind, zerstören den Ansatz von Web Performance bereits schon zu Beginn. Wie in Kapitel 2.9 gesagt, verlassen über 50% der Nutzer eine Seite nach einer Verzögerung von nur 3 Sekunden. Die emotionsvollsten Bilder und das prächtigste Design kann dem Besucher keine Botschaft vermitteln, wenn sie niemals zu sehen sein werden.

*„When you want to be fast, you have to give up the things slowing you down.“* (Osmani 2014a, p. 2)

Performance darf nicht als schlimmster Feind, sondern muss als bester Freund betrachtet werden. Dabei findet ein Balanceakt statt. Manchmal werden Entscheidungen zugunsten der Performance, ein anderes Mal für die Ästhetik getroffen. Der Schlüssel ist es, alle verfügbaren Informationen zu nutzen, um die richtigen Entscheidungen für sich und die Webanwendung zu treffen. (Hogan 2014a, p. 126)

Der Designer muss sich Fragen stellen wie: „Welchen Mehrwert hat der Nutzer durch dieses große Bild auf der Startseite“, braucht es 3 Schriftarten um einen gewissen Look & Feel zu vermitteln oder reicht eine? Gibt es eine alternative Schriftart die fast identisch aussieht, aber viel weniger Bytes benötigt?

Tabelle 1: Beispiel: Abwägung - Performance oder Ästhetik (Tabelle nach (Hogan 2014a, p. 126))

Question	Aesthetische Consideration	Performance Consideration
Can I put a large hero image at the top of every page?	Eye-catching represents the brand	This could be a really large file, we want to minimize page weight

Should I use three Font-Weights plus a text weight?	Lots of flexibility in typography	We want to reduce page weight and requests
Do I need a carousel on the landing page?	Showcases lots of different content	This adds a lot of page weight and additional requests. The user might never see the 2nd image.
How can I demonstrate the product functionalities?	Could use a GIF or embed a video	Videos and GIF's can be very heavy.

Die Antworten können je nach Projekt oder Designer unterschiedlich ausfallen.

Tabelle 2: Beispiel: Entscheidungen (Tabelle nach (Hogan 2014a, p. 127))

Question	Decision
Can I put a large hero image at the top of every page?	Yes, we use responsive images for the different screen sizes and compress them to reduce page weight! We might lose image quality.
Should I use three Font-Weights plus a text weight?	We need the 3 Font-Weights, they are used by the brand. No choice there
Do I need a carousel on the landing page?	No the extra images are not increasing the users experience.
How can I demonstrate the product functionalities?	We will use CSS-Animations instead of a GIF. This will cost some of the developers time.

Für viele Entscheidungsschritte macht es oft Sinn, dass Entwickler und Designer kollaborieren. Dabei können bereits zu einem sehr frühen Zeitpunkt Bottlenecks erkannt, Alterantiven vorgeschlagen und diskutiert werden. Dafür muss es klare Regeln geben. So haben Phrasen wie: „Das ist zu schwierig, gefällt mir nicht, dumme Idee, da gibt es keine Alternative, das Bild muss genau so aussehen“ generell nichts verloren. Das Wort „Performance“ darf nicht als **Trumpfkarte** für einen Entwickler gegenüber dem Designer gesehen werden. Viel mehr Sinn macht es Alternativen zu erarbeiten, Prioritäten zu diskutieren oder Lösungen aufzuzeigen. Ebenso kann es hilfreich sein, früh mit einem Mockup oder Prototyp auf Code-Basis anzufangen und gemeinsam an diesem auszuprobieren.

## Design



## Development

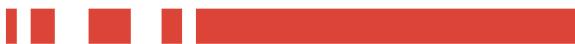


Abbildung 46: Projekt Zeitlinie

Auch hier gilt es den Balanceakt zwischen Ästhetik und Performance zu finden und nicht gegeneinander, sondern miteinander zu arbeiten. Um dem Ganzen einen Rahmen zu geben, in dem sich sowohl Designer, Entwickler als auch der Kunde bewegen dürfen, wird von vielen Performance Führsprechern ein sogenanntes „Performance Budget“ verwendet.

## 7.2 Performance Budget

Ein Performance Budget bedeutet das ein Wert gesetzt wird, der von der Seite nicht überschritten werden darf. Dieser Wert kann ganz simpel sein, wie die Ladezeit der Seite oder aber eine komplexere Metrik aus verschiedenen Werten.

*„The important point is to look at every decision, right through the design/build process, as something that has consequence. Having a pre-defined ‘budget’ is a clear, tangible way to frame decisions about what can and can’t be included, and at a suitably early stage in the project. It can also potentially provide some justification to the client about why certain things have been omitted (or rather, swapped out for something else).“*

Die Einführung eines Performance Budgets bietet den Vorteil, einen festgelegten Rahmen über die Zeitspanne des Projekts zu haben. Es ist ein Referenzpunkt, der mit darüber entscheidet welche und welche Komponenten nicht in die Seite mit einfließen können oder dürfen. Es funktioniert wie Spielgeld, das auf das Projekt aufgeteilt wird. Ist das Spielgeld leer, so gibt es diese 3 Regeln:

1. Optimiere eine existierende Funktion der Seite
2. Entferne eine existierende Funktion von der Seite
3. Füg es nicht hinzu

Das Motto heißt dabei: „Du kannst nicht ausgeben, was du nicht besitzt!“

### 7.2.1 Budget Metriken

Eine sehr einfache Budget Metrik wäre es, den Speed Index als Budget zu setzen. Jede Seite muss dann unter diesem festgelegten Wert bleiben. Oftmals ist es aber besser, verschiedene Werte als Budget zu setzen. Eine Kombination aus: Anzahl Requests, start Render, maximale Seitengröße, maximales Gewicht der Bilder und zusätzlich der Speed Index können eine sinnvolle und verständliche Basis bilden. Es können aber je nach Projekt auch anwendungsbezogene Metriken verwendet werden. So benutzt Twitter zum Beispiel die Metrik „Time to first Tweet“.

