

Bachelor-Thesis

Web Performance für den mobilen Endanwender

Zusammenfassung

19. März 2015

Columbus Interactive
Eywiesenstraße 6
88212 Ravensburg

Fakultät Elektrotechnik und Informatik
Studiengang Angewandte Informatik
Hochschule Ravensburg-Weingarten
Doggenriedstraße, 88250 Weingarten

Autor:

Andreas Lorer
andreas.lorer@hs-weingarten.de
88250 Weingarten
Wilhelmstraße 4



Bachelor-Thesis

Web Performance für den mobilen
Endanwender

Inhaltsverzeichnis

1 Einleitung	3
2 Motivation	3
2.1 Zielsetzung	4
3 Eigene Leistung	4
4 Ist-Zustand	5
5 Begriffe	5
5.1 Pattern und Antipattern	5
5.2 Latenz	5
5.3 Round Trip Time (RTT)	5
5.4 Http/1.1	5
5.5 TCP Three Way Handshake	6
5.6 TCP Slow Start	6
5.7 Content Delivery Network (CDN)	7
5.8 Above The Fold	8
5.9 Perceived Performance	9
6 Die 1000 ms Barriere	10
6.1 Touch Event	10
6.2 Netzwerke	11
6.2.1 Mobilfunknetz	11
6.3 Der HTTP-Request	12
6.4 Das Herunterladen einer 40 Kilobyte Datei	13
6.5 Zusammengefasst	14
6.6 Kritischer Rendering-Pfad	16
6.6.1 Rendering	16
6.6.2 Rendering-Pfad	17
6.6.3 Critical render path	18
6.6.4 Zusammengefasst	18
6.7 Analyse des Wasserfalls	19
7 Entwicklung	22
7.1 Tools	22
7.1.1 Google Chrome Developer Tool	22
7.1.2 Google Pagespeed Insight	22
7.1.3 Google Closure Compiler	22
7.1.4 Webpagetest	23
7.1.5 Pingdom	25
7.1.6 Speedcurve	25
7.1.7 Google Spreadsheet	25
7.1.8 Feed the Bot	25
7.1.9 What Does My Site Cost?	25
7.1.10 Critical Path CSS Generator	26
7.1.11 Http Archive	27
7.1.12 Perf Tooling Today	27
7.1.13 Twitter	27

7.2	Ausgangspunkt	28
7.2.1	Render Blocking Javascript	29
7.2.2	Render Blocking CSS	32
7.2.3	Inline von CSS	32
7.2.4	Ressourcen reduzieren	32
7.2.5	CSS-Bereitstellung optimieren	33
7.2.6	Antwortzeit des Servers reduzieren	33
7.2.7	Browser-Caching nutzen	33
7.2.8	Komprimierung aktivieren	36
7.2.9	„Keep Alive“ ermöglichen	37
7.3	Bilder optimieren	38
7.3.1	Progressive Image Rendering	38
7.3.2	Image Spriting	39
7.3.3	Bild Komprimierung	39
7.3.4	Responsive Images	40
7.3.5	Adaptive Images	42
7.4	Prozess der Validierung	42
8	Best-Practices	42
8.1	Serverseitig	42
8.1.1	Verringern von DNS Lookups	42
8.1.2	HTTP Requests	42
8.1.3	Caching	43
8.1.4	Pagespeed Mod	43
8.1.5	Images	43
8.2	Clientseitig	43
9	Workflow	43
9.1	Performanten Code schreiben	43
9.2	Minify und Uglify	43
9.3	Concatenating	43
9.4	Task Manager	43
9.5	Dependency Manager	43
9.6	Generators	43

1 Einleitung

2 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 Buffern eines Videos, beim Besuch einer Webseite oder beim Shoppen via APP. Wie oft wurde aus dem „nur mal eben diesen Begriff nachschlagen“ ein endloses Starren auf den weißen Bildschirm? Zu oft! 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“ kam schon bereits im Jahr 2008 auf 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 Smartphonenuutzter von 18% auf 50% an. Dies ist ein Wachstum von 32% innerhalb von nur 3 Jahren. (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 auch von der Nutzung eines Desktop-Pc's gewohnt ist. Diese Erwartungen werden von dem Großteil der im Internet besuchbaren Seiten 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.

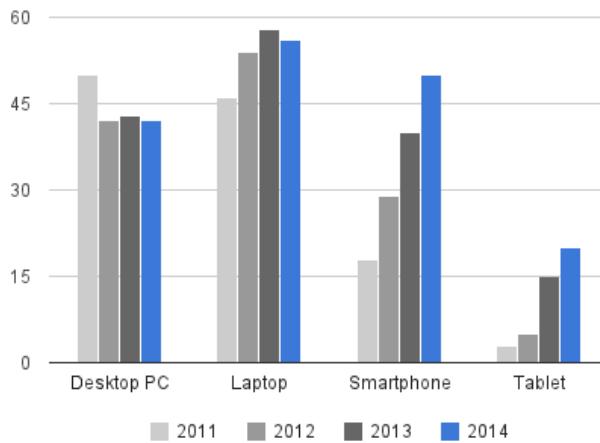


Abbildung 1: Gerätenutzung in der Gesamtbevölkerung (2011 – 2014)(TNS Infratest 2014)

2.1 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 von 10 Sekunden auf unter 1 Sekunden zu verringern. Mit Ladezeit ist dabei nicht die Zeit gemeint, die benötigt wird um die Website komplett zu laden, sondern die Zeit bis ein 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 Punkt 5.9 beschrieben.

3 Eigene Leistung

Meine Leistung besteht darin, einen Leitfaden zu erstellen, der einen Gesamtüberblick ermöglicht. Die Arbeit soll es dem Leser ermöglichen Fehler in der Struktur von Webanwendungen zu finden, die für die Geschwindigkeit hinderlich sind. Es soll herausgefunden werden, was die „Best Practices“ sind um die Ladezeit zu minimieren, wie ein moderner „Workflow“ aussehen kann, damit eine Webanwendung schon bei seiner Entstehung schnell ladet und im Projektverlauf schnell bleibt. Des weiteren soll erklärt werden, was für 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 solch einen Engpass ein schnelles Laden der Seite verhindern könnten.

4 Ist-Zustand

Die Webseite ist auf einem **shared Hosting**¹ aufgesetzt und antwortet auf ein Ping Kommando in rund 13ms. Dadurch, dass es keine Möglichkeit gibt **root Rechte**² auf einem shared hosting zu bekommen, können so manche serverseitige Einstellungen nicht durchgeführt werden. Diese werden dann zwar Aufgezeigt, aber kommen für dieses Projekt nicht zum Einsatz.

Die Website hat als Ausgangsbasis einen gängigen Aufbau. Sie besteht aus einer Bilder Gallerie basierend auf PHP und dem Bootstrap Framework.

5 Begriffe

5.1 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 durchgeföhrten, 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)

5.2 Latenz

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

5.3 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 100ms würde die RTT folglich 200ms betragen (Annahme: Hin- und Rückweg haben die selbe Zeit).

5.4 Http/1.1

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

¹Bei 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)

²Standardmäßig 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)

5.5 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 Daten zwischen Server und Browser ausgetauscht werden können, muss eine Verbindung aufgebaut werden. Abbildung 2 beschreibt den Prozess des Verbindungsauftauschs.³

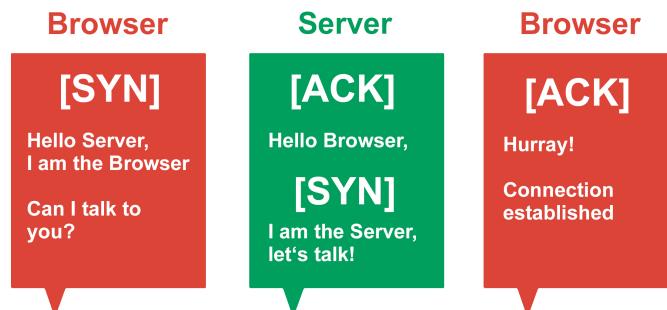


Abbildung 2: Three-Way-Handshake zum Aufbau einer TCP Verbindung zwischen Browser und Server (Eigene Abbildung nach: (Stefanov 2011))

5.6 TCP Slow Start

Ein Round Trip kann nicht beliebig viele Bytes transportieren sondern ist durch die sogenannte „Congestion Window Size“⁴ limitiert. Der Überbegriff für dieses Verhalten nennt sich „Slow Start“

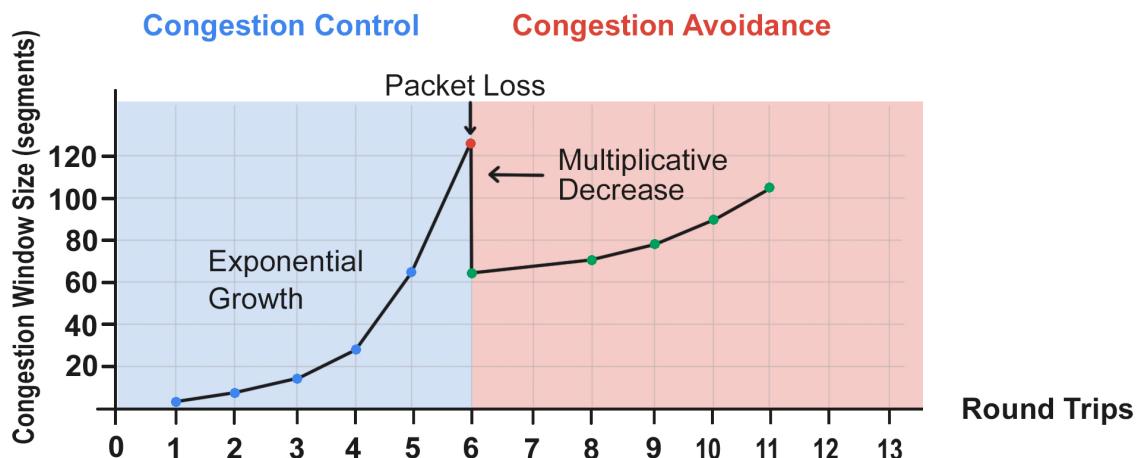


Abbildung 3: Congestion Control und Congestion Avoidance (Eigene Abbildung nach (Grigorik 2013c))

³Für ein tieferes Verständnis empfiehlt sich dieser Artikel: High Performance Browser Networking - Chapter 2: Building Blocks of TCP

⁴engl. congestion: Stauung, Überlastung, Anhäufung

- Congestion Control: Nach dem 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“ und verhindert das Aufstauen von Daten.
- Congestion Window Size: Diese Größe bestimmt, wieviel Bytes der pro Segmente geschickt werden darf, 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 davon ausgegangen, dass der erste Round Trip 4 Segmente senden darf. 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 Packetverlusten kommen kann ist dieser Aspekt besonders hervorzuheben, denn er verzögert das erreichen der maximal möglichen Datenrate.

