

Bachelorarbeit

Performanceoptimierung von Webanwendungen

**Zur Erlangung des akademischen Grades
Bachelor of Science**

Fachhochschule Stuttgart – Hochschule der Medien



://WEITCLICK

Autor: **Willi Kampe**

Erstprüfer: Prof. Dr. Toenniessen

Zweitprüfer: Christian Knott

Bearbeitungszeitraum: 01.12.2012 bis 29.02.2012

Kurzfassung

In dieser Arbeit geht es um Geschwindigkeit genauer gesagt um die Performance von Webseiten und warum diese in der heutigen Zeit so wichtig ist.

Dafür geht der erste Teil darauf ein was eine gute Performance ausmacht und welche Vorteile sich daraus ergeben.

Anschließend werden theoretische Grundlagen geschaffen, um die darauffolgenden Technologien und Best Practises zum Verbessern der Performance verstehen zu können.

Der dritte große Teil dieser Arbeit befasst sich dann konkret mit dem Web-Projekt www.fein.de. Dieses wird auf seine Performanceschwächen hin analysiert und optimiert. Ein Vorher-Nachher-Vergleich zeigt dann im Anschluss was die Änderungen konkret gebracht haben.

Inhaltsverzeichnis

Kurzfassung.....	2
Inhaltsverzeichnis	3
Abbildungsverzeichnis	6
Quellcodeverzeichnis	8
Tabellenverzeichnis	9
Abkürzungsverzeichnis	10
Vorwort.....	11
Motivation	12
Ziele.....	13
1 Performance von Webseiten	14
1.1 Wie schnell ist gut?.....	14
1.2 Usability / User Experience	16
1.3 Weniger Traffic - Bessere Performance – Weniger Kosten	17
1.3.1 Optimierungsbeispiel: ImmobilienScout24	17
1.1 Suchmaschinen-Ranking	19
1.2 Performance Golden Rule	19
1.3 Performance als Business-Case	20
2 Ladevorgang einer Webseite	22
2.1 Wo bleibt die Zeit?	23
2.1.1 Processing Time.....	24
2.1.2 Bandbreite und Round-Trip-Time	24
2.1.3 Client-Side	27
3 Tweaks zur Steigerung der Performance	28
3.1 Content.....	28
3.1.1 HTTP-Requests minimieren	29
3.1.2 Minimieren und Zusammenfassen von JavaScript und CSS	29
3.1.3 DNS-Lookups minimieren und HTTP-Requests parallelisieren	31
3.1.4 Seitenaufbau planen.....	32
3.1.5 Redirects verhindern.....	33
3.1.6 Slash in Links	33
3.1.7 DOM-Elemente: Weniger ist mehr schneller!.....	34

3.2 Bilder	34
3.2.1 Bild-Formate.....	35
3.2.2 Automatisierte verlustfreie Kompression	35
3.2.3 Größenangaben im img-Tag	36
3.2.4 Inline Bilder	36
3.3 JavaScript.....	37
3.3.1 Skripte extern.....	37
3.3.2 Skripte ans Ende	38
3.4 CSS	38
3.4.1 CSS-Sprites.....	41
3.5 Netzwerk/Übertragungsweg	42
3.5.1 Komprimierung der Datenübertragung.....	42
3.5.2 HTTP keep-alive.....	43
3.5.3 Expires Header und der ETag.....	43
3.5.4 Content Delivery Network.....	46
3.5.5 Cookies	47
4 Optimieren von Fein.de.....	48
4.1 Auswertung.....	49
4.1.1 Klickpfade.....	50
4.1.2 Geschwindigkeit messen	50
4.1.3 Zieldefinition	54
4.1.4 Viele HTTP-Requests.....	54
4.1.5 HTTP Keep-Alive nicht verwendet.....	56
4.1.6 Unnötiger Redirect.....	57
4.1.7 Großteil des Traffics von einer Domain	57
4.1.8 Unsortiertes HTML-Dokument	60
4.1.9 Ineffiziente Bildformate.....	61
4.2 Umsetzung	63
4.2.1 Aktivieren von HTTP Keep-Alive.....	63
4.2.2 Redirect-Fehler beheben	65
4.2.3 Bilder optimieren	66
4.2.4 Content-Verteilung auf eine weitere Domain.....	68
4.2.5 Minimierung der HTTP-Requests	69
4.3 Ergebnis	71
5 Ausblick	73
5.1 HTTP-Pipelining	74
5.2 HTML5 Application Cache	75
5.3 WebP	77
5.4 SPDY	77

6 Fazit.....	79
Anhang A: Verwendete Tools.....	80
Page Speed.....	80
YSlow	81
WebPagetest	83
PCAP Web Performance Analyzer.....	84
Firebug für den Firefox.....	85
SpeedTracer	86
Anhang B: Inhalt der CD.....	87
Anhang C: Bildanteil der Top 10 Webseiten	88
Glossar	89
Literaturverzeichnis	90
Erklärung	92

Abbildungsverzeichnis

Abbildung 1: Startseite von ImmobilienScout24 nach dem Relaunch	17
Abbildung 2: Technischer Ablauf eines Webseitenaufrufs	22
Abbildung 3: Aufgabenverteilung beim Seitenladevorgang	23
Abbildung 4: Auswirkung der Round-Trip-Time auf die Seitenladezeit (Belshe, 2010)	25
Abbildung 5: Auswirkungen der Bandbreite auf die Seitenladezeit (Belshe, 2010).....	26
Abbildung 6: JavaScript blockiert Seitenladevorgang.....	38
Abbildung 7: CSS-Sprite der Google Webseite	41
Abbildung 8: Cacheabfrage mit dem ETag	44
Abbildung 9: Cacheabfrage mit dem Expires Header	45
Abbildung 10: Startseite von Fein.de	48
Abbildung 12: econda-Ausschnitt der User-Klickpfade von Fein.de.....	50
Abbildung 13: Zugriffe pro Land – Januar 2012 – Fein.de	51
Abbildung 14: Besucher pro Browser - Januar 2012 - Fein.de	52
Abbildung 15: Performancetest von Fein.de (Testumgebung) - Urzustand	53
Abbildung 16: Wasserfall-Modell von Fein.de vor der Optimierungsmaßnahme	55
Abbildung 17: Wasserfall-Modell-Ausschnitt mit inaktivem HTTP Keep-Alive.....	56
Abbildung 18: Anfrage- und Antwort-Header eines HTTP-Requests (Keep-Alive Off)	56
Abbildung 19: Wasserfall-Modell - Redirect Fehler	57
Abbildung 20: Datenmenge pro Domain im Verhältnis zum Gesamtvolumen (vor der Optimierung)	58
Abbildung 21: Wasserfallmodell von Fein.de (Internet Explorer 7)	59
Abbildung 22: HTTP-Anfrage-Header von Fein.de mit gesetzten Cookies.....	60
Abbildung 23: Zusammensetzung des Inhaltes der Fein-Webseite	61
Abbildung 24: Ausschnitt aus dem CSS-Sprite flag.png.....	62
Abbildung 25: Wasserfall-Modell-Ausschnitt mit aktivem HTTP Keep-Alive	64
Abbildung 26: Performanceanalyse nach dem Einschalten von HTTP Keep-Alive.....	65
Abbildung 27: Performanceanalyse nach Behebung des Redirect-Fehlers	66
Abbildung 28: Performanceanalyse nach dem Optimieren der Bilder	67
Abbildung 29: Datenmenge pro Domain im Verhältnis zum Gesamtvolumen (nach der Optimierung)	68
Abbildung 30: Performanceanalyse nach der Content-Verteilung auf eine weitere Domain	69
Abbildung 31: Wasserfall-Modell nach Content-Verteilung auf eine weitere Domain	69
Abbildung 32: Performanceanalyse nach Content-Verteilung auf eine weitere Domain	70
Abbildung 33: Vergleich der Ladezeiten vor und nach den Optimierungsmaßnahmen	71