*„The most important metric we used was “time to first Tweet”. This is a measurement we took from a sample of users, of the amount of time it takes from navigation (clicking the link) to viewing the first Tweet on each page’s timeline. The metric gives us a good idea of how snappy the site feels.“ (Twitter 2012)*

### 7.2.2 Wie schnell ist schnell genug?

Wie wird das Performance Budget gesetzt? Dazu ist es erforderlich zu wissen, wie schnell eigentlich schnell genug ist? Schnelligkeit ist ein relativer Begriff.

*„In the high-speed world of automated financial trading, milliseconds matter. So much so, in fact, that a saving of just **6 milliseconds** in transmission time is all that is required to justify the laying of the first transatlantic communications cable for 10 years at a cost of more than \$300m between London and the New York Wall Street.“* (Williams 2011, vgl.)

Aber auch Google stellt fest, dass bereits der Einfluss von nur **100 - 400 ms** einen messbaren Einfluss auf die Anzahl an Suchanfragen (-0,2% bis -0,6%) pro Anwender hat.(Google 2009) Während 0,4% nach wenig klingen mag, so ist bei jährlich **2,2 Billionen** Suchanfragen (2013) wohl klar, was das für Google an Mehreinnahmen durch Werbung und Klicks bedeuten kann.(Statista.com 2014)

Studien haben ergeben, dass Menschen den Unterschied zwischen 2 Zeitspannen erst dann erkennen, wenn die Differenz 20% überschreitet.(Steve Seow 2009) Damit der Anwender eine Verbesserung gegenüber der Konkurrenz bemerkt, benötigt es in Folge dessen mindestens einen 20 prozentigen Unterschied. Anhand der Konkurrenzanalyse lässt sich ein Ergebnis bestimmen, mit dessen Hilfe das Performance Budget festgelegt werden kann. Bei einem Relaunch kann auch die alte Seite als Referenzpunkt dienen.

Zu beachten bleibt die Daten richtig zu interpretieren. Eine Seite die eine 7 Sekunden Ladezeit hat, muss nicht unbedingt langsamer sein (Stichwort: Perceived Performance) als eine Seite mit 5 Sekunden. Hier lohnt sich vor allem die Beachtung des Speed Indexes, der die Seite nach dem „visuellen Fortschritt über Zeit“ bewertet. Die 20%-Methode liefert den wohl minimalsten „schnell-genug“ Wert, der erreicht werden sollte um sich von der Konkurrenz abzusetzen.

Paul Irish, Mitarbeiter des Google Chrome Teams, sieht das ganze drastischer und gibt folgendes Statement:

*„My answer to how fast is fast enough? A Speed Index of under 1000. And for professionals that get there, they should shoot for delivering the critical-path view (above the fold) in the first 14kb of the page.“* (Irish 2014)

Für viele (vor allem im Bereich des E-Commerce) kann es sich rechnen, nicht nur schneller als die Konkurrenz zu sein.

### 7.2.3 Arbeiten mit einem Performance Budget

Im folgenden soll beispielhaft verdeutlicht werden wie das Arbeiten mittels Performance Budget aussehen kann.

Zu Projektbeginn wird das Performance Budget festgelegt. Das macht Sinn, denn wenn das Design der Webseite entsprechend mächtig erstellt wird, dann gibt es nachträglich nur noch wenig Möglichkeiten die Seite in den Budget Rahmen zu überführen, ohne dass erhebliche Überarbeitungsmaßnahmen nötig wären.

Ein Beispiel Budget kann aus folgenden Metriken bestehen:



Abbildung 47: Projekt Zeitline (Eigene Abbildung nach (Kovalcin 2015, p. 58))

- Maximale Seitengröße 600 kB
- Start Render 1000 ms
- Speed Index 1000

Die Seitengröße lässt sich nun aufteilen:

Tabelle 3: Seitengröße mit einem Budget von 600 kB

Content	Size
CSS	50 kB
Fonts	50 kB
Scripts	100 kB
Images	400 kB

Mit diesem Budget ist festgelegt, wie viel Spielraum jede Komponente hat. Will man 500 kB den Bildern zu Verfügung stellen, so muss in den anderen Bereichen eingespart werden. Dieses Beispiel verdeutlicht den Ansatz: „Du kannst nicht ausgeben, was du nicht besitzt!“ Nachdem das Budget festgelegt wurde ist sowohl das Design, die Entwicklung als auch das Projektmanagement damit beauftragt, das Budget beizubehalten.