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.

5.7 Content Delivery Network (CDN)

Ein Content Delivery Network (CDN), oder auch Content Distribution Network genannt, ist ein Netz lokal 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. Große CDNs unterhalten tausende Knoten mit zehntausenden Servern.(wikipedia 2015)



Abbildung 4: Schematische Darstellung eines CDN (Eigene Abbildung nach (Ritz 2014))

5.8 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 beim Aufrufen der Seite landen kann. Zum Beispiel wenn dieser an- oder abgemeldet ist. Paradebeispiel dafür sind Facebook oder Twitter.

5.9 Perceived Performance

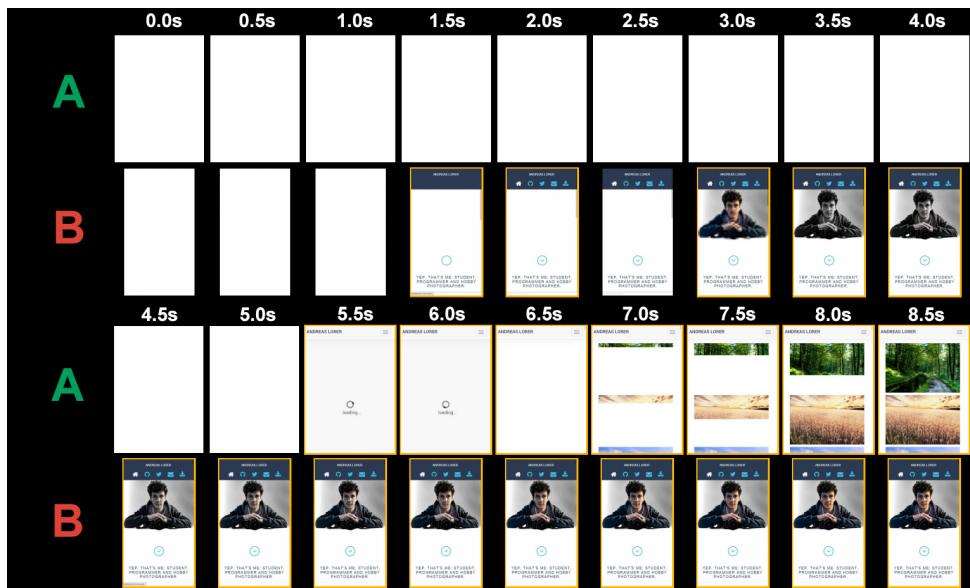


Abbildung 6: Zwei Seiten im Vergleich (Eigene Abbildung via webpagetest.org)

Abbildung 6 zeigt die Seiten A und B, mit nahezu identischer Ladezeit. Der Unterschied besteht darin, dass Seite B bereits nach 1.5 Sekunden eine erste visuelle Rückmeldung für den Anwender zu sehen ist, währendgegen Seite A erst nach 5.5 Sekunden dem Anwender zeigt, dass sie überhaupt ladet. „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 Infographik aufbereitet sind.

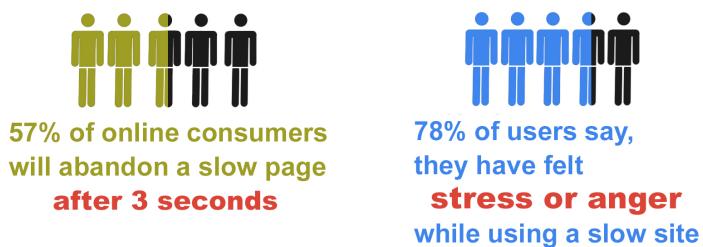


Abbildung 7: Einfluss und Effekt einer langsamen Seite auf den Anwender (Eigene Abbildung nach Daten von: (Radware 2014, p. 8))

Bereits kleine Verbesser- oder Verschlechterungen der Ladezeit können einen großen Einfluss auf den Anwender haben. Yahoo hat herausgefunden, dass wenn eine Seite um nur 400 Millisekunden schneller ist, sich der Traffic um 9% erhöhte.(Stefanov 2008) 57% der Online Konsumenten haben eine Seite, die länger als 3 Sekunden ladet wieder verlassen. 78% der Anwender empfinden sogar Zorn oder Stress wenn eine Seite nicht Ladet oder dies nicht ersichtlich ist.

6 Die 1000 ms Barriere

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

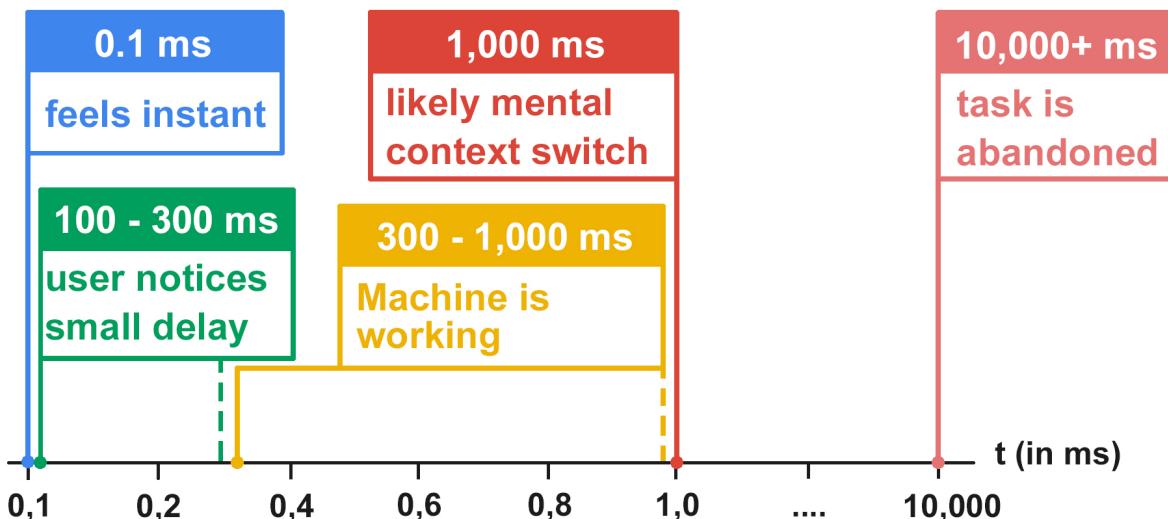


Abbildung 8: Zeit und Wahrnehmung durch den Anwender (Eigene Abbildung nach Daten von: (Grigorik 2013d))

Wie zu sehen ist bleiben gerade einmal eine Sekunde, 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“, dies wurde bereits in Punkt 5.9 „Perceived Performance“ angesprochen. Auf vielen Webseiten sieht man deshalb, Ladebalken oder sogenannte **Spinner**, die dem Anwender sagen, dass der Ladevorgang in Gang, aber noch nicht abgeschlossen ist.

Um das Ziel von einer Sekunde Ladezeit bis zum ersten Render zu erreichen, ist es nötig zu verstehen, womit die meiste Zeit beim Aufrufen einer Webanwendung verbracht wird. Bevor eine Seite mittels Smartphone vom Browser dargestellt wird läuft eine ganze Reihe von Prozessen ab.

6.1 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 123 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 Browsern 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.⁵

6.2 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, vergleicht man sie zum Beispiel mit der Latenz des 3G Netzes (Abbildung 10). Diese Studie kam zu dem Ergebnis, dass fast 70% der Ladezeit nur durch warten auf das Netzwerk verbracht wird:

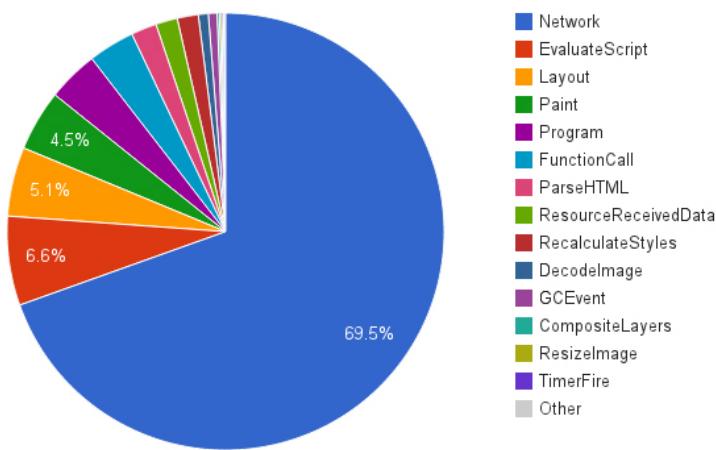


Abbildung 9: Untersuchung der top 1 Millionen Alexa Seiten (Abbildung von: (Tonyg 2013))

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

6.2.1 Mobilfunknetz

Es gibt unterschiedliche Mobilfunktsstandards mit denen Anwender Anbindung an das Internet erlangen. Aber selbst wenn einem Anwender 4G vom Mobilfunkanbieter versprochen wird, so ist die Netzaabdeckung mit 4G noch nicht vollständig deckend.