Abbildung 34: Aktivieren von HTTP-Pipelining im Firefox	74
Abbildung 35: WebP vs. JPEG.....	77
Abbildung 36: Page Speed Online Übersicht	80
Abbildung 37: YSlow Übersicht	81
Abbildung 38: Smush.it Übersicht	82
Abbildung 39: WebPagetest Übersicht.....	83
Abbildung 40: PCAP Web Performance Analyzer Übersicht	84
Abbildung 41: Firebug Übersicht	85
Abbildung 42: SpeedTracer Übersicht	86

Quellcodeverzeichnis

Quellcode 1: nicht minifiziertes JavaScript.....	30
Quellcode 2: minifiziertes JavaScript.....	30
Quellcode 3: HTML img-Tag.....	36
Quellcode 4: Data-URI Format innerhalb eines HTML <i>img</i> -Tags.....	36
Quellcode 5: Beispiel CSS-Selector	39
Quellcode 6: Beispiel effizienterer CSS-Selektor	40
Quellcode 7: HTML5 Application Cache - <i>manifest</i> -Attribut.....	75
Quellcode 8: HTML5 Application Cache - Manifest-Datei	76

Tabellenverzeichnis

Tabelle 1: Technische vorher-nachher-Betrachtung des optimierten Bildes.....	18
Tabelle 2: Verhältnis First-Byte-Time zu Gesamtladezeit aktueller Webseiten	20
Tabelle 3: Parameter für den Performancetest mit <i>webpagetest.org</i>	20
Tabelle 4: aktuelle Browser und deren maximalen Verbindungen pro Server.....	32
Tabelle 5: Parameter für den Performancetest mit <i>webpagetest.org</i>	53
Tabelle 6: Optimierungspotenzial der Bilder im PNG-Format	62
Tabelle 7: Aufteilung der HTTP-Requests vor und nach der Optimierung.....	70
Tabelle 8: Top 10 Webseiten und deren Bildanteil	88

Abkürzungsverzeichnis

CSS	Cascading Style Sheet
DNS	Domain Name Service
GIF	Graphics Interchange Format
HdM	Hochschule der Medien
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
JPEG	Joint Photographic Experts Group
JS	JavaScript
PI	Page Impression
PM	Projektmanager
PNG	Portable Network Graphics
TCP	Transmission Control Protocol
UX	User Experience

Vorwort

Diese Arbeit ist im Rahmen des Medieninformatik-Studiums und der Bachelor Thesis von Willi Kampe an der Hochschule der Medien in Stuttgart entstanden. Auftraggeber war die Digitalagentur Weitclick aus Stuttgart.

Viele Begriffe, die den technischen Bereich des Internets betreffen sind geprägt durch die englische Sprache. Da es häufig keine adäquate Übersetzung für diese im Deutschen gibt, wurden stets die englischen Bezeichnungen verwendet und wo es nötig war in den Fußnoten erläutert.

Ich möchte mich bedanken ..

.. und zwar bei Prof. Dr. Toenniessen, für die Betreuung dieses Projektes seitens der Hochschule der Medien.

.. bei Weitclick, für die Möglichkeit bei euch diese Zeilen verfassen zu dürfen, besonders bei Christian Knott und Markus Antoni für ihre immer offenen Ohren.

.. bei ImmobilienScout24, vielen Dank Thomas Lehmann und André von Deetzen, ich habe viel von meinem Besuch bei euch mitgenommen.

.. und bei Konex Media, für tolle Gespräche die mich immer wieder neu inspiriert haben, danke André König und Robert Böing.

Motivation

In seiner bisherigen Ausbildung hat der Autor vieles über Programmiersprachen, Datenbanken und Internet-Technologien gelernt, das Thema Performance hat dabei aber immer eine eher untergeordnete Rolle gespielt.

Grund genug, sich im Rahmen einer Bachelorarbeit ausgiebig mit den neuesten Technologien und Denkansätzen zur Beschleunigung von Webseiten vertraut zu machen und an einem konkreten Projekt auszuprobieren.

Ziele

Ziel ist es, den Leser für den Performancegedanken zu sensibilisieren, es soll klar werden, wie wichtig eine gute Performance heutzutage auch im Internet ist. Dabei sollen sich auch Leser ohne technischen Hintergrund angesprochen fühlen. Es wird ein Grundverständnis von webbasierten Client/Server Systemen vorausgesetzt.

Außerdem soll diese Arbeit Interessierten einen guten Überblick über den aktuellen Stand der Entwicklungen bieten und darüber hinaus einen Leitfaden an die Hand geben, um seine eigenen Projekte auf Performanceschwachpunkte hin zu untersuchen.

1 Performance von Webseiten

Dieser Abschnitt beschäftigt sich damit, was Performance überhaupt für eine Webseite bedeutet, wie gute Performance definiert ist und warum der Betreiber einer Webseite dieses Thema nicht unterschätzen sollte.

Im Zusammenhang mit Webseiten teilt sich Performance¹ in zwei Kategorien auf. Erstens die Effizienz, mit der zum Ausdruck gebracht wird, wie gut eine Seite ihren Zweck erfüllt, z.B. „Erwirtschaftet mein Online-Shop genug Umsatz?“ oder „Finden meine Besucher das was sie auf meiner Seite suchen?“.

Und zweitens die Page-Load-Time, nachfolgend Seitenladezeit genannt. Damit ist die Zeit gemeint, die eine Webseite nach der Eingabe der URL braucht, um auf dem Bildschirm des Besuchers vollständig und funktionstüchtig zu erscheinen. Diese Arbeit beschäftigt sich mit dem zeitlichen Aspekt der Performance, also letzterem Punkt.

Heutige Webseiten sind in vielerlei Hinsicht keine klassischen Homepages mehr, die Inhalte sind längst dynamisch, usergeneriert und werden immer komplexer. Hohe Datentransferraten, effizientere Browser und durchdachte Frameworks sind der Grund für diese Entwicklung. Es ist damit möglich mehr Logik auf den Client auszulagern und dort berechnen zu lassen. Der Begriff Homepage wird deshalb in dieser Arbeit nicht verwendet und stattdessen die Begriffe Webanwendung oder Webseite verwendet.

1.1 Wie schnell ist gut?

Wie schnell sollte eine Webseite im Browser geladen sein, damit aus heutiger Sicht die Geschwindigkeit keinen negativen Einfluss auf die Usability und damit auf die User Experience² hat?

Robert B. Miller hat bereits 1968, lange Zeit vor dem Internet-Boom, eine Antwort darauf gegeben. Seine Maßstäbe gelten noch heute als anerkannte Richtlinien für Mensch-Computer-Interaktionen. Seiner Auffassung nach gibt es 3 wichtige Abstufungen (Miller, 1968):

¹ Performance dt. Leistung

² Siehe hierzu den Abschnitt 3.2 „Usability / User Experience“

³ Siehe hierzu den Abschnitt 3.2 „Usability / User Experience“

⁴ Ein Relaunch beschreibt das Ablösen eines Systems, was bisher verwendet wurde, durch ein Neues.

- Jede Aktion die schneller ist als **100ms** bedarf keines Feedbacks an den User, der Benutzer hat den Eindruck seine Interaktion mit der Software passiert in Echtzeit.
- Wartezeiten bis zu **1s** sind für den Benutzer erträglich und benötigen kein gesondertes Feedback. Allerdings geht das Gefühl für den Benutzer verloren, dass seine Eingaben unmittelbar umgesetzt werden.
- Dauert die Verarbeitung bis zu **10s**, möchte der Benutzer zumindest darüber informiert sein, dass die Verarbeitung noch andauert. Dies wird beispielsweise über eine Sanduhr am Maus-Cursor oder ein sich wiederholendes Element auf einer Webseite umgesetzt.

Alle Verarbeitungen die noch länger andauern, sollten laut Müller ein Feedback mit Zeitangabe besitzen und möglichst den Fortschritt visualisieren. Dazu eignen sich hervorragend Fortschrittsbalken.