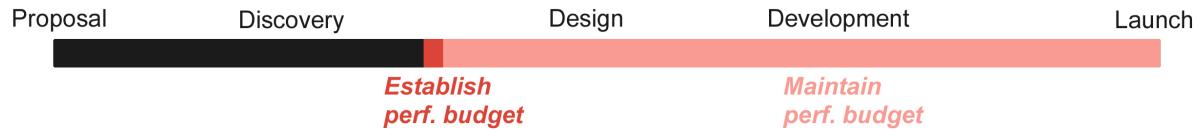


Abbildung 48: Projekt Zeitline (Eigene Abbildung nach (Kovalcin 2015, p. 59))

Dafür muss dem Kunden vermittelt werden, um was es sich bei diesem Budget handelt und warum es verwendet wird: (Aufzählung nach (Kovalcin 2015, p. 72))

- Was ist ein Performance Budget?
- Was ist das Performance Budget für dieses Projekt?
- Was sind die Bedingungen für dieses Budget?
- Warum wird es verwendet?
- Wie werden neue Funktionen hinzugefügt? (Optimieren, entfernen, nicht hinzufügen)
- Wieviel Budget hat jede Seite- / Unterseite

Für Designer (als auch Entwickler) ergibt sich ein Rahmen. Dies muss in keiner Weise schlecht sein und Dan Mall, Art Director und Designer sagt dazu folgendes:

*„I believe designers do their best work within constraints, **and knowing those constraints before starting a design can be incredibly enabling**. What I wouldn't give to know that I could use up to 10 images and 4 webfonts before starting a design! What a day that would be! Here's how to make that possible. [We define a Performance Budget]“*(Mall 2014)

Wenn diese Weichen für das Projekt gestellt wurden kann das Entwickler-Team die Anwendung im Sinne der Performance umsetzen.

Mittels Grunt (leider gibt es noch keine gute Gulp Alternative) lässt sich ein npm Paket namens „Grunt perfbudget“ installieren. Dieses ermöglicht es für die Seite einen Budget Task festzulegen. Dieser Task kann nun in den Build-Prozess der Seite integriert werden. Wird das Budget überschritten, so würde der Build-Task fehlschlagen.

```
Running "perfbudget:default" (perfbudget) task
>> -----
>> Test for http://cfarman.com    FAILED
>> -----
>> render: 2092 [FAIL]. Budget is 1000
>> SpeedIndex: 5426 [FAIL]. Budget is 1000
>> Summary: http://www.webpagetest.org/result/141107_HW_19HS/
Warning: Task "perfbudget:default" failed. Use --force to continue.

Aborted due to warnings.
```

Abbildung 49: Fehlschlagender Grunt Budget Task (Abbildung von (Farman 2014))

Das Setzen eines Performance Budget und das Arbeiten damit ist eine Herausforderung. Das Budget unterscheidet sich je nach Projekt und es kann schwierig sein die richtigen Metriken festzulegen. Dieser Abschnitt hat auch gezeigt, dass Web Performance nicht abhängig von einem einzigen Entwickler ist, sondern nur gelingen kann, wenn alle an einem Strang ziehen.

## 8 Ausblick

Sowohl für die Endanwender als auch für die Entwickler und Designer hat die Zukunft vor allem positive Änderungen parat. Das Http/1.1 Protokoll wird durch Http/2.0 abgelöst. Der LTE-Netzausbau schreitet weiter voran, deckt dabei eine immer größere Fläche ab und bringt den Nutzern damit eine niedrigere Latenz und eine höhere Bandbreite. Auf der diesjährigen CeBIT wurde der neue Mobilfunkstandard 5G von Vodafone präsentiert. Dieser steht noch in der Entwicklung, wird aber für das Jahr 2020 für den kommerziellen Einsatz vorhergesagt. 5G soll dabei Latenzen zwischen 1 bis 10 Millisekunden liefern und eine 100 mal höhere Bandbreite von bis zu 10.000 MBit/s zur Verfügung stellen. (lte-anbieter.info 2015) Ob diese Geschwindigkeiten auch wirklich innerhalb von nur 5 Jahren erreicht werden können, oder ob sie in erster Linie nur der PR dienen, bleibt dabei abzuwarten.

### 8.1 Http/2.0



Abbildung 50: Http 2.0 wurde veröffentlicht

Dieses Jahr ist es soweit und das Http/1.1 Protokoll aus dem Jahre 1999 wird durch Http/2 abgelöst. Natürlich verschwindet das alte Protokoll nicht von heute auf morgen. Http/1.1 wird noch viele Jahre erhalten bleiben, vielleicht auch nie gänzlich verschwinden. Dennoch sagt Daniel Stenberg, Mitglieder der Http/2 Work Group, voraus, dass bis Ende 2015 der Anteil an Http/2 traffic mehr als 10% beträgt (Stenberg 2015). Die großen Browser Chrome und Firefox haben in ihrer aktuellen Version Http/2 bereits aktiviert und auch der Internet Explorer soll mit Windows 10 (in der Technical Preview bereits aktiv) Http/2 unterstützen. (Microsoft 2014) Schlusslicht bildet Apple mit seinem Safari Browser, von denen bis dato (01.04.2015) noch keine Stellungnahme zu Http/2 verlautet wurde. Die zwei weitverbreitesten Server Apache und Nginx sind dabei, Http/2 zur Verfügung zu stellen. Apache hat bereits ein Http/2 Modul in der Alpha Phase. Nginx sagt, dass sie bis Ende 2015 Http/2 unterstützen werden.

Google (damit auch Youtube und andere Produkte die zu Google gehören) und Twitter haben bereits Http/2 seit einigen Monaten aktiviert. Http/2 bietet folgende Performance Verbesserungen:

- Parallel Multiplexing anstatt der Verwendung von Parallelen Verbindungen
- Verwendet nur eine einzige TCP Verbindung
- Server-Push

### 8.1.1 Http/1.1 Optimierungen in Http/2

Viele Optimierungs-Pattern für Http/1.1 stellen sich für das Http/2 Protokoll als Anti-Pattern heraus. Http/2 bringt den Vorteil, dass es genau eine TCP Verbindung benötigt. Damit wird Domain Sharding ein Performance Anti-Pattern für HTTP 2.0. Auch die Verwendung von Image Sprites, das Zusammenfügen von CSS und Javascript zu einer Datei ist ein Anti-Pattern wenn Http/2 verwendet wird. Eine Vielzahl von kleinen einzelnen Dateien werden zu keinem Performance Problem mehr durch Http/2.(Grigorik 2013f) „Server Push“ ist eine Mechanik die in Http/1.1 durch das Inlinen von CSS in das HTML Dokument bereits simuliert wird. De facto ist dies ein Workaround für eine Funktionalität die nun durch Http/2 bereitstellt wird. Durch Inlinen wird bei einer Anfrage gleich das CSS / Javascript als Antwort mitgesendet, das der Browser zur Darstellung des „above the fold“ Inhaltes braucht.