*„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:

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 weiter leiten. Die Zeit bis eine Authentifizierung erfolgt ist, kann je nach Anbieter und Mobilfunkstandard zwischen <100ms (LTE) und 2,5 Sekunden (3G) liegen

⁵ 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))

(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.

6.3 Der HTTP-Request

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

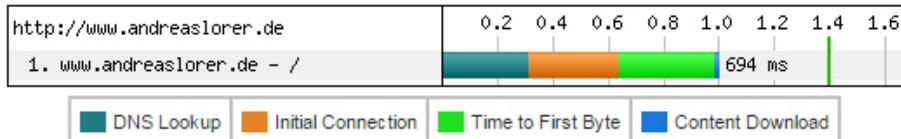


Abbildung 11: Anfrage der HTML-Datei von Irland mittels 3G Netz (Abbildung nach <http://webpagetest.org>)

- DNS Lookup: Um eine Verbindung mit dem Server herzustellen benötigt das HTTP Protokoll die IP Adresse des Ziels. Das heißt der DNS Server wird für den Namen „<http://andreaslorer.de>“ die zu diesem Namen zugehörige IP Adresse zurückgegeben.
- Initial Connection bezeichnet die Zeit die vergeht, bis eine neue Verbindung zum Server hergestellt wurde damit eine Kommunikation zwischen Browser und Server stattfinden kann. 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 beim Browser ankommt. Der Server muss den Request erst zusammenstellen bevor er ihn versenden kann. Dafür werden unter umständen Daten aus der Datenbank abgefragt oder es müssen berechnungen stattfinden. Diese Faktoren beeinflussen die TTFB und kann 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.

6.4 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 die Latenz die Geschwindigkeit einer Seite viel höher beeinflusst als die Bandbreite.

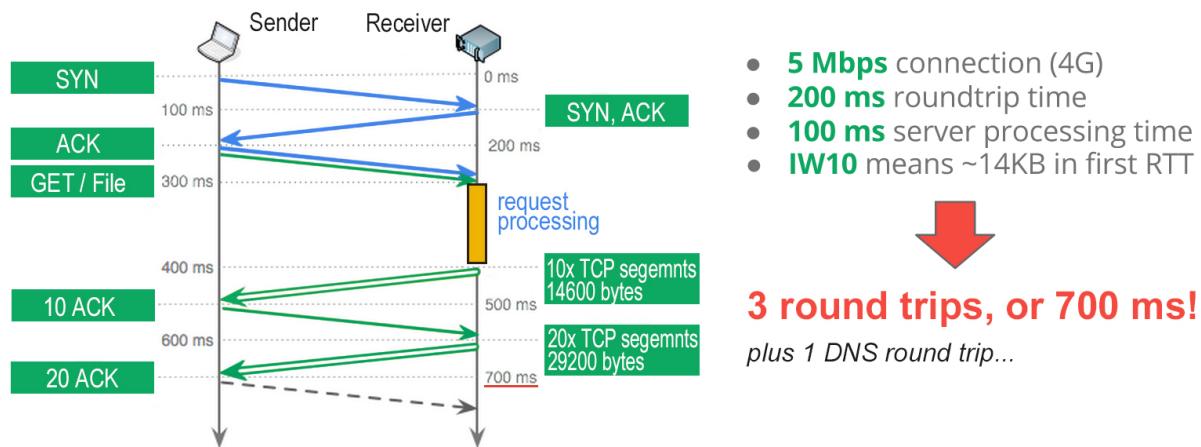


Abbildung 12: Herunterladen von 40kb mittels TCP (Abbildung nach (Grigorik 2013b))

Zuerst erfolgt der DNS Lookup, dann muss TCP per **three way handshake** eine Verbindung aufbauen. Dies kostet bereits 2 round trips, was in diesem Beispiel 400 ms entspricht. Durch den TCP slow start (siehe Punkt: 5.6) steht bei einer neuen TCP Verbindung nicht die volle 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. Wieviel round trips sind nötig, bis ich dem Browser Informationen übermittelt habe, so dass dieser etwas anzeigen kann? Idealerweise genau einer.

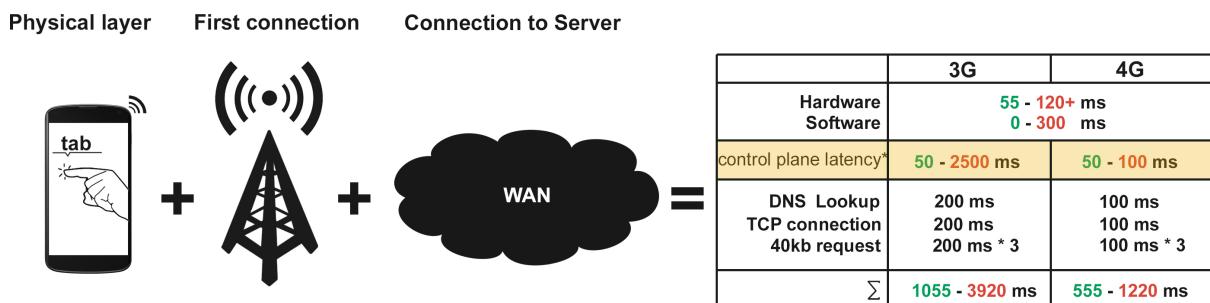


Abbildung 13: *control plane latency: Wenn noch keinerlei Verbindung zu einem Sendeturm aufgebaut wurde, entstehen einmalige Authentifizierungskosten (Eigene Abbildung nach Daten von: (Takahashi 2013)(Grigorik 2013a, p. 7, 12))

Wie in Abbildung 13 zu sehen ist bleibt von dem 1000 Millisekunden Budget nicht mehr viel übrig, 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 200 bis 500 ms haben kann und

hier schon der bestmögliche Wert von 200 ms verwendet wurde.

Für Nutzer des 4G schaut es besser aus und es bleiben 445 ms übrig.

Das bedeutet es müssen in den ersten Kilobytes, soviel nützliche Informationen vorhanden sein, damit der Browser bereits anfangen kann mit dem rendering zu beginnen, 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.

6.5 Zusammengefasst

Dieses Kapitel hat aufgezeigt, dass die Bandbreite eine nur 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 1000ms Barriere enorm erleichtert.
- Bei Anwendern die mittels 3G Netzwerk im Internet surfen besteht wenig Möglichkeiten eine Seite in unter einer Sekunde zu übermitteln, denn 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 unter Punkt ?? bla statt.
- Näheres Platzieren der Bits: Durch die Benutzung eines CDN's lassen sich Bits und Bytes näher am Endanwender platzieren 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. Nachteil: Wenn der Nutzer nie zu Schritt 2 oder 3 kommt, wurde das Internetvolumen des Anwenders (für die der Nutzer eventuell zahlt) umsonst belastet.
- 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 183 ms. Dies kann mehrere Gründe haben, so kann sich das Routing zum Server geändert haben, der Server kann ein Update erhalten haben oder die Maschine kann gewechselt worden sein. Was letzten Endes dazu geführt hat kann nicht genau gesagt werden.

- Senden von weniger Daten: Das schnellste Bit ist das, dass 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. Aus Abschnitt 5.4 wissen wir bereits, dass die Anzahl von parallelen TCP Verbindungen limitiert ist und Abschnitt 5.6 hat den Einfluss des TCP slow start gezeigt. Deshalb macht es Sinn, möglichst wenige Dateien auszuliefern, denn jede Datei benötigt einen extra TCP Verbindungsaufbau und jede neue TCP Verbindung unterliegt dem TCP slow start. Wie dies in der Praxis umgesetzt wird soll unter Punkt ?? konkretisiert werden.



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

- Stelle nützliche bytes zu Verfügung: Wie in Abbildung 12 zu sehen ist, erfolgt mit dem ersten round trip eine Datenübertragung von ca. 14 KB. Optimal ist es, wenn bereits mit dieser ersten Antwort genügend Informationen vorliegen um etwas auf den Bildschirm des Anwenders zu rendern. Das setzt voraus, dass die HTML Datei nicht größer ist als 14 KB (nach Kompression).
- 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)

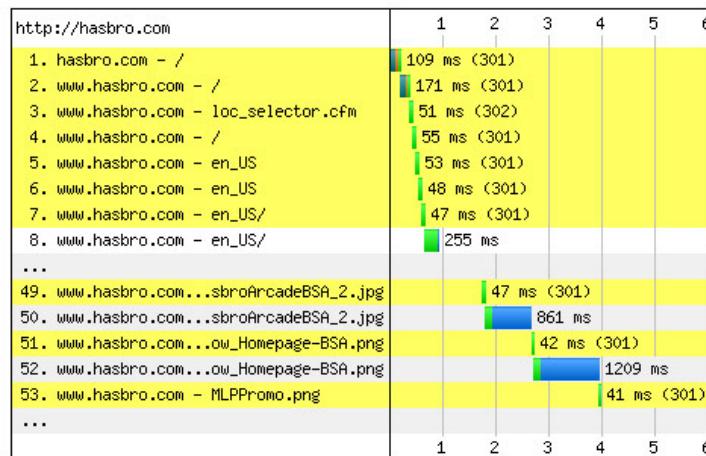


Abbildung 15: Testlauf von hasbro.com: „Dulles, VA USA - Thinkpad T430 - Chrome - Cable“ via webpagetest.org)

Wenn ein Anwender `hasbro.com` eingibt erfolgt der DNS Lookup und die TCP Verbindung wird aufgebaut. Der Browser erfährt dann, dass die Ressource unter einer anderen Adresse zur Verfügung steht. Danach erfolgt die DNS Auflösung für `www.hasbro.com` bei dem der gleiche Vorgang vonstatten geht. Es erfolgt die Weiterleitung auf die jeweilige Ländersprache von `www.hasbro.com`, wieder mit dem selben Vorgang. Nach rund 900ms 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 Weiterleitungen für den Smartphonenuutzer bedeuten können.