Auch wenn sich diese Arbeit mit der Geschwindigkeit von Webanwendungen beschäftigt und die Benutzer im Web, historisch gesehen, das Warten gewohnt sind, bleibt das Anwendungsgebiet immer noch die Interaktion zwischen Mensch und Computer. Wie vor 43 Jahren gilt, je langsamer auf eine Benutzeraktion reagiert wird, desto schneller ist der Benutzer von seinem eigentlichen Vorhaben abgelenkt. Interessant in diesem Zusammenhang sind zwei Experimente.

Im Jahre 2009 machte Google ein Experiment indem sie dem Suchenden die Ergebnisse erst mit einer Verzögerung von 400ms präsentierten. Das Ergebnis war ein Rückgang der täglichen Suchanfragen pro Benutzer um 0.6%. Darüber hinaus ist sogar ein Nachhaltigkeitseffekt aufgetreten, die Benutzer haben sogar nach dem Entfernen der Performancebremse aus Gewohnheit noch weniger mit dem System interagiert und haben nur langsam wieder zurück zu ihrem alten Verhalten gefunden (Brutlag, 2009).

Das klingt im ersten Moment nicht viel, ist aber, wenn man sich klar macht wie viele Suchanfragen Google täglich beantwortet, ein enormer Verlust an Benutzeraktivität und damit auch ein wertvoller Verlust von Werbeeinnahmen.

Ein weiteres Beispiel für die Wichtigkeit von guter Performance ist die 2008 umgestaltete Einkaufsplattform Shopzilla, die Funktionäre dieses E-Commerce-Unternehmens haben genau gemessen, was ihnen dieser Relaunch gebracht hat. Die neue Seite konnte dem Besucher fünf Sekunden schneller präsentiert werden, von ursprünglich 6-9s auf 1,2s. Während die Einnahmen durch Verkäufe über den Shop dadurch um 12% stiegen, wurden 50% der Hardwarekosten durch Optimierungen der neuen Seite eingespart. Diese benötigte im Vergleich zur alten Version nur noch 10% der Hardware-Ressourcen (Dixon, 2009).

Diese Beispiele zeigen wie wichtig eine gute Performance von Webseiten sowohl für den Betreiber, als auch für den Benutzer und damit sein Nutzererlebnis³ ist.

“By making your apps slow you actual make your users feel older”

Kelly Norton (Google Speed Tracer Team)

1.2 Usability / User Experience

Im Zusammenhang von Webseiten und der dahinterliegenden Psychologie fallen häufig die Begriffe Usability und User Experience. Da deren Bedeutung aber oft vermischt oder vertauscht wird, werden diese hier erläutert. Beide Begriffe sind mittlerweile in ISO-Standards definiert und klar voneinander abgegrenzt.

Usability

“... das Ausmaß, in dem ein Produkt durch bestimmte Nutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen.“

Auszug aus DIN EN ISO 9241-11

Mit dem Begriff Usability oder auch Gebrauchstauglichkeit wird also beschrieben wie gut eine Sache, ein Produkt oder wie im Falle dieser Arbeit eine Webseite seine Aufgabe erledigt. Dabei steht aber nicht das Produkt selbst, sondern immer derjenige der es benutzen soll, im Mittelpunkt. Es spielt eine große Rolle für wen das Produkt effektiv und zufriedenstellend funktionieren soll, ein Kinderspielzeug braucht keinen Studenten zu erfreuen. Ein Bankautomat sollte allerdings für alle zu bedienen sein, die Geld von einem Konto abheben dürfen.

User Experience

“Wahrnehmungen und Reaktionen einer Person, die aus der tatsächlichen und/oder der erwarteten Benutzung eines Produkts, eines Systems oder einer Dienstleistung resultieren“

Auszug aus DIN EN ISO 9241-210

³ Siehe hierzu den Abschnitt 3.2 „Usability / User Experience“

Das Konzept der User Experience erweitert nun den Usability-Begriff um die Betrachtung des Benutzers vor und nach dem Verwenden eines Produktes. So macht sich der Käufer eines Produktes im Vorfeld darüber Gedanken wie es zu benutzen ist und was es ihm bringen wird. Ob diese Erwartungshaltung nun erfüllt wird oder nicht, schlägt sich nach dem tatsächlichen Benutzen in eine positive, nachhaltige emotionale Bindung oder in eine Distanzbildung nieder.

Eine gute Usability ist also ein Baustein für eine positive, intensive und nachhaltige User Experience.

1.3 Weniger Traffic - Bessere Performance – Weniger Kosten

Wird die Größe einer Seite im Zuge einer Optimierung verringert, wirkt sich das positiv auf die Performance aus, da weniger Daten übertragen werden müssen. Außerdem hat es direkte Auswirkungen auf die laufenden Kosten einer Webseite, da die meisten Tarifmodelle der Hosting-Agenturen eine Bezahlung nach produziertem Traffic (pro Gigabyte) vorsehen.

Das bedeutet, je höher eine Webseite frequentiert ist, desto relevanter wirken sich auch kleine Optimierungen aus. Das folgende Beispiel verdeutlicht dieses Verhältnis.

1.3.1 Optimierungsbeispiel: ImmobilienScout24

Ende des Jahres 2011 hat ImmobilienScout24.de sein Webportal für das Anbieten und Suchen von Immobilien einem Relaunch⁴ unterzogen.

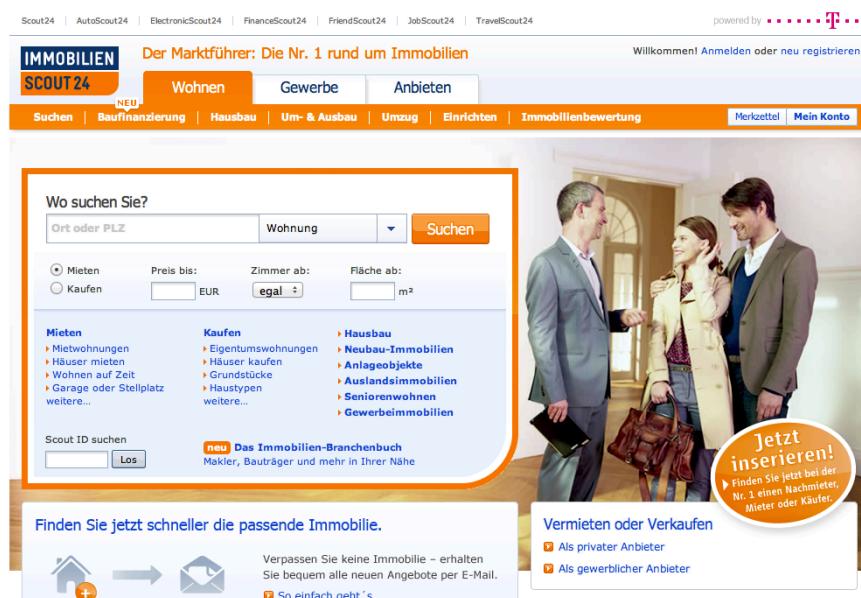
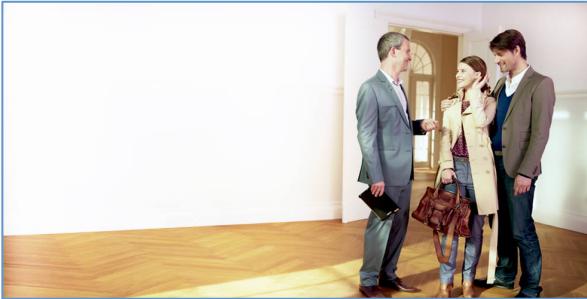


Abbildung 1: Startseite von ImmobilienScout24 nach dem Relaunch

⁴ Ein Relaunch beschreibt das Ablösen eines Systems, was bisher verwendet wurde, durch ein Neues.