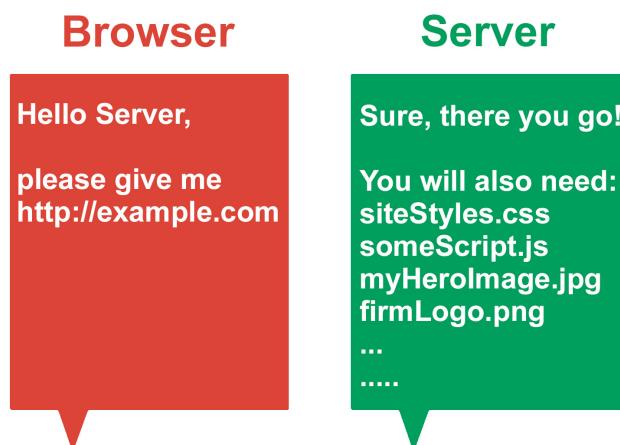


Abbildung 51: Veranschaulichung der Server Push Mechanik (Eigene Abbildung)

Durch Server Push wird genau dies möglich, ohne es inline in das Dokument zu schreiben. Dadurch sind die Ressourcen vom Browser im Cache speicherbar und können auch in den Unterseiten verwendet werden. Bereits bevor der Anwender die Seite besucht weiß man, welche Ressourcen ihm zur Verfügung gestellt werden müssen, um die Seite anzuzeigen. Diese Ressourcen lassen sich somit gleich auf die erste Anfrage als Antwort mitsenden und es wird damit das explizite Anfragen und Antworten eingespart.<sup>50</sup>

Durch Http/2 ergeben sich viele Vereinfachungen, aber auch neue Herausforderungen. Der Umstieg von Http/1.1 auf Http/2 wird nicht über Nacht geschehen. Für viele Seitenbetreiber wird es deshalb nötig sein, sowohl auf das alte, wie auch das neue Protokoll zu optimieren und an den jeweiligen Anwender die jeweils richtige Version auszuliefern. Dafür gibt es mehrere Ansätze und eine ausführliche Beschreibung gibt es hier: <http://tinyurl.com/phnq5d9>.<sup>51</sup>

<sup>50</sup>Dieses Video stellt eindrücklich die Auswirkung der Server Push Mechanik vor (Länge 5 Minuten): [https://youtu.be/4Ai\\_rrhM8gA?t=29](https://youtu.be/4Ai_rrhM8gA?t=29)

<sup>51</sup>Eine detaillierte Erklärung zu Http/2.0 ist dem PDF: http2 explained - Daniel Stenberg: <http://daniel.haxx.se/http2/http2-v1.11.pdf> zu entnehmen.

## 9 Fazit

Web Performance ist ein wichtiges Thema und gewinnt weiterhin an Bedeutung. Dieses Projekt hat gezeigt, dass durch Analyse der eigenen Webanwendung und mittels Anwendung der **Best Practices** sehr schnelle Ladezeiten erzielt werden können. Es wurde auch erklärt, dass dies einen nicht unerheblichen Mehraufwand bedeutet, der nicht durch eine einzelne Person gestämmmt werden kann, sondern sich innerhalb der gesamten Unternehmenskultur durchsetzen muss. Web Performance ist kein neues Thema, sondern wurde bereits sehr früh als eine Nummer Eins Priorität kommuniziert. Die Infrastruktur wird schneller, Browser werden weiterentwickelt und die Geräte werden leistungsstärker. Aber auch der Anwender wird dadurch immer verwöhnter und steigert seine Erwartungen an die Anwendung und gibt sich mit lägeren Wartezeiten nicht mehr zufrieden. Dies hat zur Folge, dass Web Performance nun auch für kleinere Seiten oder Produktanbieter ein wichtiger Aspekt wird, wenn das Angebot auch mittels Smartphone Akzeptanz finden soll. Es kann sich lohnen dem Anwender eine umfassende, schnelle Online-Erfahrung zu bieten. Auch Agenturen wie zum Beispiel Deep-Impact aus der Schweiz, springen auf den Zug der Web Performance auf und beginnen damit zu werben:

*„We value stability, security, **performance**, and flexibility. We also love new and shiny.“ <http://www.deep-impact.ch/en/>*

Es ist nicht ein Opfer das erbracht werden muss um den Endanwender glücklich zu stellen, sondern kann auch eine Chance sein, als „frist-mover“ den Trend zu erkennen und der Konkurrenz einen Schritt voraus zu sein.