Abbildung 16: Testlauf mittels Smartphone von hasbro.com: „Dulles VA USA - Modell: MOTOG. 3G shaped 1.6Mbps / 300ms RTT“ Detaillierter Test unter: http://www.webpagetest.org/result/150310_AH_HVD/1/details/

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

Die Weiterleitung von `hasbro.com` auf `www.hasbro.com` ist für die Suchmaschinenoptimierung⁶ 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.⁷

6.6 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.

6.6.1 Rendering

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 be-

⁶engl. SEO - search engine optimization

⁷Mehr zu diesem Thema gibt es unter <http://www.sem-deutschland.de/seo-tipps/duplicate-content-definition/>

anspricht 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 besonderst das Auftreten des ersten Zeichnens interessant (engl. „First Paint Event“). Je früher das sogenannte **First Paint Event** auftritt umso höher ist die **Perceived Performance** der Webanwendung.

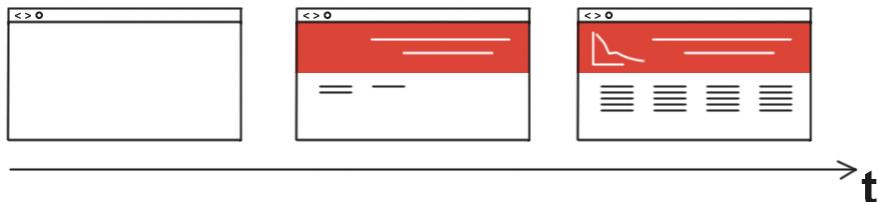


Abbildung 17: Der Render Prozess (Eigene Abbildung)

6.6.2 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.