Dabei wurde auf der Startseite ein Eye Catcher⁵ in Form eines JPEG–Bildes eingesetzt, dieses zeigt eine typische Situation zwischen Immobilienmakler und seinen Kunden. Zunächst wurde dieses Bild in einer für das Web unnötig hohen Qualität in die Seite eingebunden. Nachdem dieses dann optimiert wurde, konnte erheblich Traffic eingespart werden.

Tabelle 1: Technische vorher-nachher-Betrachtung des optimierten Bildes



	Vorher	Nachher
Größe	150 KB	47.8 KB
Maße	984px × 495px	984px × 495px
Format	JPEG	JPEG

Die Optimierung dieses Bildes hat also eine Komprimierung von 68% bewirkt und sorgt dafür, dass 102,2KB Daten bei jedem neuen Besucher der Startseite nicht mehr übertragen werden müssen. Die ImmobilienScout24-Seite wird von durchschnittlich 6 Millionen eindeutigen Besuchern im Monat aufgerufen (ImmobilienScout24, 2012). Das bedeutet, es müssen durch diese Maßnahme 584,8GB Daten pro Monat nicht mehr übertragen werden. Und da ImmobilienScout24 bei seinem Content Delivery Network für jedes übertragene Gigabyte zahlt, ist mit dieser simplen Optimierung nicht nur die Geschwindigkeit der Seite optimiert, sondern auch finanziell eingespart worden.

⁵ Als Eye Catcher bezeichnen die Grafiker Bilder, die eine positive Emotion in Verbindung mit der Marke auslösen sollen.

1.1 Suchmaschinen-Ranking

Nach eigener Aussage sind Suchmaschinenanbieter immer darauf bedacht, dem Benutzer auf seine Anfrage hin, möglichst wertvolle und relevante Inhalte zu präsentieren. Da die Performance ein wichtiger Bestandteil für gute Usability ist, zählt Google diese seit April 2010 mit in ihren Ranking-Algorithmus. Das bedeutet, je schneller eine Webseite lädt, desto besser wird auch ihr Suchmaschinenrang (Googleblog, 2012).

Da sich die großen Suchmaschinen aber nicht direkt in die Karten schauen lassen und verraten mit wie viel Gewicht dieser Faktor zählt, lässt sich darüber nur spekulieren. Im Zuge einer SEO-Maßnahme⁶ sollte die Geschwindigkeit allerdings immer Beachtung finden.

1.2 Performance Golden Rule

Steve Souders, eine Koryphäe auf dem Gebiet der Web-Performanceoptimierung beschreibt in seinem Buch „High Performance Websites“ wie er einen Großteil seiner Arbeit im Web damit verbracht hat, die Back-End-Performance von Webseiten zu optimieren. Dabei hat sein Focus auf Dingen wie Compiler-Optionen, Datenbankindizes oder Speicherverwaltung gelegen. Dies ist seiner Meinung nach auch ein wichtiger Aspekt der ganzheitlichen Optimierung einer Webseite. Allerdings ist ihm auch ein weitverbreiteter Irrglauben vieler Entwickler aufgefallen, es ist nämlich falsch zu denken, dass die Backendperformance maßgeblich für die Gesamtladezeit ist. Vielmehr sollte das Augenmerk auf der Seite des Clients und dem Weg dorthin liegen.

Seine „Performance Golden Rule“ lautet deshalb sinngemäß:

„Nur 10-20% der Gesamtladezeit einer Webseite werden gebraucht um das HTML-Dokument herunterzuladen. Die anderen 80-90% werden für das Herunterladen und Interpretieren aller Komponenten der Seite gebraucht.“

Der Focus einer Optimierung sollte also zunächst beim Benutzer auf der Client-Seite liegen. Also dort wo 80-90% der Zeit gebraucht werden (Steve Souders, 2007, S. 4).

Ein Test von fünf bekannten Webseiten bestätigt diese These. Es wurde die First-Byte-Time gemessen und ins Verhältnis zur Gesamtladezeit gesetzt. Die First-Byte-Time beschreibt die Zeit vom ersten Request einer Seite bis zur Ankunft des ersten Bytes des HTML-Dokumentes. Diese Zeit gibt Aufschluss darüber, wie lange der Server braucht um das HTML-Dokument auszuliefern.

⁶ SEO (Search Engine Optimizing) zu dt. Suchmaschinen Optimierung

Tabelle 2: Verhältnis First-Byte-Time zu Gesamtladezeit aktueller Webseiten

Webseite	First-Byte-Time	Gesamtladezeit	Verhältnis
http://www.berlin.de/	0.294s	5.227s	5,62%
http://www.fein.de/	1.286s	37.144s	3,46%
http://www.immobilienscout24.de/	0.764s	6.380s	11,97%
http://www.zeit.de/	0.504s	6.591s	7,65%
http://www.bild.de/	0.362s	20.936s	1,73%

Dieser Test wurde mit dem Analyse-Tool webpagetest.org⁷ und folgenden Parametern durchgeführt.

Tabelle 3: Parameter für den Performancetest mit webpagetest.org

Parameter	Wert
Bandbreite	DSL (1,5Mbps/384Kbps)
Round Trip Time (RTT)	50ms
Standorte	Frankfurt, DE
Browser	Internet Explorer 9

1.3 Performance als Business-Case

Im Zuge der Recherchen für diese Arbeit ist dem Autor bewusst geworden, wie relevant das Thema Performance bei der Umsetzung von erfolgreichen Webprojekten ist. In einem Expertengespräch mit André von Deetzen und Thomas Lehmann hat sich dann heraus kristallisiert, dass dieses Thema vorrangig im Umfeld der technisch belasteten Arbeitsfelder Beachtung findet. Dieser Umstand führt dazu, dass in Projekten bei denen die Ressourcen knapp bemessen sind, das Thema Performance eine eher untergeordnete Rolle spielt und somit die Qualität der Produkte leidet.

⁷ siehe Anhang A: Verwendete Tools

In diesem Gespräch waren sich alle Teilnehmer einig, dass es sinnvoll wäre das Thema Performance in Zukunft als einen Business-Case zu betrachten. Profitorientierte Abteilungen, bei denen Performance bisher kein Thema war, sollten hinreichend sensibilisiert sein, um dieses Feature zu akzeptieren. Das Marketing könnte es z.B. als Argumentation für einen Mehrwert des Produktes verwenden.

Des Weiteren hat sich in diesem Gespräch ein Problem mit der Zuständigkeit ergeben. Es gibt keine konkrete Person die verantwortlich ist, dass eine gute Performance am Ende eines Projektes gewährleistet werden kann.

Als Lösung wäre, nach André von Deetzens Auffassung, in Zukunft der Produktmanager für die Performance zuständig. Denn dieser ist normalerweise verantwortlich für die Projektfeatures.

In dem Interview ist außerdem noch die Frage aufgekommen, an welcher Stelle innerhalb des Projektablaufs ein guter Zeitpunkt für die Performancebetrachtung wäre.

Es hat sich herausgestellt, dass diese, wie jedes andere Feature auch, in die Entwicklungsphase gehört. Also je nach Vorgehensmodell ständig Aufmerksamkeit genießt und auf „Funktionalität“ bzw. Qualität hin überprüft wird. Eine Optimierung wie sie häufig am Ende eines Projektes gemacht wird, wäre damit hinfällig.

Die Umsetzung dieser Überlegungen ist natürlich daran gekoppelt, dass verantwortliche Entscheidungsträger hinreichend auf das Thema sensibilisiert sind und einsehen, dass dies in Zukunft ein wichtiger Faktor für qualitativ hochwertige Projekte ist und in Zukunft einen Mehrwert in der Kundenwahrnehmung darstellt (Lehmann, von Deetzen & Kampe, 2012).

2 Ladevorgang einer Webseite

Um die Optimierungsmaßnahmen im Kapitel 5 richtig einordnen und verstehen zu können, ist ein grundsätzliches Verständnis der Technologiekonzepte hinter dem World Wide Web wichtig. Aus diesem Grund beleuchtet dieser Abschnitt den Ladevorgang einer Webseite genauer und zeigt mögliche Geschwindigkeitsflaschenhälse auf.