## 10 Anhang

### 10.1 Webpagetest Teststandorte

Name	Standort
Eu-West, Chrome, Cable	ec2-eu-west-1:Chrome
Eu-West, Chrome, 3G	ec2-eu-west-1:Chrome.3G
Eu-Central, Firefox	ec2-eu-central-1:Firefox
Dulles, MotoG, Chrome	Dulles_MotoG:Motorola G - Chrome
Us-East, Chrome, 3G	ec2-us-east-1:Chrome.3G
Eu-Central, Chrome	ec2-eu-central-1:Chrome.3G
Eu-Central, IE_11	ec2-eu-central-1:IE 11
Us-East, IE11	ec2-us-east-1:IE 11
Us-East, Firefox	ec2-us-east-1:Firefox
Us-West, Chrome	ec2-us-west-1:Chrome
Us-West, IE11	ec2-us-west-1:IE 11
Us-West, Firefox	ec2-us-west-1:Firefox
Us-West-2, Chrome	ec2-us-west-2:Chrome
Us-West-2, IE_11	ec2-us-west-2:IE 11
Us-West-2, Firefox	ec2-us-west-2:Firefox
Ap-Northeast, Chrome	ec2-ap-northeast-1:Chrome
Ap-Northeast, IE_11	ec2-ap-northeast-1:IE 11
Ap-Northeast, Firefox	ec2-ap-northeast-1:Firefox
Ap-Southeast-1, Chrome	ec2-ap-southeast-1:Chrome
Ap-Southeast-1, IE_11	ec2-ap-southeast-1:IE 11
Ap-Southeast-1, Firefox	ec2-ap-southeast-1:Firefox
Ap-Southeast-2, Chrome	ec2-ap-southeast-2:Chrome
Ap-Southeast-2, IE_11	ec2-ap-southeast-2:IE 11
Ap-Southeast-2, Firefox	ec2-ap-southeast-2:Firefox
SA-East, Chrome	ec2-sa-east-1:Chrome
SA-East, IE_11	ec2-sa-east-1:IE 11
SA-East, Firefox	ec2-sa-east-1:Firefox

## 10.2 Argumentations Sammlung

Die folgende Sammlung an Argumenten ist der Seite: <http://blog.apakau.com/tag/web-performance/page/2/> entnommen. Dort befinden sich auch alle Links zu den Quellen.

“In 2014, the median top 100 e-commerce page takes 6.2 seconds to render its primary content, 10.7 seconds to fully load; i.e. 27% longer to begin rendering than it did in 2013”

“When faced with a negative mobile shopping experience, 43% of consumers will go to a competitor’s site next”

“Mobile users get frustrated after 1 second of delay. 500 ms delay increases user frustration by 26% and lowers engagement by 8%”

“47% of web users expect a page load of 2 seconds or less”

“57% of online users will abandon a website that takes more than 3 seconds to load”

“Shoppers remember online wait times as being 35% longer than they actually are”

“51% of online shoppers in the US say that site slowness is the top reason they’d abandon a purchase”

“64% of smartphone users expect pages to load in less than 4 seconds”

“32% of online consumers will start abandoning slow sites between 1 and 5 seconds”

“39% of users say speed is more important than functionality for most websites”

“18% of shoppers will abandon their cart if pages are too slow”

“37% of consumers said they would not return to a slow site, and 27% would likely jump to a competitor’s site”

“After experiencing a slow site, 14% of shoppers will begin shopping at another site, and 23% will stop shopping or walk away from their computer”

“80% of users will not return to a site after a disappointing experience. Of these, 37.5% will go on to tell others about their experience”

“64% of shoppers who are unhappy with their site visit will go elsewhere to shop next time”

“52% of online shoppers claim that quick page loads are important for their loyalty to a site”

“73% of mobile users said they’ve encountered a site that was too slow to load”

“Users who experience a 2-second site slowdown make almost 2% fewer queries, click 3.75% less often, and report being significantly less satisfied with their overall experience”

“A site that loads in 3 seconds experiences 22% fewer page views, 50% higher bounce rate and 22% lower conversion rate than a site that loads in 1 second”

“A site that loads in 5 seconds experiences 35% fewer page views, 105% higher bounce rate, and 38% lower conversion rate than a site that loads in 1 second”

“A site that loads in 10 seconds experiences 46% fewer page views, 135% higher bounce rate, and 42% lower conversion rate than a site that loads in 1 second”

“Issues with application performance are affecting overall business revenues by up to 9%”

“A 1-second delay in page response time decreases pages views by 11%, customer satisfaction by 16% and conversion by 7%”

“When pages are slow, business metrics suffer more now than they did a few years ago. For example, a page that took 6 seconds to load in 2010 suffered a -40% conversion hit vs. -50% hit in 2013”

“If Amazon increased page load time by 100 ms they would lose 1% of sales”

“Shopzilla achieved a 7–12% increase in conversion rate and a 25% increase in page views after a 5-second improvement in page load time”

“Shopzilla was able to support the same volume with 50% (402 to 200 nodes) less nodes, cutting server costs in half after a 5-second improvement in page load time”

“Facebook pages that are 500 ms slower result in a 3% drop-off in traffic, and 6% drop-off for 1 second”

“Search engines like Google recommend improving loading time when a site is slower than 95% of others”

“Yahoo saw a 5 – 9% drop in full-page traffic after increasing page load times by 400 ms”

“If Google increased page load by 500 ms, they get 25% fewer searches”

“AOL realized 160% increase in average number of page views from users in the top 10th percentile (fastest experience), compared to the bottom 10% (slowest experience)”

“Bing test results for a 2 second delay in page load time: Linear impact with increasing delay Time-to-click changed by double the delay User satisfaction dropped by 3.8% Re-

venue per user dropped by 4.3% Number of clicks dropped by 4.4%"

### 10.3 In Zusammenhang stehende Arbeiten

- Ilya Grigorik - High Performance Networking  
<http://chimera.labs.oreilly.com/books/123000000545/index.html>
- Tim Kadlec - Implementing Responsive Design  
<http://timkadlec.com/>
- Lara Callender Hogan - Designing for Performance  
<http://shop.oreilly.com/product/0636920033578.do>
- Scott Jehl - Responsible Responsive Design  
<http://abookapart.com/products/responsible-responsive-design>