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

Gegeben ist das folgende Beispiel einer simplen HTML Datei.

```

1  <!DOCTYPE html>
2  <meta charset="utf-8">
3  <title>Web Performance fuer den mobilen Endanwender</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 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 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-Baum⁸ 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 mit dem Rendern beginnen.

6.6.3 Critical render path

Critical render path sind genau die Javascript und CSS Dateien, die für den für das Rendern des **above the fold** (Punkt: 5.8) von nötig sind. Um dies umzusetzen ist es nötig, die Ressourcen in zwei Teile zu zerlegen: Für das Rendering **absolut** notwendig und nicht notwendig. Alle Dateien die nicht notwendig für das erste Rendern sind sollten so lange mit dem Laden verzögert werden, bis die Anwendung geladen ist. Wie genau so eine Umsetzung aussieht, wird in Punkt: ?? ausführlich gezeigt.

6.6.4 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 fortgefahrene 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 der Scripte vor den „</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 solange 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 Server Antwort mitgesendet. Dadurch muss der Browser die Datei nicht anfordern und heruntergeladen, 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

⁸Der DOM-Baum: <http://wiki.selfhtml.org/wiki/JavaScript/Objekte/DOM>

Beispiel Social Media Buttons (Facebook, Google+, Twitter ect.), Widgets oder Tracking Codes wie Google Analytics.

6.7 Analyse des Wasserfalls

Für ein besseres Verständnis des Kritischen Rendering-Pfads soll ein praktisches Beispiel einer nicht optimierten Seite helfen. In Abbildung 19 ist ein Ausschnitt eines Wasserfallmodells dargestellt.

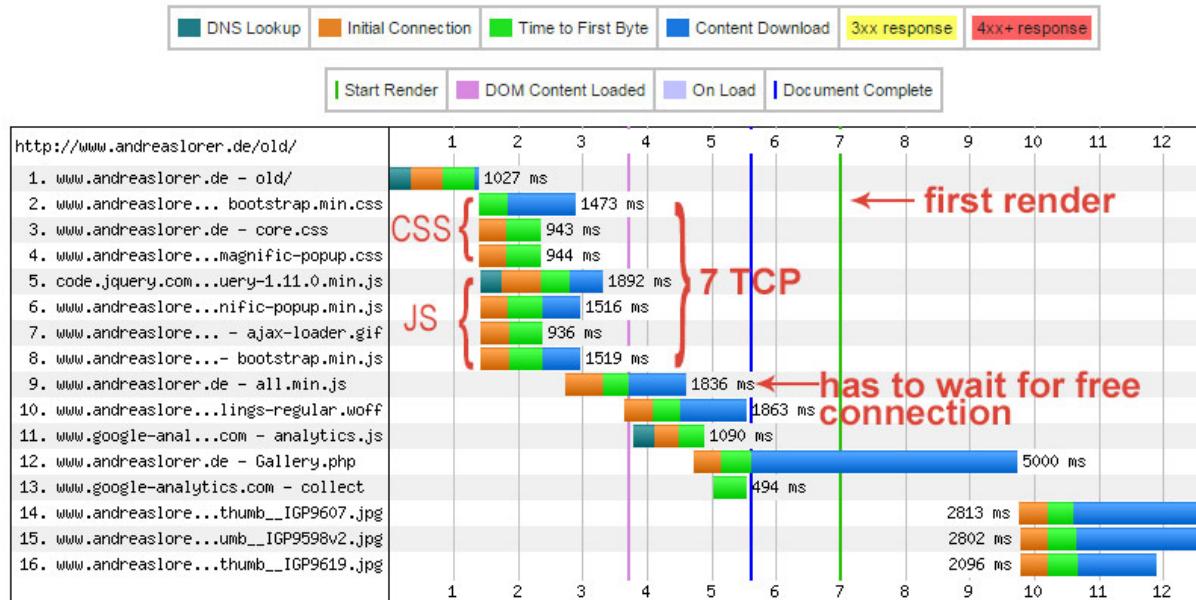


Abbildung 19: Testlauf von: „Dulles VA USA - Modell: MOTO.G. 3G shaped 1.6Mbps / 300ms RTT“ Ganzer Test: http://www.webpagetest.org/result/150308_A1_2W4/8/details/

Die Abbildung zeigt, dass typische Verhalten des Browsers: Es werden zuerst CSS, Javascript und anschließend die Bilder heruntergeladen. Hierbei fällt auf, dass er nicht mit allen Dateien gleichzeitig beginnen kann, sondern wie in Punkt 5.4: HTTP/1.1 nur 6 TCP Verbindungen pro Host Name aufbauen darf.⁹ Das bedeutet, das alle weiteren Ressourcen auf eine frei werdende TCP Verbindung warten müssen. Je weniger einzelne Dateien die Webseite benötigt, umso weniger bilden sich Warteschlangen für eine frei werdende TCP Verbindung (der Wasserfall wird flacher).

Auch in diesem Diagramm ist die Latenz als dominierender Faktor zu sehen. Es fällt auf, dass es nur einen relativ langen blauen Balken gibt (Anfrage #12 gallery.php) bei dem für längere Zeit etwas heruntergeladen wird. Das herunterladen der meisten Inhalte dauert überwiegend nur so lange, wie die Zeit die nötig war um eine Verbindung herzustellen.

Die senkrechte blaue Linie bedeutet: `Document Complete` und das heißt für den Browser, dass alle für das Rendern nötigen Ressourcen fertig heruntergeladen wurden und nun vorhanden sind. Dadurch kann er bei Sekunde 7 (senkrechte grüne Linie) mit dem Rendern beginnen. 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 minimieren. Zum Vergleich nun das Wasserfallmodell einer Optimierte Seite:

⁹Es sind deshalb 7 Verbindungen, da die Datei: „code.jquery“ von einer Google Domäne kommt und deshalb als neuer Host Name zählt.

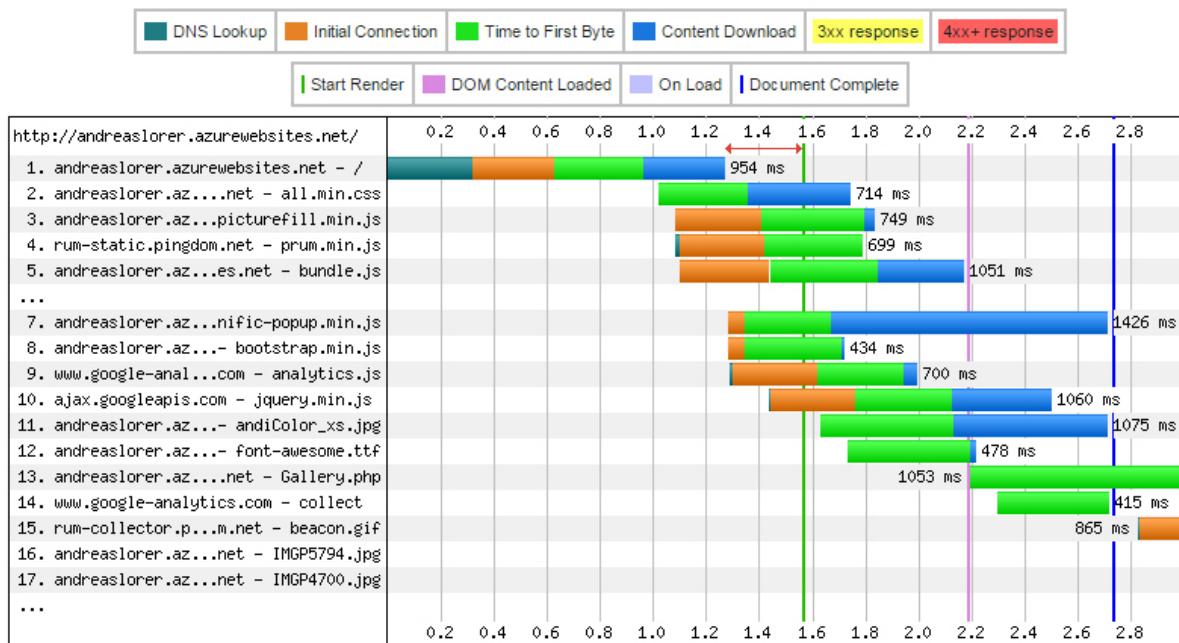


Abbildung 20: Selbe Testbedingungen wie bei Abbildung 19. Ganzer Test: http://www.webpagetest.org/result/150308_5V_JSD/6/details/

Wie zu sehen ist, fällt die senkrechte grüne Linie bereits viel früher bei rund 1.6 Sekunden. 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 zeigt genau dass, was im Kapitel „**6.2 Netzwerke**“ beschrieben wurde.

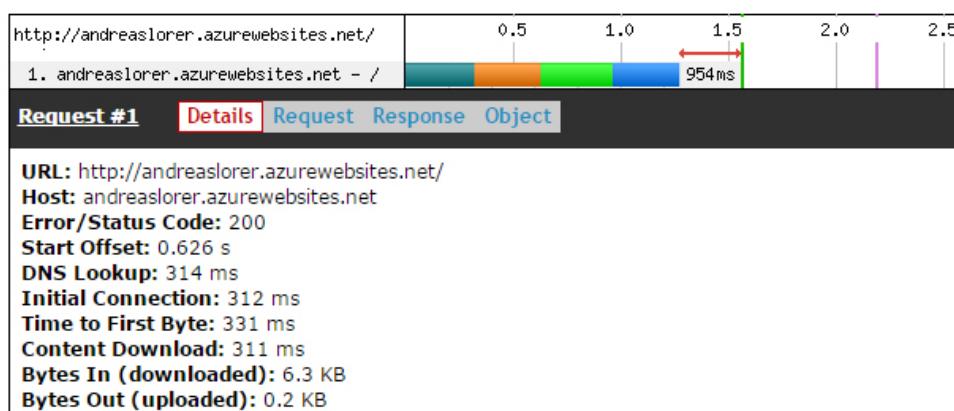


Abbildung 21: Request Nr. 1 im Detail (Abbildung nach [webpagetest.org](http://www.webpagetest.org))

Mit gerundeten Werten ergibt sich:

- DNS Lookup: 1 RTT = 300 ms

- Initial Connection (TCP - 3 Way Handshake): 1 RTT = 300 ms
- TTFB: Server Processing Time¹⁰: = 300 ms
- Content Download¹¹: 1 RTT = 300 ms
- Zeit fürs Parsen, Ausführen und Rendern: ca. 500 ms ¹²:

Obwohl hier nur eine 6.3kb Datei heruntergeladen wurde und keine 40kb so 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 300 ms zu erreichen, dies wird per Datenauswertung in Kapitel ?? ? gezeigt.

¹⁰die Antwort muss erst vom Server generiert werden

¹¹Das HTML Dokument beträgt 6.3 KB (siehe in Abbildung 21 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.

¹²Dies ist in der Abbildung mittels rotem Pfeil (<- ->) markiert

7 Entwicklung

Dieses Kapitel soll den Entwicklungsprozess konkretisieren und den Optimierungsprozess einer Webanwendung aufzeigen. Es soll erläutern, welche Fragen sich stellten und welche Antworten darauf gefunden wurden. Wie bestimmte Probleme gelöst wurden. Welche Tools und Hilfsmittel zur Verwendung kamen. Dies soll ein Bewusstsein dafür schaffen, was möglich ist und wie eine technische Umsetzung aussehen kann.

7.1 Tools

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

7.1.1 Google Chrome Developer Tool

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

- **Device Emulation**¹³: Damit lassen sich verschiedene Devices wie Smartphones, Ipad oder verschiedene Desktopauflösungen simulieren. Auch das Touchverhalten wird Simuliert.
- In der Device Emulation lässt sich auch die Netzwerkgeschwindigkeit simulieren. Dies ist allerdings nur eine Simulation und kann unter wahren Bedingungen stark abweichen.
- 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 es 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)

7.1.2 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:¹⁴. Dabei wird ein Rating von 1 (schlecht) bis 100 (gut) vergeben. Mobile und Desktop Version werden voneinander unabhängig bewertet. Findet das Tool Verstöße gegen die **best practices**, so gibt es Hilfestellungen wie zum Beispiel weiterführende Links oder Hinweise zur Behebung des Problems. Für die Verbesserung der Performance ist dieses Tool eines der besten Anlaufziele, um einen Überblick zu bekommen wo sich die Probleme befinden. Pagespeed Insight gibt es auch als Plugin für das Google Chrome Developer Tool.¹⁵

7.1.3 Google Closure Compiler

Ein simples Tool von Google¹⁶, mit der Aufgabe Javascript zu verkleinern. Dieser Vorgang nennt sich auch „minify“ und ist auch für HTML und CSS möglich. Ein Beispiel:

¹³Bei geöffnetem Tool (F12): strg + shift + M oder klick auf das Smartphone Symbol

¹⁴<http://tinyurl.com/nvxksks>

¹⁵Plugin - Pagespeed Insight: <http://tinyurl.com/mv8fcx8>

¹⁶<http://closure-compiler.appspot.com/>

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 += encodeURI(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+=encodeURI(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.

7.1.4 Webpagetest

[Webpagetest.org](http://webpagetest.org) ist das wohl umfangreichste und beste Website-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“) und es gibt zahllose Einstellungen und undokumentierte Funktionen auf die man nur in Vorträgen oder Foren stoßt. Es gibt auch ein Buch, dass sich nur mit diesem Tool beschäftigt, beim Verlag O'Reilly.¹⁷ Die Features für Webpagetest sind vielseitig:

- Es lassen sich Webanwendungen mittels eines in der realität existierenden Geräts testen. So kann vom Standort Dulles VA ein MOTOG zum Testen einer Seite verwendet werden. Dieses Gerät ruft dann auch wirklich die eingegebene URL auf und die darunterliegende Schicht misst die Zeit. Abbildung 22 zeigt den Teststandort Dulles VA.¹⁸
- Webpagetest hat die wohl genaueste Erfassung von Netzwerkzeiten und spiegelt damit realitätsgerecht die Ladezeiten einer Seite wieder.
- Webpagetest liefert eine enormes Spektrum an Daten und Diagrammen, was ausführliche Analysen zulässt.
- Speed Index: Dies ist eine von diesem Tool eigene Maßeinheit zum bestimmen der **Perceived Performance** einer Seite.

¹⁷ Buch - Using WebPagetest: <http://shop.oreilly.com/product/0636920033592.do>

¹⁸ Einen detaillierten einblick vom Gründervater und Entwickler Patrick Meenan gibt es hier: <http://tinyurl.com/o4b3rxh>

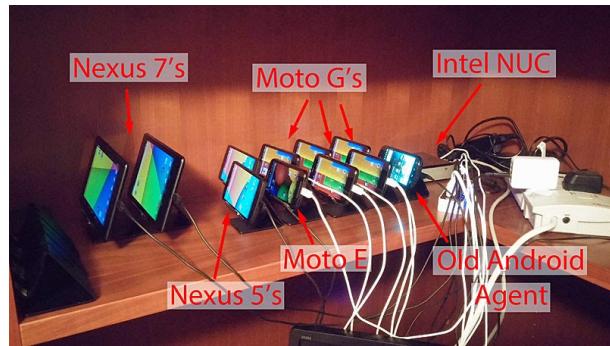


Abbildung 22: Webpagetest Android device farm (Abbildung von (Meenan 2014))

„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.“ (wevpagetest.org 2015)

- Man kann Tests direkt miteinander vergleichen. Das ist möglich, indem diese URL eingegeben wird: www.webpagetest.org/video/compare.php?tests= und nach dem „=“ Zeichen die Test ID eingibt, beispielsweise „150310_8E_GRH“. 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.
- Video erstellung: 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.
- 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 Punkt: ?? ?. 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 ist, 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 soviele Tests ausführen, wie die Leistungs des Servers bietet.

7.1.5 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 solch einer Form abrufbar.

7.1.6 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 denen der Konkurrenz vergleichen.



Abbildung 23: Speedcurve Life Monitoring (Abbildung von <http://speedcurve.com/>)

7.1.7 Google Spreadsheet

Ist im Grunde 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 kleine Programme schreiben lassen. So sind zum Beispiel API Abfragen möglich, dessen Ergebnis dann direkt in die Tabelle geschrieben werden kann.

7.1.8 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.

7.1.9 What Does My Site Cost?

„Was kostet es eigentlich meinene 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://whatdoessitecost.com/> benutzt die webpagetest Schnittstelle um eine eingegebene URL zu Analyisieren 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.“*(whatdoessitecost.com 2015)



Abbildung 24: Find out how much it costs for someone to use your site on mobile networks around the world.(whatdoessitecost.com 2015)

In Deutschland kostet also der Seitenaufruf von 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 zum Ärger auch noch bares Geld kosten kann.

7.1.10 Critical Path CSS Generator

Im Kapitel „Brechen der 1000 ms Barriere“⁶ wurde gesagt, man solle das CSS des above the fold direkt in das HTML als inline CSS schreiben. <http://jonassebastianohlsson.com/criticalpathcssgenerator/> erstellt aus einer gegebener 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 einfügen.

Dieser Generator funktioniert allerdings nur dann gut, wenn sowohl die Smartphone, 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 Elemente kennt

dieser Generator natürlich nicht und kann sie folglich auch nicht beachten. Eine alternative Methode wird in Kapitel ... ?? ? vorgestellt.

7.1.11 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.“ (httpArchive 2015)

7.1.12 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).

7.1.13 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 unentdeckt bleiben.

7.2 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 konkrete Daten zu fassen. Wie bereits unter Punkt 7.1.7 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 Programm¹⁹ geschrieben (MIT License), dass es ermöglicht Webpagetest innerhalb einer Google Tabelle²⁰ aufzurufen. Damit wurde während der Entwicklungsphase täglich tests aufgezeichnet.²¹ Die Auswertung dieser Daten erfolgt in Punkt ?? ?

Da nur in Dulles VA eine Testinstanz mit richtigen 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 eliminieren. Dadurch lässt sich exakter bestimmen, wie schnell ein Smartphone mit 3G Netz die Seite aufrufen kann. Leider steht keine Testinstanz mit 4G Netz 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: ²²

- 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: ²³

- 17 requests: 4 Images, 5 JS, 3 CSS, 4 other
- 363 kb Seitengröße
- Speed Index: **10642**
- Start Render **6968** ms
- Load Time: 5587 ms
- TTFB: 1292 ms

¹⁹ WebPageTest Bulk Tester via GitHub: <https://github.com/andydavies/WPT-Bulk-Tester>

²⁰ Das Google Dokument ist hier zu finden: <http://tinyurl.com/nv4pge5>

²¹ Die gesamten Daten sind hier zu finden: <http://tinyurl.com/l5usz79>

²² Webpagetest: http://www.webpagetest.org/result/150312_Z1_18QD/

²³ Webpagetest: http://www.webpagetest.org/result/150308_A1_2W4/

Diese Werte sind nicht gut und für dieses Projekt wurden eine Start Render Zeit von weniger als einer Sekunde und ein Speed Index von unter 1000, für sowohl 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 (oft 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 hier eine Alternative gefunden werden. „Font Awesome“²⁴ 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 die 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](#) 7.1.2 Analyisiert. Das Ergebnis liefert Anhaltspunkte, was für schnellere Ladezeit alles umgesetzt werden sollte. Im folgenden soll erläutert werden, was es alles an Verbesserungen gibt und wie eine mögliche Umsetzung in der Praxis aussieht.

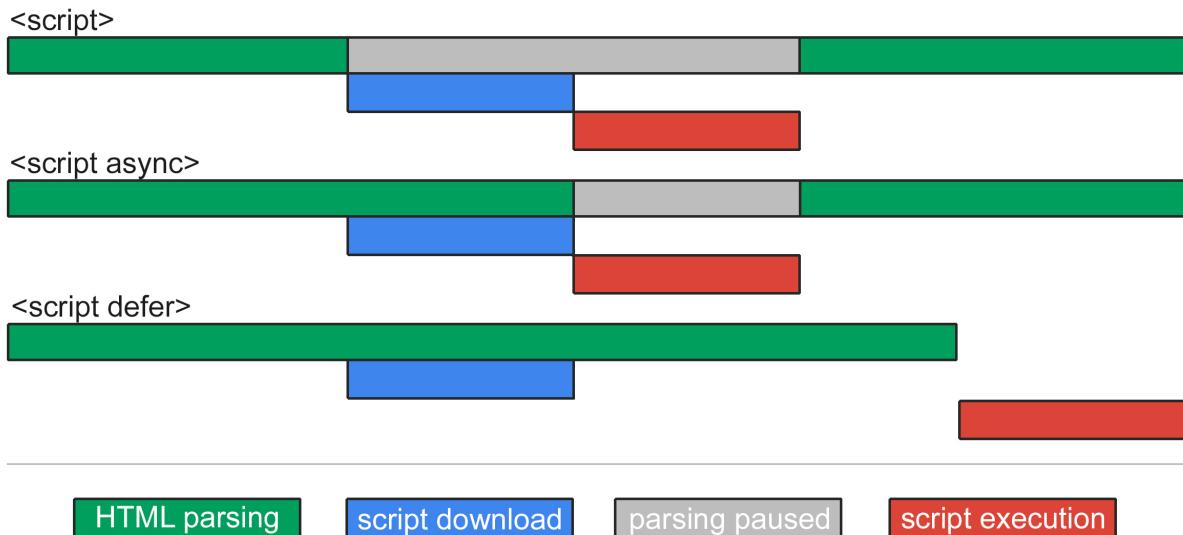
7.2.1 Render Blocking Javascript

Bereits unter Punkt 6.6.2 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 für das Rendern 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 Browsertyp 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 die Reihenfolge der Skripte im HTML Dokuments vorliegen.

Diese Methode erlaubt es, Skripte parallel herunterzuladen, ohne dass der Renderprozess warten muss. Was damit nicht erreicht werden kann ist, dass der Download so lange verzögert wird, bis die für die Seiten primär wichtigen Ressourcen zuerst heruntergeladen wurden.

²⁴Font-Awesome: <http://fontawesome.github.io/Font-Awesome/>



Mit Hilfe des Javascripts in Listing 4, kann die Datei „defer.js“ komplett mit dem Laden verzögert werden, bis der Ladeprozess der Seite abgeschlossen ist.