Abbildung 2 zeigt die technischen Abläufe während ein User die Webseite besucht.

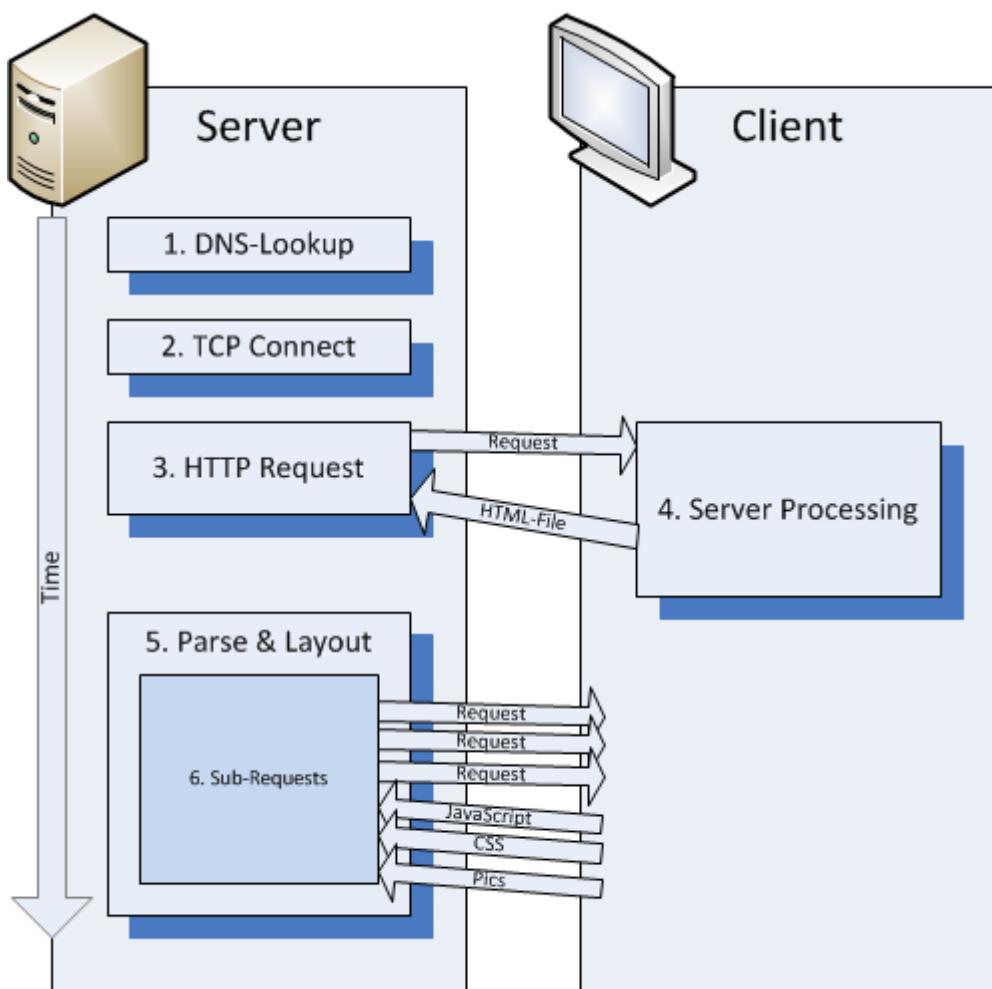


Abbildung 2: Technischer Ablauf eines Webseitenaufrufs

1. Zunächst wird mit Hilfe eines DNS-Lookups herausgefunden wie die IP-Adresse zu dem Server mit der gewünschten Seite lautet.
2. Mit Hilfe der IP-Adresse wird nun eine TCP-Verbindung zum Server aufgebaut. Dieser Verbindungsauflauf ist, wenn man die Zeitkosten betrachtet, relativ teuer.

Für einen Verbindungsaufbau müssen 3 Anfragen⁸ über das Netzwerk gesendet und deren Antworten abgewartet werden. Wenn die Round-Trip-Time also 50ms beträgt, dauert ein Verbindungsaufbau 150ms.

3. Danach kann auf Anwendungsebene eine Verbindung über das HTTP-Protokoll zum Server aufgebaut werden und die Anfrage des Clients nach einer bestimmten Webseite an den Server gesendet werden.
4. Dieser verarbeitet diese Anfrage dann und gibt ein HTML-Dokument als Antwort zurück.
5. Der Browser des Clients nimmt dieses Dokument entgegen und beginnt mit dem Rendern der Seite.
6. Dabei stößt er auf Komponenten, wie z.B. JavaScript, CSS, Bilder oder Flash, aus denen die Seite besteht. Diese Dateien fragt er nun ebenfalls vom Server ab und muss dafür jeweils einen extra HTTP-Request senden.

2.1 Wo bleibt die Zeit?

Eine etwas andere Betrachtung des Seitenladevorgangs visualisiert die Aufgaben und Komponenten, die für Performanceeinbußen verantwortlich sein können. Im Wesentlichen gibt es beim Abruf einer Webseite drei beteiligte technische Komponenten. Den Client, damit ist der Browser welcher vom Benutzer gesteuert wird gemeint. Das Netzwerk als Kommunikationsmedium und den Server, welcher die angeforderte Seite zur Verfügung stellt.

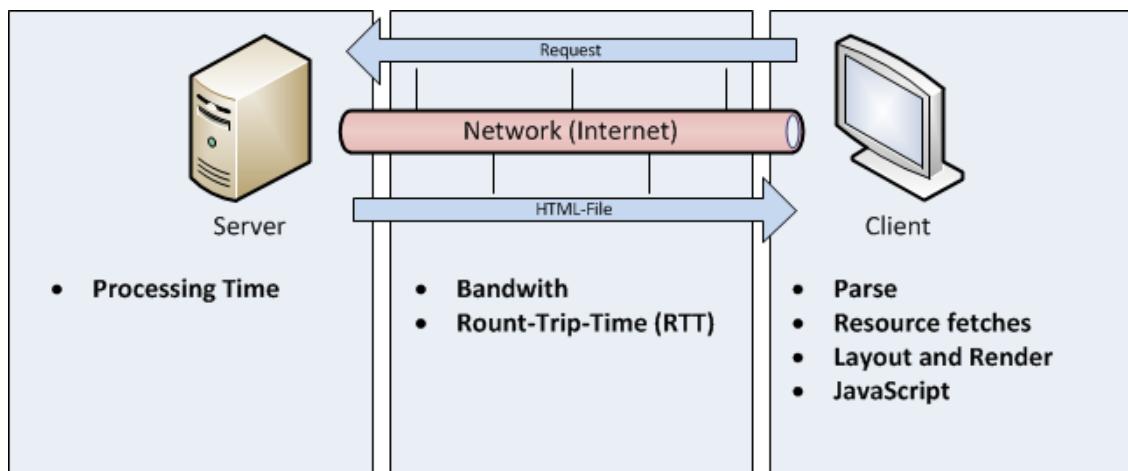


Abbildung 3: Aufgabenverteilung beim Seitenladevorgang

⁸ Der Drei-Wege-Handschlag (Three-Way-Handshake) ist das typische Verfahren zum Aufbauen einer TCP-Verbindung.

Jede einzelne dieser Aufgaben kostet Zeit und bietet Potenzial für Optimierungen. Wie lang jede einzelne Aufgabe dauert, ist je nach technischer Realisierung, von Projekt zu Projekt unterschiedlich. Tendenziell liegt allerdings im Frontend das größte Potenzial verborgen (s. Kapitel 3.2 „Performance Golden Rule“).