## Literatur

- [Apa15] Apache.org. *When (not) to use .htaccess files.* <http://tinyurl.com/m6v5rut> [Aufgerufen am 17.03.2015]. 2015 (siehe S. 38).
- [Arc15] http Archive. *Mission.* <http://httparchive.org/about.php> [Aufgerufen am 12.03.2015]. 2015 (siehe S. 28).
- [Bar14] Bart. *Where is the best place to put <script> tags in HTML markup?* <http://stackoverflow.com/questions/436411/where-is-the-best-place-to-put-script-tags-in-html-markup> [Aufgerufen am 08.03.2015]. 2014 (siehe S. 19).
- [Bix13] Joshua Bixby. *Top ecommerce sites are 22 percent slower than they were last year.* <http://www.webperformancetoday.com/2013/03/27/top-ecommerce-sites-are-slower-than-they-were-last-year/> [Aufgerufen am 27.03.2015]. 2013 (siehe S. 62).
- [Bun14] Bundesnetzagentur. *Jahresbericht 2014.* <http://tinyurl.com/18r7f1v> [Aufgerufen am 19.03.2015]. Seite 78f. 2014 (siehe S. 10).
- [can15] caniuse.com. *Latest supported data.* <http://caniuse.com/> [Aufgerufen am 13.03.2015]. 2015 (siehe S. 31, 43, 44).
- [Far14] Catherine Farman. *Automate Performance Testing with Grunt.js.* <http://www.sitepoint.com/automate-performance-testing-grunt-js/> [Aufgerufen am 01.04.2015]. 2014 (siehe S. 68).
- [Goo09] Google. *Speed Matters.* <http://googleresearch.blogspot.de/2009/06/speed-matters.html> [Aufgerufen am 30.03.2015]. 2009 (siehe S. 66).
- [Goo10] Google. *Using site speed in web search ranking.* Website. 2010 (siehe S. 1).
- [Goo11] Google. *Creating Fast Buttons for Mobile Web Applications.* [https://developers.google.com/mobile/articles/fast\\_buttons](https://developers.google.com/mobile/articles/fast_buttons) [Aufgerufen am 03.03.2015]. 2011 (siehe S. 9).
- [Goo14a] Google. *Browser-Caching nutzen.* <https://developers.google.com/speed/docs/insights/LeverageBrowserCaching> [Aufgerufen am 16.03.2015]. 2014 (siehe S. 35).
- [Goo14b] Google. *Mobile Analyse in PageSpeed Insights.* <https://developers.google.com/speed/docs/insights/mobile> [Aufgerufen am 09.03.2014]. 2014 (siehe S. 17).
- [Goo15] Google. *Antwortzeit des Servers verbessern.* <https://developers.google.com/speed/docs/insights/Server> [Aufgerufen am 13.03.2015]. 2015 (siehe S. 34).
- [Gril3a] Ilya Grigorik. *Breaking the 1000ms Mobile Barriere.* [https://docs.google.com/presentation/d/1wAxB5DPN-rcelwbG06lC0us\\_S1rP24LMqA8m1eXEDRo/present?slide=id.g11c1373c5\\_3\\_0](https://docs.google.com/presentation/d/1wAxB5DPN-rcelwbG06lC0us_S1rP24LMqA8m1eXEDRo/present?slide=id.g11c1373c5_3_0) [Aufgerufen am 04.03.2015]. Slides. 2013 (siehe S. 11, 13).

- [Gri13b] Ilya Grigorik. *Breaking the 1000ms Mobile Barriere.* [https://docs.google.com/presentation/d/1wAxB5DPN-rcelwbG06lC0us\\_S1rP24LMqA8m1eXEDRo/present?slide=id.g11c1373c5\\_5\\_35](https://docs.google.com/presentation/d/1wAxB5DPN-rcelwbG06lC0us_S1rP24LMqA8m1eXEDRo/present?slide=id.g11c1373c5_5_35) [Aufgerufen am 04.03.2015]. Slides. 2013 (siehe S. 12).
- [Gri13c] Ilya Grigorik. *High Performance Browser Networking.* <http://tinyurl.com/p5dds9p> [Aufgerufen am 02.03.2015]. Chapter 2 Slow-Start. 2013 (siehe S. 6).
- [Gri13d] Ilya Grigorik. *High Performance Browser Networking.* <http://tinyurl.com/lz8t3mh> [Aufgerufen am 02.03.2015]. Chapter 10 Speed, Performance, and Human Perception. 2013 (siehe S. 9).
- [Gri13e] Ilya Grigorik. *High Performance Browser Networking.* <http://tinyurl.com/nojaxxa> [Aufgerufen am 02.03.2015]. Chapter 7 Table 7.1. 2013 (siehe S. 11).
- [Gri13f] Ilya Grigorik. *Removing 1.x Optimizations.* <http://chimera.labs.oreilly.com/books/1230000000545/ch13.html> [Aufgerufen am 01.04.2015]. 2013 (siehe S. 70).
- [Gri14a] Ilya Grigorik. *Codierung und Übertragungsgröße textbasierter Ressourcen optimieren.* <https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/optimize-encoding-and-transfer?hl=de> [Aufgerufen am 21.03.2015]. 2014 (siehe S. 38).
- [Gri14b] Ilya Grigorik. *HTTP-Caching.* <https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching?hl=de> [Aufgerufen am 16.03.2015]. 2014 (siehe S. 35).
- [gro14] growingwiththeweb.com. *async vs defer attributes.* <http://www.growingwiththeweb.com/2014/02/async-vs-defer-attributes.html> [Aufgerufen am 13.03.2015]. 2014 (siehe S. 31).
- [Gro15] Filament Group. *Defer Loading Javascript.* <https://github.com/filamentgroup/loadJS> [Aufgerufen am 13.03.2015]. GitHub. 2015 (siehe S. 31, 32).
- [Guy14] Guypo.com. *Responsive Web Design Adoption, 2014.* <http://www.guypo.com/rwd-2014/> [Aufgerufen am 27.03.2015]. 2014 (siehe S. 62).
- [Hog14a] Lara Hogan. *Designing for Performance.* O'Reilly, 2014. ISBN: 1-4919-0251-5 (siehe S. 63, 64).
- [Hog14b] Lara Hogan. *Performance Cops and Janitors.* <http://davidwalsh.name/performance-cops-janitors> [Aufgerufen am 27.03.2015]. 2014 (siehe S. 61).
- [Hol10] Urs Holzle. *Velocity 2010: Urs Holzle.* <http://tinyurl.com/px7m64m> [Aufgerufen am 27.02.2015]. Video from Velocity Conference. 2010 (siehe S. 1, 61).
- [Iri14] Paul Irish. *Fast-Enough.* <http://tinyurl.com/pdjmbp3> [Aufgerufen am 30.03.2015]. 2014 (siehe S. 66).
- [ItW15] ItWissen.info. *Shared Hosting.* <http://www.itwissen.info/definition/lexikon/Shared-Hosting-shared-hosting.html> [Aufgerufen am 26.02.2015]. 2015 (siehe S. 3).