```

1   // the function to asynchronous load js
2   function loadJS( src, cb ){
3     "use strict";
4     var ref = window.document.getElementsByTagName( "script" )[ 0 ];
5     var script = window.document.createElement( "script" );
6     script.src = src;
7     script.async = true;
8     ref.parentNode.insertBefore( script, ref );
9     if (cb && typeof(cb) === "function") {
10       script.onload = cb;
11     }
12     return script;
13   }
14
15 // the function call to load your script
16 loadJS( "path/to/script.js" );
  
```

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, 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 das HTML Dokument zu schreiben. Eine bessere Alternative ist **jQ1**.²⁵ Die Verwendung ist sehr simpel:

```

1 <script type="text/javascript">
2     // first include jQ1 inline (missing here) then call these functions
3     jQ1.loadjQ('jquery.js');
4     jQ1.loadjQdep('defer.js');
5 </script>

```

Listing 6: jQ1 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 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 und das Skript aus Listing 6 befindet sich vor dem „</body>“-Tag.

²⁵jQ1 an asynchronous jQuery Loader: <http://www.yterium.net/jQ1-an-asynchronous-jQuery-Loader>

7.2.2 Render Blocking CSS

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

```

1   <script>
2     // minified script after:
3     // https://github.com/filamentgroup/loadCSS/blob/master/loadCSS.js
4     // [c]2014 @scottjehl, Filament Group, Inc.
5     // Licensed MIT
6     function loadCSS(e,a,g,h){function f(){for(var a,c=0;c<d.length;c++)d[c].href&&
7       -1<d[c].href.indexOf(e)&&(a!=0);a?b.media=g||"all":setTimeout(f)}
8       var b=window.document.createElement("link");a=a||
9       window.document.getElementsByTagName("script")[0];
10      var d=window.document.styleSheets;b.rel="stylesheet";b.href=e;b.media="only x";
11      b.onload=h||function(){a.parentNode.insertBefore(b,a);f()};return b
12    };
13
14    loadCSS( "path/to/css" );
15  </script>
16
17  <!-- fallback if javascript is disabled in browser -->
18  <noscript><link href="path/to/css"></noscript>
```

Listing 7: load a CSS file asynchronously

Mehr als dieses Skript ist nicht notwendig.

7.2.3 Inline von CSS

Kleinere Mengen an CSS lassen sich direkt **Inline** in das HTML Dokument einfügen. Dadurch sind diese gleich mit dem ersten Request bereits im Dokument enthalten und müssen nicht erst angefordert und heruntergeladen werden.

7.2.4 Ressourcen reduzieren

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

- „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 7.1.2 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 (7.1.3) das richtige Werkzeug. CSS lässt sich per <http://cssminifier.com/> verkleinern.
- „Uglify“: Dabei werden Variablennamen, auf nur wenige Zeichen reduziert. Aber auch 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
9      // output:
10     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, denn dafür gibt es, wie bereits erwähnt wurde (Punkt 5.4), ein Limit.

7.2.5 CSS-Bereitstellung optimieren

Wenn externe Ressourcen klein sind, können diese direkt in das HTML Dokument **Inline** platziert werden. Dabei sollte darauf geachtet werden, dass das HTML Dokument Komprimiert 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 werden und mittels Skript Verzögert geladen werden. Das CSS für den above the fold Bereich sollte **Inline** im „**<head>**“ Bereich der Seite stehen.

7.2.6 Antwortzeit des Servers reduzieren

Die Zeit zur Antwort des Servers lässt sich zum Beispiel mit Webpagtest 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: 6.5 wurde es nahegelegt ein gutes Hosting zu wählen. Besser Sie wechseln ihr Hosting als ihre Kunden den Service.

7.2.7 Browser-Caching nutzen

Fehlendes Browser-Caching (das lokale Speichern von Daten) wird von Pagspeed 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 Ressourchen außer das HTML Dokument selbst, gecached 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      ## EXPIRES CACHING ##
2      <IfModule mod_expires.c>
3      ExpiresActive On
4      ExpiresByType image/jpg "access 1 year"
5      ExpiresByType image/jpeg "access 1 year"
6      ExpiresByType image/gif "access 1 year"
7      ExpiresByType image/png "access 1 year"
8      ExpiresByType text/css "access 1 year"
9      ExpiresByType text/woff "access 1 year"

```

```

10    ExpiresByType application/pdf "access 1 year"
11    ExpiresByType text/x-javascript "access 1 year"
12    ExpiresByType application/x-shockwave-flash "access 1 year"
13    ExpiresByType image/x-icon "access 1 year"
14    ExpiresDefault "access 1 month"
15    </IfModule>
16    \#\# EXPIRES CACHING \#\#

```

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

Listing 9 hat 2 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 mit gesendet. 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 2014) Das bedeutet, dass die Latenz bei Smartphones für gecachte Dateien komplett negiert 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 sind jedes mal ein round trip nötig.