2.1.1 Processing Time

Die *Processing Time* (Verarbeitungszeit) beginnt, sobald die Anfrage nach einer Webseite bei dem richtigen Server angekommen ist und endet damit, dass der Server das angeforderte HTML-Dokument an den Client zurück sendet. Während dieser Zeit passieren projektabhängig unterschiedliche Dinge. Oft wird auch der Begriff *First-Byte-Time* in diesem Zusammenhang verwendet, er beschreibt die Zeit zwischen dem Absenden des Initial-Requests vom Browser und dem Ankommen des ersten Bytes wiederum beim Browser.

Mögliche Szenarien, die sich auf dem Server abspielen wenn ein Page-Request ihn erreicht, könnten beispielsweise folgende sein:

- Es müssen für die angeforderte Seite Daten aus einer Datenbank geholt, anschließend verarbeitet und in ein HTML-Dokument eingebettet werden.
- Oder aber das HTML-Dokument liegt bereits fertig gerendert im Cache des Servers und muss nur noch zurück zum Client gesendet werden.

Die Letzte der beiden Varianten ist natürlich wesentlich schneller.

2.1.2 Bandbreite und Round-Trip-Time

Die Werbemaschinerien der Internet-Provider versuchen es dem Kunden der einen Internetanschluss sucht, möglichst einfach zu machen. So werben sie fast ausschließlich mit hohen Bandbreiten⁹, mittlerweile werden Verbindungsgeschwindigkeiten von bis zu 50Mbps angepriesen. Da der Großteil der Internetbenutzer ihren Anschluss allerdings nur für das Surfen von Webseiten, E-Mail- und Instant-Messaging-Dienste nutzt, wird das Potenzial dieser hohen Bandbreiten selten ausgenutzt. Der Glaube eine schnellere Internetverbindung sorge dafür, dass die Webseiten schneller laden, ist weit verbreitet, aber nur teilweise richtig.

Neben der Bandbreite gibt es eine weitere wichtige Stellgröße, die maßgeblich für die Ladegeschwindigkeit von Webseiten verantwortlich ist, die Round-Trip-Time (RTT). So wird die Zeit bezeichnet, die ein Datenpaket in einem Computernetzwerk braucht, um von der Quelle (Client) zum Ziel (Server) und wieder zurück übertragen zu werden.

⁹ oder auch Datenübertragungsrate

Bei einem Vergleich einer Internetverbindung mit einem Wasserrohrsystem wird klar, wie die beiden Stellgrößen zusammen hängen. Angenommen die Bandbreite wäre der Durchmesser der Rohre, dann könnte umso mehr Wasser zwischen zwei Punkten transportiert werden, je größer der Durchmesser ist. Vorausgesetzt es ist immer Wasser in der Leitung, kann zu jedem Zeitpunkt immer die volle Menge an Wasser aus dem System entnommen werden. Ist allerdings kein Wasser in den Leitungen, braucht es eine gewisse Zeit um am Ziel anzukommen. Je nachdem, welchen Weg es durch das Rohrsystem nimmt und wie viel Strecke es dabei zurücklegen muss, variiert diese. Verglichen mit dem Internet wird die Zeit, die das Wasser braucht um von Punkt A nach Punkt B und zurück zu gelangen, die Round-Trip-Time genannt.

Besucht ein Benutzer aus den USA eine in Deutschland gehostete Webseite, ist eine RTT von 125ms normal. Innerhalb Deutschlands sind RTTs bis 50ms üblich (Wikipedia - Paketumlaufzeit, 2012).

Ein Test von Mike Belshe zeigt, wie sich die Reduzierung der Round-Trip-Time auf die Gesamtladezeit einer Seite auswirkt. Für den Test wurden 25 der bekanntesten Webseiten analysiert und eine feste Bandbreite von 5Mbps verwendet.

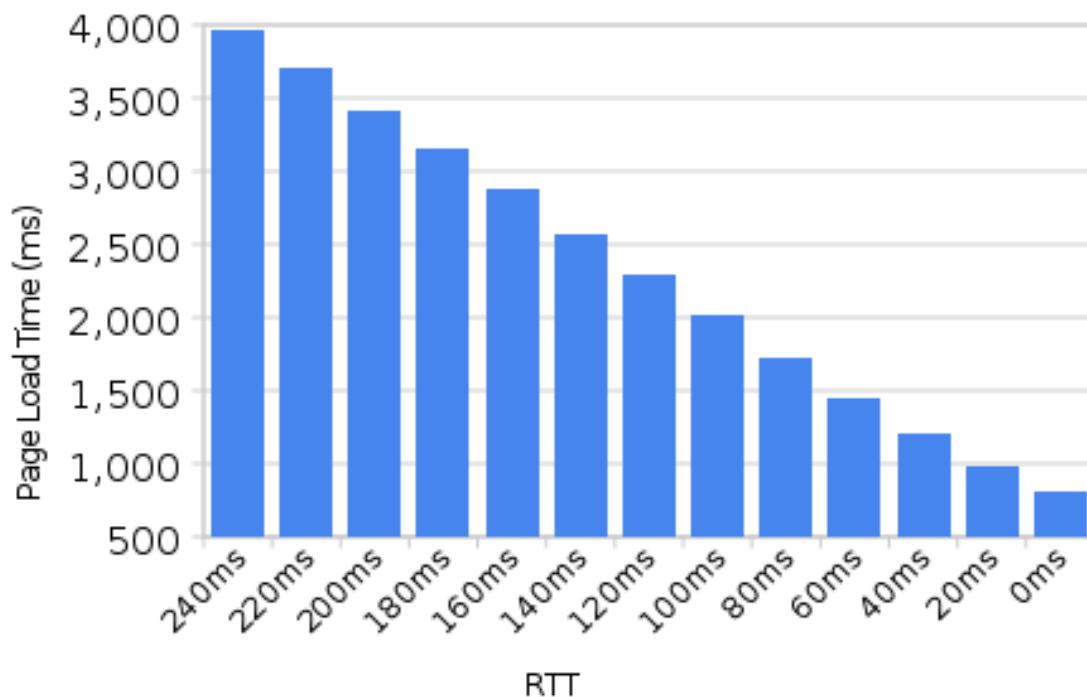


Abbildung 4: Auswirkung der Round-Trip-Time auf die Seitenladezeit (Belshe, 2010)

Abbildung 4 zeigt den unmittelbaren Zusammenhang von Round-Trip-Time und Seitenladezeit. Die schrittweise Reduzierung der RTT sorgt für eine kontinuierliche Verbesserung der Seitenladezeit.

Ein weiteres Performance-Problem ist die Architektur des TCP-Protokolls. Auf der Transportschicht ist es dafür zuständig eine Verbindung zwischen zwei Kommunikationspartnern herzustellen und dafür zu sorgen dass die Datenpakete beim Ziel ankommen. Das Problem ist, dass dieses Protokoll nicht von vornherein weiß, wie groß die Bandbreite der Verbindung ist. Um das Netz nicht zu überlasten, sorgt ein Algorithmus¹⁰ für die schrittweise Annäherung an die maximal zur Verfügung stehende Bandbreite.

Für die Übertragung von großen Dateien ist das ein gutes Verfahren, denn hier kann eine große Bandbreite ihre Vorteile ausspielen. Im Falle einer Webseite, die aus vielen Einzelverbindungen für kleine Dateien besteht, ist das allerdings nicht sehr vorteilhaft. Bevor die Bandbreite voll ausgenutzt werden kann, sind die Dateien bereits vollständig übertragen.

Auch hier visualisiert ein Test von Mike Belshe wie effektiv sich eine höhere Bandbreite auf die Seitenladezeit auswirkt.