- [Kov15] Katie Kovalcin. *The Path to Performance*. <https://speakerdeck.com/katiekovalcin/the-path-to-performance> [Aufgerufen am 27.03.2015]. 2015 (siehe S. 61, 62, 67).
- [Les14] Kornel Lesiński. *MozJPEG 3.0*. <http://calendar.perfplanet.com/2014/mozjpeg-3-0/> [Aufgerufen am 12.03.2015]. 2014 (siehe S. 43).
- [lte15] lte-anbieter.info. *Cebit 2015: Vodafone präsentiert 5G made in germany*. <http://www.lte-anbieter.info/lte-news/cebit-2015-vodafone-praesentiert-5g-made-in-germany> [Aufgerufen am 01.04.2015]. 2015 (siehe S. 69).
- [Mal14] Dan Mall. *HOW TO MAKE A PERFORMANCE BUDGET*. <http://danielmall.com/articles/how-to-make-a-performance-budget/> [Aufgerufen am 30.03.2015]. 2014 (siehe S. 68).
- [Mee14] Patrick Meenan. *Titel*. <http://tinyurl.com/o4b3rxh> [Aufgerufen am 11.03.2015]. blog. 2014 (siehe S. 25).
- [Mic14] Microsoft. *Http/2: The Long-Awaited Sequel*. <http://blogs.msdn.com/b/http/archive/2014/10/08/http-2-the-long-awaited-sequel.aspx> [Aufgerufen am 01.04.2015]. 2014 (siehe S. 69).
- [Osm14a] Addy Osmani. *CSS Performance Tooling*. <https://speakerdeck.com/addyosmani/css-performance-tooling> [Aufgerufen am 27.03.2015]. 2014 (siehe S. 63).
- [Osm14b] Addy Osmani. *Front-end Tooling Workflows*. <https://speakerdeck.com/addyosmani/front-end-tooling-workflows> [Aufgerufen am 21.03.2015]. 2014 (siehe S. 49).
- [Pat14] The Guardian Patrick Hamann. *Breaking news at 1000ms - Front-Trends 2014*. <https://speakerdeck.com/patrickhamann/breaking-news-at-1000ms-front-trends-2014> [Aufgerufen am 27.03.2015]. 2014 (siehe S. 62).
- [Rad13] Radware. *Radware Mobile Infographic*. Website. [http://blog.radware.com/wp-content/uploads/2013/11/Radware\\_SOTU\\_Fall\\_2013\\_Mobile\\_Infographic\\_Final1.jpg](http://blog.radware.com/wp-content/uploads/2013/11/Radware_SOTU_Fall_2013_Mobile_Infographic_Final1.jpg) [Aufgerufen am 15.01.2015]. 2013 (siehe S. 2).
- [Rad14] Radware. *State of the union - Ecommerce Page Speed and Web Performance*. <http://www.radware.com/assets/0/314/6442478110/c810eee1-e86f-438a-b82f-3ad002bf1c75.pdf> [Aufgerufen am 19.03.2015]. 2014 (siehe S. 8, 41).
- [Rit14] Michael Ritz. *Weltkarte in Schwarz*. <http://www.landkartenindex.de/kostenlos/?p=31> [Aufgerufen am 25.02.2015]. 2014 (siehe S. 4).
- [Sch15] Amy Schade. *The Fold Manifesto: Why the Page Fold Still Matters*. Techn. Ber. <http://www.nngroup.com/articles/page-fold-manifesto/> [Aufgerufen am 26.02.2015]. Nielsen Norman Group, 2015 (siehe S. 7).
- [Sex15a] Patrick Sexton. *Enable Keep Alive*. <http://www.feedthebot.com/pagespeed/keep-alive.html> [Aufgerufen am 17.03.2015]. 2015 (siehe S. 40).
- [Sex15b] Patrick Sexton. *Leverage Browser Caching*. <http://www.feedthebot.com/pagespeed/leverage-browser-caching.html> [Aufgerufen am 16.03.2015]. 2015 (siehe S. 35).