Abbildung 26: Gecachte Ressourcen müssen nicht mehr abgefragt werden

Das rot unterstrichene zeigt, dass 59 ms im Netzwerk verbraucht wurde (Kabel Verbindung). Der Server antwortete mit: „Not-Modified“. Grün zeigt, 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 Verbindungsauftakt 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 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 für alle Anwender neu Angefordert.
3. Ressourcen können mit einer ID versehen werden: `styles.css` wird in `styles.v1.0.1.css` umbenannt.
4. Alternativ zur ID ist auch in `Fingerprint` möglich. Dabei wird eine ID aus der Datei generiert. Ändert sich die Datei so ändert sich auch der Fingerprint. Dieser Fingerprint wird auch wiederrum dem Dateiname angefügt. Das kann so aussehen: `styles.82s0dfa.css`.²⁶

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.

„[...] 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 Tool²⁷ zu Verfügung, mit dessen Hilfe sich überprüfen lässt, ob die eigene Webseite „Browser-Caching“ richtig einsetzt. Abbildung Nummer 27 zeigt Links eine Seite mit aktiviertem Browser-Caching und Rechts eine Seite ohne.

²⁶Dieses Verfahren lässt sich auch automatisieren, ich verweise auf folgenden Artikel: <https://adactio.com/journal/8504>

²⁷Tool: <http://www.feedthebot.com/tools/if-modified/>

The image shows two side-by-side responses from a web server. On the left, under 'Yes.', the response includes a 'Last-Modified' header. On the right, under 'No.', it does not. Both responses are otherwise identical.

Response Headers (Yes.)	Response Headers (No.)
Server Response HTTP/1.0 200 OK	Server Response HTTP/1.0 200 OK
HTTP/1.0 200 OK	HTTP/1.0 200 OK
Date: Mon, 16 Mar 2015 13:06:40 GMT	Date: Mon, 16 Mar 2015 15:13:41 GMT
Server: Apache	Server: Apache
● Last-Modified: Fri, 13 Mar 2015 16:03:43 GMT	
● ETag: "2160276-3630-5112da50015c0"	
Accept-Ranges: bytes	
Content-Length: 13872	
● Cache-Control: max-age=2592000	● Cache-Control: max-age=0
● Expires: Wed, 15 Apr 2015 13:06:40 GMT	Expires: Mon, 16 Mar 2015 15:13:41 GMT
Connection: keep-alive, close	Connection: close
Content-Type: text/html	Content-Type: text/html

There does not appear to be a "last modified header response"

Abbildung 27: Beispiel: Eine Webseite mit und ohne „Cache Control“. (Eigene Abbildung nach feedthebot.com)

7.2.8 Komprimierung aktivieren

Nachdem durch das in Punkt 7.2.4 beschriebene Verfahren die Ressourcen soweit wie möglich verkleinert wurden, können diese vor dem Versenden komprimiert werden.²⁸ Das gängigste Komprimierungsprogramm ist GZIP und wird von allen modernen Browsern unterstützt. Je nach Dateigröße können so die zu übertragenden Datenmenge drastisch Reduziert werden. GZIP kann mittels `.htaccess` Eintrag von Listing 10 in Apache aktiviert werden.²⁹ 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 verlangsamt.(Apache.org 2015)

```

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

Listing 10: gzip

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

²⁸Eine ausführlichere Beschreibung über den GZIP Algorithmus und der textbasierter Dokumentkomprimierung ist hier zu finden: <http://www.infinitepartitions.com/art001.html>. Für das tiefere Verständnis bietet auch dieses Video von Google: <http://tinyurl.com/mfxt5zt>

²⁹Für andere Server wie zum Beispiel Nginx oder Node.js gibt es auf GitHub fertige Konfigurationen <https://github.com/h5bp/server-configs>

Gzip compression test



Abbildung 28: Testen von GZIP anhand von hs-weingarten.de (Eigene Abbildung nach: feedTheBot.com)

7.2.9 „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.³⁰

³⁰Bei shared Hostings kann es trotz dieser Einstellung oft nicht erreicht werden, keep alive zu aktivieren.

7.3 Bilder optimieren

„Ein Bild sagt mehr als Tausend Worte.“ Diesen Spruch gibt es nicht umsonst und 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 **Eyeatcher**. In Abbildung 29 ist die durchschnittliche Seitengröße der Top 1000 Seiten³¹ des Webs abgebildet. So beträgt durchschnittliche die Seitengröße (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 (Mittelwert) 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)

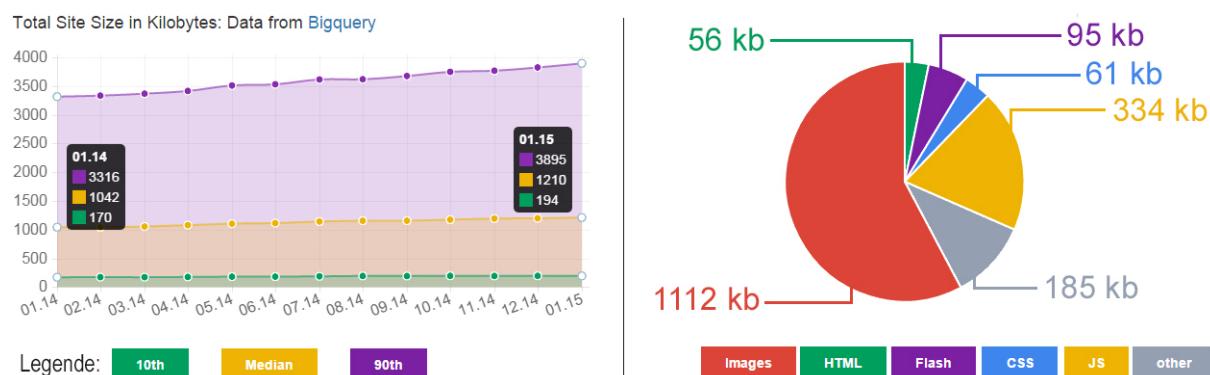


Abbildung 29: Seitenanalyse der populärsten Seiten des Webs (Eigene Abbildung - Daten via bigqueri.es: Gesamte Auswertung <http://tinyurl.com/o8vawxd>)

7.3.1 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 nun 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:

Use Progressive JPEGs: 100/100
124.8 KB of a possible 124.8 KB (100%) were from progressive JPEG images

Abbildung 30 zeigt einmal den normalen Ladevorgang eines Bildes (unten) gegenüber dem „Progressive Rendering“ (oben).

³¹Top 1000 Seiten nach dem Ranking von Alexa: (Alexa gehört zu Amazon und liefert umfassende Analysen über das Web <http://www.alexa.com/>)



Abbildung 30: Progressives Bilder 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.

7.3.2 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 31: 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 angefragt und ausgetauscht werden.

7.3.3 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.

- Lossly ist die reduzierung der Bildgröße auf kosten von Qualität. Für die meisten Bilder ist diese Reduzierung aber durchaus Legetim und oftmals sind die Einbußen mit bloßem Auge nicht zu erkennen. Lossly 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-Code³² nötig, um dies auf dem eigenen Betriebssystem zu verwenden.

Glücklicherweise es gibt eine Webanwendung³³ 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 jedenfall in betracht gezogen werden.

7.3.4 Responsive Images

Bei Bootstrap gibt es die Möglichkeit dem Bild eine Klasse zuzuweisen und das Bild wird dann abhängig von der Fenstergröße skaliert. Genau das steckt aber nicht hinter diesem Begriff. „Responsive Images“ ist eine auf das Gerät des Endanwenders angepasste Auslieferung von Bildern. Auf vielen Webanwendungen sieht man folgendes:



Abbildung 32: Downsampling auf GitHub (Eigene Abbildung via Chrome Developer Tool)

³²Das Repository ist hier zu finden: <https://github.com/mozilla/mozjpeg> und eine Anleitung gibt es hier: <http://calendar.perfplanet.com/2014/mozjpeg-3-0/>

³³Webanwendung zur Verwendung von moz jpeg: <https://imageoptim.com/mozjpeg>

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. 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 Polyfill³⁴ namens „Picturefill“³⁵, dass genau diese Funktionalität 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 12: 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: Zeile 8). Man kann sogar einen Schritt weiter gehen und auch das Bildformat auf 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)">
6     <source srcset="someImg_xs.webp" type="image/webp" media="(min-width: 300px)">
7     <source srcset="someImg_xs.jpg" media="(min-width: 300px)">
8     <img srcset="fallback_md.jpg" alt="Some alt text">
9   </picture>
```

Listing 13: Srcset mit webp

Wie zu sehen ist wurden 3 Zeilen eingefügt, die sich lediglich im Typ 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, Downsampling zu vermeiden und einen auf das Gerät angepasste Version des Bildes auszuliefern. Dadurch sinkt die Anzahl von Bytes die Übertragen werden müssen enorm.

³⁴ „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.“ ([wikipediaPolyfill](#))

³⁵ Das offizielle Picturefill Projekt ist hier zu finden: <http://scottjehl.github.io/picturefill/>

7.3.5 Adaptive Images

Adaptive Images ist ein Skript, das mit Hilfe eines PHP-Skripts und 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.³⁶

7.4 Prozess der Validierung

Bereits zu Beginn des Projekts war es wichtig es messbar zu machen. Dafür wurde zu Beginn eine Testumgebung aufgebaut, mit deren Hilfe die Seite nach ihrer Geschwindigkeit getestet werden kann.