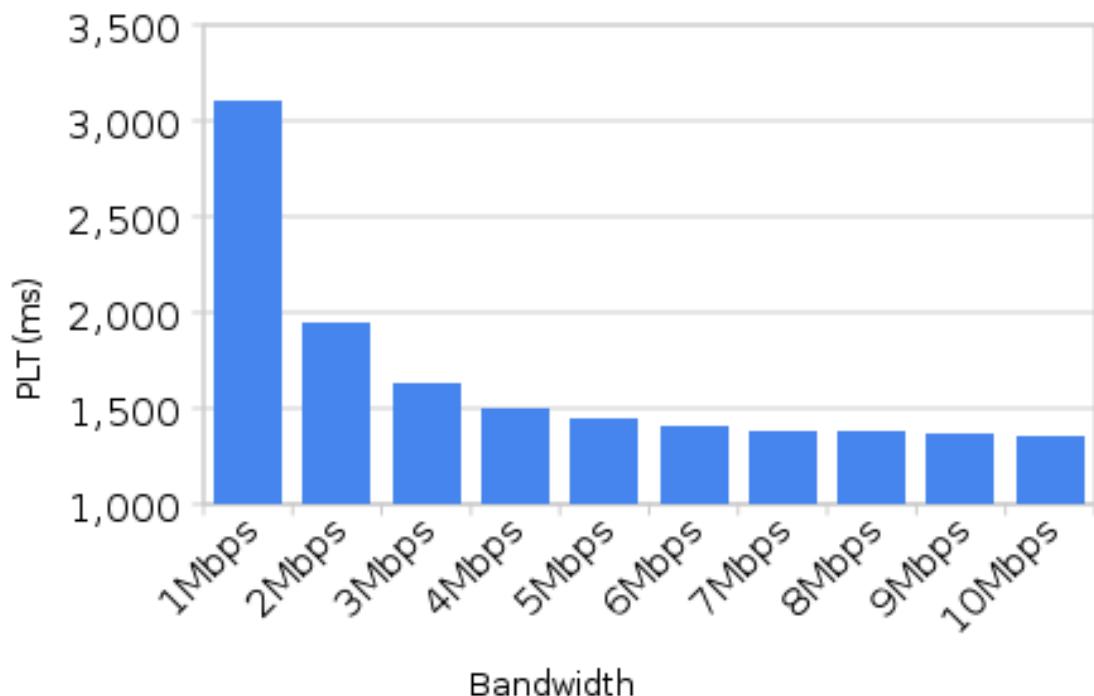


Abbildung 5: Auswirkungen der Bandbreite auf die Seitenladezeit (Belshe, 2010)

Aus der Abbildung 5 ist ersichtlich, dass eine Bandbreite zwischen 4-6Mbps für das Surfen auf Webseiten absolut ausreichend ist und jeder Megabit mehr nur marginale Performanceverbesserungen bietet.

¹⁰ Die beiden hier angesprochenen Verfahren: Slow-Start-Algorithmus und Congestion Avoidance.

2.1.3 Client-Side

Sobald das erste Byte der HTML-Datei beim Client-Browser ankommt, endet die sogenannte *Time-To-First-Byte* oder auch *First-Byte-Time*. Dies ist ein wichtiger Zeitpunkt im „Leben“ einer Webseite, da hier die Processing-Time des Servers endet und der Browser den Rest des Seitenaufbaus übernimmt. Mit Blick auf die „Performance Golden Rule“¹¹ beginnen nun die 80-90% der Seitenladezeit.

Schon während die HTML-Datei beim Client ankommt, beginnt der Browser diese zu parsen und schaut sich dabei an, welche externen Dateien in dem Dokument referenziert sind. Diese werden dann mit Hilfe von HTTP-Requests nachgeladen. Die Reihenfolge in der dies geschieht legt deren Position innerhalb der HTML-Struktur fest. Ein Stylesheet, das im Header referenziert ist wird z.B. vor einem JavaScript am Ende des Bodys geladen.

Nachdem der Browser mit dem Parsen des Headers fertig ist, beginnt er das Layout zu berechnen und zeichnet die ersten Bilder der Webseite, der Benutzer kann an dieser Stelle das erste Mal wahrnehmen, dass sich die Webseite aufbaut. In diesem Prozess werden auch die geladenen JavaScript-Dateien interpretiert und angewendet.

¹¹ siehe 3.2 „Performance Golden Rule“

3 Tweaks zur Steigerung der Performance

Um eine Webanwendung auf ihre Performance hin zu optimieren, gibt es eine Reihe von Best Practices und Technologien, nachfolgend vereinheitlicht Tweaks¹² genannt.

Bevor allerdings genauer darauf eingegangen wird, sei erwähnt, dass die Grundvoraussetzung für jede Optimierung ein strukturierter und validierter Quelltext ist, das betrifft nicht nur HTML-Dokumente, sondern auch CSS- und JavaScript-Code.

Eine gute Struktur trägt dazu bei, dass nachträgliche Änderungen nicht zu ungewollten Seiteneffekten führen, Fehler schneller gefunden werden und hilft neuen Projektbeteiligten, sich schnell in einem Projekt zurechtzufinden.

Für das Schreiben von HTML gibt es klare Regeln, welche im DOCTYPE¹³ einer Seite definiert sind. Das kann für HTML1.1 z.B. die Regel sein, dass sich keine DIV-Elemente innerhalb eines A-Tags befinden dürfen, im HTML5-Standard ist dies wiederum erlaubt. Da es keine Regeln dafür gibt, wie die unterschiedlichen Browser mit invalidem Code umgehen, kann dieser zu Performanceproblemen und Darstellungsfehlern führen.

Das World Wide Web Consortium (W3C) ist eine internationale Gemeinschaft, welche die Web-Standards für viele Technologien, die dem Internet zu Grunde liegen, definiert. Auf der Webseite des W3C ist es möglich seinen HTML-¹⁴ und CSS-Code¹⁵ auf Validität zu testen. Treten Fehler während des Tests auf, werden diese erläutert und Tipps für die Behebung gegeben.

Für JavaScript gibt es von Douglas Crockford auch ein Web-Tool¹⁶, welches den Code auf Qualitätsstandards hin überprüft.

3.1 Content

Bei aller Komplexität heutiger Webanwendungen bestehen diese, aus Sicht des Browsers, grundsätzlich immer aus einer HTML-Datei, in welcher die Inhalte strukturiert sind. In diesem Abschnitt geht es darum, was der Entwickler in Bezug auf diese Datei beachten sollte, bzw. besser machen kann.

¹² Hier im Sinne des englischen Verbs „to tweak“ zu verstehen. Bedeutet justieren, optimieren, frisieren.

¹³ Die „Document Type Definition“ oder DOCTYPE gibt die Struktur eines Dokumentes vor.

¹⁴ HTML-Validator: <http://validator.w3.org/>

¹⁵ CSS-Validator: <http://jigsaw.w3.org/css-validator/>

¹⁶ JavaScript-Analyse: <http://www.jslint.com/>

3.1.1 HTTP-Requests minimieren

Die meisten Webseiten bestehen aus einer Vielzahl von externen Dateien, z.B. Bilder, Flash, JavaScript oder CSS, diese sind im HTML der Seite referenziert und werden im Zuge des Seitenladevorgangs vom Webserver geladen. Für jede einzelne Datei muss dafür ein eigener HTTP-Request an den Server gestellt werden, welcher daraufhin die gewünschte Datei an den Browser zurücksendet.

Die Dauer für das Nachladen einer Komponente ist abhängig von mehreren Faktoren:

- Evtl. Aufbau einer TCP-Verbindung zum Server
- Round-Trip-Time zwischen Server und Client
- Bandbreite
- Größe der herunterzuladenden Datei

Gehen wir davon aus, dass die TCP-Verbindung noch weiter genutzt werden kann, (HTTP Keep-Alive) muss dennoch für jede nachzuladende Komponente mindestens eine RTT eingeplant werden, welche innerhalb Deutschlands bis zu 50ms dauern kann. Würde bedeuten, bei 80 externen Komponenten machen allein die Anfragen an den Server eine Zeit von vier Sekunden aus. Dieser Wert ist natürlich fiktiv, da aktuelle Browser ja auch sechs Verbindungen gleichzeitig aufbauen können. Aber es sollte beachtet werden, dass so ein Request immer Zeit kostet.