- [Sta14] Statista.com. *Anzahl der Suchanfragen bei Google weltweit in den Jahren 2000 bis 2013 (in Milliarden)*. <http://de.statista.com/statistik/daten/studie/71769/umfrage/anzahl-der-google-suchanfragen-pro-jahr/> [Aufgerufen am 30.03.2015]. 2014 (siehe S. 66).
- [Ste06] Guido Stepken. *Anti-Pattern in der Softwareentwicklung*. <http://www.little-idiot.de/teambuilding/AntiPatternSoftwareentwicklung.pdf> [Aufgerufen am 26.02.2015]. 2006 (siehe S. 4).
- [Ste08] Stoyan Stefanov. *Exceptional Website Performance with YSlow 2.0*. <http://de.slideshare.net/stoyan/yslow-20-presentation> [Aufgerufen am 27.02.2015]. Slide Nummer 4. 2008 (siehe S. 8).
- [Ste09] Ph.D. Steve Seow. *User Interface Timing Cheatsheet*. Techn. Ber. <http://tinyurl.com/nbl3cuz> [Aufgerufen am 30.03.2015]. Microsoft, 2009 (siehe S. 66).
- [Ste11] Stoyan Stefanov. *Book of Speed*. <http://www.bookofspeed.com/chapter3.html> [Aufgerufen am 02.03.2015]. siehe Abbildung 3.4 - The-three-way handshake. 2011.
- [Ste15] Daniel Stenberg. *The state and rate of http2 adoption*. <http://daniel.haxx.se/blog/2015/03/31/the-state-and-rate-of-http2-adoption/> [Aufgerufen am 01.04.2015]. 2015 (siehe S. 69).
- [t3n15] t3n. *Ist deine Website „mobile-friendly“? – Google steigert Druck auf Webmaster*. <http://t3n.de/news/google-mobile-friendly-589402/> [Aufgerufen am 25.02.2015]. 2015 (siehe S. 1).
- [Tak13] Dean Takahashi. *Apple's iPhone 5 touchscreen is 2.5 times faster than Android devices*. <http://venturebeat.com/2013/09/19/apples-iphone-5-touchscreen-is-2-5-times-faster-than-android-devices/> [Aufgerufen am 03.03.2015]. Grafik. 2013 (siehe S. 9, 13).
- [TNS14] Google TNS Infratest BVDW. *Global Connected Consumer Study*. Website. <http://www.netzproduzenten.de/wp-content/uploads/2014/08/global-connected-consumer-studie-deutschland.pdf> [Aufgerufen am 14.12.2014]. 2014 (siehe S. 2).
- [Ton13] Rmistry und Tonyg. *Loading measurement: alexa top million netsim*. <https://docs.google.com/document/d/1cpLSSYpqj4SprkJcVxbS7af6avKM0qc-imxvkexmCZs/edit> [Aufgerufen am 04.03.2015]. 2013 (siehe S. 10).
- [Twi12] Twitter. *Improving performance on twitter.com*. <https://blog.twitter.com/2012/improving-performance-on-twittercom> [Aufgerufen am 30.03.2015]. 2012 (siehe S. 65).
- [Web08] WebSiteOptimization.com. *The Psychology of Web Performance*. <http://www.websiteoptimization.com/speed/tweak/psychology-web-performance/> [Aufgerufen am 25.02.2015]. 2008 (siehe S. 1).
- [web15] webpagetest.org. *Speed Index*. <https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index> [Aufgerufen am 11.03.2015]. 2015 (siehe S. 25).

- [wha15] whatdoessitecost.com. *What Does My Site Cost?* <http://whatdoessitecost.com/site/www.hs-weingarten.de> [Aufgerufen am 12.03.2015]. 2015 (siehe S. 27).
- [Wik14] Wikipedia. *HTTP-Statuscode*. <http://de.wikipedia.org/wiki/HTTP-Statuscode> [Aufgerufen am 04.03.2015]. 2014 (siehe S. 14).
- [wik14] wiki.ubuntuusers. *Der Benutzer root*. <http://wiki.ubuntuusers.de/sudo> [Aufgerufen am 26.02.2015]. 2014 (siehe S. 3).
- [Wik15a] Wikipedia. *Content Delivery Network*. [http://de.wikipedia.org/wiki/Content\\_Delivery\\_Network](http://de.wikipedia.org/wiki/Content_Delivery_Network) [Aufgerufen am 04.03.2015]. 2015 (siehe S. 4).
- [Wik15b] Wikipedia. *Design*. <http://de.wikipedia.org/wiki/Design> [Aufgerufen am 27.03.2015]. 2015 (siehe S. 63).
- [Wik15c] Wikipedia. *Polyfill*. <http://de.wikipedia.org/wiki/Polyfill> [Aufgerufen am 19.09.2015]. 2015 (siehe S. 44).
- [Wik15d] Wikipedia. *Entwurfsmuster*. <http://de.wikipedia.org/wiki/Entwurfsmuster> [Aufgerufen am 26.02.2015]. 2015 (siehe S. 4).
- [Wil11] Christopher Williams. *The 300m dollar cable that will save traders milliseconds*. <http://www.telegraph.co.uk/technology/news/8753784/The-300m-cable-that-will-save-traders-milliseconds.html> [Aufgerufen am 30.03.2015]. 2011 (siehe S. 66).
- [Yah07] Yahoo. *Performance Research, Part 2: Browser Cache Usage - Exposed!* <http://yuiblog.com/blog/2007/01/04/performance-research-part-2/> [Aufgerufen am 16.03.2015]. 2007 (siehe S. 37).