1089 Testläufe über Webpagetest: Zeitraum: 1 Monat

Overview: First View

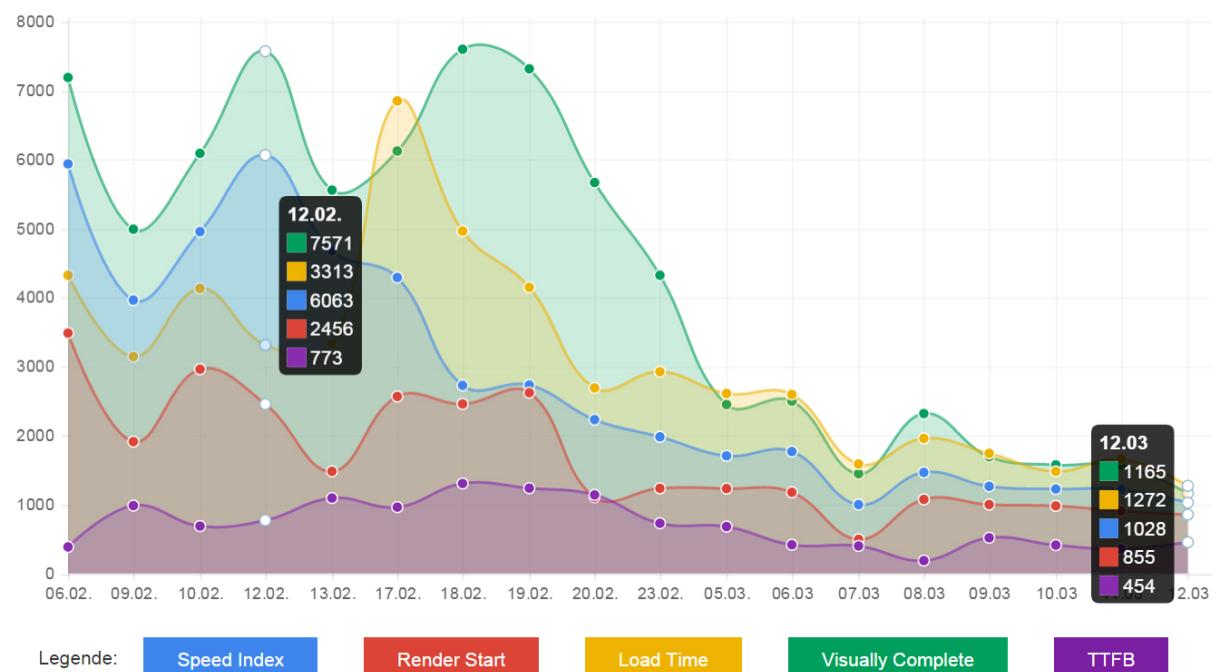


Abbildung 33: ...

todo

8 Best-Practices

8.1 Serverseitig

8.1.1 Verringern von DNS Lookups

8.1.2 HTTP Requests

pro / contra von sprites und großen concatenated files

³⁶Mehr über dieses Skript ist hier zu finden: <http://adaptive-images.com/>

8.1.3 Caching

pro / contra / problem bei updates

8.1.4 Pagespeed Mod

8.1.5 Images

8.2 Clientseitig

9 Workflow

9.1 Performanten Code schreiben

9.2 Minify und Uglify

9.3 Concatenating

9.4 Task Manager

9.5 Dependency Manager

9.6 Generators

Literatur

- [Apa15] Apache.org. *When (not) to use .htaccess files*. <http://tinyurl.com/m6v5rut> [Aufgerufen am 17.03.2015]. 2015 (siehe S. 36).
- [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. 18).
- [Bun14] Bundesnetzagentur. *Jahresbericht 2014*. <http://tinyurl.com/l8r7flv> [Aufgerufen am 19.03.2015]. Seite 78f. 2014 (siehe S. 11).
- [can15] caniuse.com. *defer attribute for external scripts*. <http://caniuse.com/> [Aufgerufen am 13.03.2015]. 2015 (siehe S. 29, 40, 41).
- [Goo10] Google. *Using site speed in web search ranking*. Website. 2010 (siehe S. 3).
- [Goo11] Google. *Creating Fast Buttons for Mobile Web Applications*. https://developers.google.com/mobile/articles/fast_buttons [Aufgerufen am 03.03.2015]. 2011 (siehe S. 10).
- [Goo14a] Google. *Browser-Caching nutzen*. <https://developers.google.com/speed/docs/insights/LeverageBrowserCaching> [Aufgerufen am 16.03.2015]. 2014 (siehe S. 34).
- [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. 33).
- [Gri13a] Ilya Grigorik. *Breaking the 1000ms Mobile Barriere*. https://docs.google.com/presentation/d/1wAxB5DPN-rcelwbG06lCOus_S1rP24LMqA8m1eXEDRo/present?slide=id.g11c1373c5_3_0 [Aufgerufen am 04.03.2015]. Slides. 2013 (siehe S. 12, 13).
- [Gri13b] Ilya Grigorik. *Breaking the 1000ms Mobile Barriere*. https://docs.google.com/presentation/d/1wAxB5DPN-rcelwbG06lCOus_S1rP24LMqA8m1eXEDRo/present?slide=id.g11c1373c5_5_35 [Aufgerufen am 04.03.2015]. Slides. 2013 (siehe S. 13).
- [Gri13c] Ilya Grigorik. *High Performance Browser Networking*. <http://tinyurl.com/p5dds9p> [Aufgerufen am 02.03.2015]. Chapter 2 Slow-Start. 2013 (siehe S. 6, 7).
- [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. 10).
- [Gri13e] Ilya Grigorik. *High Performance Browser Networking*. <http://tinyurl.com/nojaxxa> [Aufgerufen am 02.03.2015]. Chapter 7 Table 7.1. 2013 (siehe S. 12).
- [Gri14] 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. 34).
- [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. 30).
- [Gro15] Filament Group. *Defer Loading Javascript*. <https://github.com/filamentgroup/loadJS> [Aufgerufen am 13.03.2015]. GitHub. 2015 (siehe S. 30).
- [Hol10] Urs Holzle. *Velocity 2010: Urs Holzle*. <http://tinyurl.com/px7m64m> [Aufgerufen am 27.02.2015]. Video from Velocity Conference. 2010 (siehe S. 3).

- [htt15] httpArchive. *Interesting stats*. <http://httparchive.org/interesting.php> [Aufgerufen am 04.03.2015]. 2015 (siehe S. 27).
- [ItW15] ItWissen.info. *Shared Hosting*. <http://www.itwissen.info/definition/lexikon/Shared-Hosting-shared-hosting.html> [Aufgerufen am 26.02.2015]. 2015 (siehe S. 5).
- [Les14] Kornel Lesiński. *MozJPEG 3.0*. <http://calendar.perfplanet.com/2014/mozjpeg-3-0/> [Aufgerufen am 12.03.2015]. 2014 (siehe S. 40).
- [Mee14] Patrick Meenan. *Titel*. <http://tinyurl.com/o4b3rxh> [Aufgerufen am 11.03.2015]. blog. 2014 (siehe S. 24).
- [Rad13] Radware. *Radware Mobile Infographic*. Website. 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. 3).
- [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. 9, 38).
- [Rit14] Michael Ritz. *Weltkarte in Schwarz*. <http://www.landkartenindex.de/kostenlos/?p=31> [Aufgerufen am 25.02.2015]. 2014 (siehe S. 7).
- [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. 8).
- [Sex15a] Patrick Sexton. *Enable Keep Alive*. <http://www.feedthebot.com/pagespeed/keepalive.html> [Aufgerufen am 17.03.2015]. 2015 (siehe S. 37).
- [Sex15b] Patrick Sexton. *Leverage Browser Caching*. <http://www.feedthebot.com/pagespeed/leverage-browser-caching.html> [Aufgerufen am 16.03.2015]. 2015 (siehe S. 34).
- [Ste06] Guido Stepken. *Anti-Pattern in der Softwareentwicklung*. <http://www.little-idiot.de/teambuilding/AntiPatternSoftwareentwicklung.pdf> [Aufgerufen am 26.02.2015]. 2006 (siehe S. 5).
- [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. 9).
- [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 (siehe S. 6).
- [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. 3).
- [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. 10, 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. 3, 4).

- [Ton13] Rmistry und Tonyg. *Loading measurement: alexa top million netsim*. <https://docs.google.com/document/d/1cpLSSYpqi4SprkJcVxbS7af6avKM0qc-imxvkexmCZs/edit> [Aufgerufen am 04.03.2015]. 2013 (siehe S. 11).
- [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. 3).
- [weg15] wegpagtest.org. *Speed Index*. <https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index> [Aufgerufen am 11.03.2015]. 2015 (siehe S. 24).
- [wha15] whatdoesmysitecost.com. *What Does My Site Cost?* <http://whatdoesmysitecost.com/site/www.hs-weingarten.de> [Aufgerufen am 12.03.2015]. 2015 (siehe S. 26).
- [wik14a] wikipedia. *HTTP-Statuscode*. <http://de.wikipedia.org/wiki/HTTP-Statuscode> [Aufgerufen am 04.03.2015]. 2014 (siehe S. 15).
- [wik14b] wiki.ubuntuusers. *Der Benutzer root*. <http://wiki.ubuntuusers.de/sudo> [Aufgerufen am 26.02.2015]. 2014 (siehe S. 5).
- [Wik15] Wikipedia. *Entwurfsmuster*. <http://de.wikipedia.org/wiki/Entwurfsmuster> [Aufgerufen am 26.02.2015]. 2015 (siehe S. 5).
- [wik15] wikipedia. *Content Delivery Network*. http://de.wikipedia.org/wiki/Content_Delivery_Network [Aufgerufen am 04.03.2015]. 2015 (siehe S. 7).
- [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. 35).