3.1.2 Minimieren und Zusammenfassen von JavaScript und CSS

CSS und JavaScript sind Skript-Sprachen die nicht vor dem Ausführen durch den Browser kompiliert werden, dementsprechend direkt interpretiert werden. Das sorgt dafür, dass viele Entwickler ihren Code aus der Entwicklung direkt für die Produktivseite übernehmen. Kommentare, Leerzeilen oder lange Variablennamen sind allerdings unnötig und können vor dem produktiven Einsatz entfernt werden.

Um diese Dateien automatisiert aufzuräumen und alles Unnötige rauszuwerfen gibt es entsprechende Tools, die diese Arbeit erledigen. Ein online Tool für JavaScript ist z.B. jscompress¹⁷.

¹⁷ <http://jscompress.com/>

Als Beispiel hier ein kurzer JavaScript-Code, in einer Form wie er bei der Entwicklung vorliegt.

```

1  var addEvent = function( obj, type, fn ) {
2      if (obj.addEventListener)
3          obj.addEventListener(type, fn, false);
4      else if (obj.attachEvent)
5          // some comments are often in here
6          obj.attachEvent('on' + type, function() {
7              return fn.apply(obj, new Array(window.event));
8          });
9      }

```

Quellcode 1: nicht minifiziertes JavaScript

Nachdem nun das Minimierungs-Tool den Code verarbeitet hat, sieht dieser wie folgt aus (Quellcode 2).

```

1  var addEvent=function(a,b,c){if(a.addEventListener)a.addEventListener
(b,c,false);else if(a.attachEvent)a.attachEvent("on"+b,function()
{return c.apply(a,new Array(window.event))})}

```

Quellcode 2: minifiziertes JavaScript

Für CSS-Dateien gibt es vergleichbare Werkzeuge, wie z.B. MinifyCSS¹⁸, die ähnliches leisten. Die dadurch reduzierte Größe der Dateien wirkt sich positiv auf die Performance aus, auch wenn dieser Effekt nicht gut messbar ist, da in der Regel nur einige Kilobyte dabei gespart werden können.

Ein weiteres Problem ist die Modularisierung von CSS und JS in verschiedene Dateien. Das sorgt einerseits für einen guten Überblick und eine gute Erweiterbarkeit des Codes aber andererseits erhöht es auch die Anzahl der Requests.

Für Abhilfe sollte an dieser Stelle, im Falle von CSS die @import-Direktive sorgen. Mit ihr ist es möglich CSS-Dateien ineinander zu verschachteln. Stößt aber der Browser auf so eine eingebundene CSS-Datei, stellt er sofort einen neuen Request und lädt die Datei nach. Um in diesem Bereich Zeit zu sparen, bleibt nur, den Code in möglichst wenige Dateien zu bündeln.

¹⁸ <http://www.minifycss.com/css-compressor/>

3.1.3 DNS-Lookups minimieren und HTTP-Requests parallelisieren

Ein DNS-Lookup löst einen Hostnamen (z.B. www.beispiel.de) in eine IP-Adresse auf, mit dieser kann dann der Server kontaktiert werden und eine HTTP-Verbindung aufgebaut werden.

Ist der Hostname zuvor noch nicht aufgelöst worden und steht somit noch nicht im Cache des Benutzers, kann eine Abfrage zwischen 300 und 400ms dauern. Diese muss der Browser für alle eindeutigen Hostnamen innerhalb eines Seitenaufrufs machen, deshalb sollte sparsam mit DNS-Lookups umgegangen werden.

Warum dann aber nicht einfach alle Komponenten über einen Hostnamen beziehen?

Die Antwort auf diese Frage scheint ein wenig veraltet zu sein, hat aber dennoch einen aktuellen Bezug. Jeder Browser ist in der Lage mehrere Ressourcen, wie CSS- oder JavaScript-Dateien, parallel von einem Server zu beziehen. In der HTTP/1.1 Spezifikation ist dazu verankert, dass nicht mehr als zwei Downloads pro Hostname gleichzeitig gestartet werden sollten. Diese Spezifikation ist aus dem Jahr 1999. Damals hätte ein höherer Wert dazu geführt, dass die Systeme der Benutzer aufgrund mangelnder Hardware-Ressourcen instabil werden.

Da sich im Laufe der letzten 13 Jahre allerdings viel an der User-Hardware geändert hat und auch die Browser deutlich stabiler geworden sind, konnte die Barriere von zwei parallelen Downloads ignoriert werden. Dafür mussten die Komponenten einfach von verschiedenen Hostnamen bezogen werden. Also z.B. www.beispiel.de liefert die HTML-Datei, das CSS und JavaScript aus und von static.beispiel.de werden die Bilder bezogen. Es kommt dabei nicht darauf an ob die IP-Adresse variiert, der Browser limitiert lediglich die maximale Anzahl der Downloads pro Hostnamen. Die Daten können also auf einem Server liegen.

Die Kehrseite dieses Tweaks ist der DNS-Request, denn dieser muss für jeden eindeutigen Hostnamen separat gemacht werden und kostet Zeit. Außerdem wird die CPU des Benutzers mit jedem weiteren Download stärker belastet, was sich indirekt auch auf die Performance auswirken kann. Es gilt also einen Zwischenweg zu finden, es sollten so viele parallele Downloads wie möglich gestartet werden, ohne das die DNS-Lookups den daraus resultierenden Geschwindigkeitsvorteil zu Nichte machen.

Die Studie „Maximizing Parallel Downloads in the Carpool Lane“ von Tenni Theurer hat sich mit dieser Problematik auseinandergesetzt. Er empfiehlt daraufhin, dass der Inhalt einer Webseite auf 2 bis 4 Hostnamen verteilt werden sollte, da jeder weitere die Performance eher verschlechtert bzw. nicht deutlich verbessert (Theurer, 2007, Part 4).

Mittlerweile haben aber die Browserhersteller das Problem erkannt und die Beschränkung der maximalen parallelen Downloads auf sechs angehoben. Nachstehende Tabelle gibt darüber Aufschluss.

Tabelle 4: aktuelle Browser und deren maximalen Verbindungen pro Server

Browser	max. Verbindungen
Internet Explorer 7 und älter	2
Internet Explorer 8	6
Internet Explorer 9	6
Firefox 9	6
Google Chrome 17	6
Opera	6

Das bedeutet der Tweak ist für alle neuen Browser nicht so sehr relevant, da diese ohnehin sechs Komponenten parallel downloaden. Der Webentwickler von heute bräuchte sich also eigentlich keine Gedanken um das Thema machen, wäre da nicht der Internet Explorer. Da er in den Versionen kleiner als 8 noch einen signifikanten Marktanteil hat, muss er für die meisten Projekte immer noch berücksichtigt werden.

3.1.4 Seitenaufbau planen

Der Aufbau der HTML-Datei ist maßgeblich dafür verantwortlich, wie der Benutzer den Ladevorgang der Seite wahrnimmt. Einige Sekunden können als sehr lang wahrgenommen werden, wenn nichts passiert. Sieht der Benutzer allerdings nach einer Sekunde bereits das Logo und nach dreien das Menü, so ist er nicht gelangweilt und kann sich mit den bereits sichtbaren Inhalten beschäftigen.

Um das zu realisieren sollten zunächst alle Komponenten geladen werden, die für die Initial-Darstellung gebraucht werden. Das heißt, Bilder die erst ersichtlich werden wenn das Menü aufklappt oder JavaScript-Dateien welche für Animationen zuständig sind, brauchen erst zum Schluss geladen werden.

JavaScript sollte außerdem nicht im Head eines HTML-Dokumentes referenziert sein. Lädt der Browser nämlich eine JS-Datei, so startet er keinen weiteren Download, sondern konzentriert sich zunächst auf die Auswertung des Scripts. Es könnte nämlich sein, dass dieses eine DOM-Manipulation vornimmt und somit den weiteren Ladeprozess verändert. JavaScript-Dateien blockieren also den Ladevorgang, deshalb sollten diese soweit es möglich ist erst am Ende, also kurz vor dem schließenden BODY-Tag, eingebunden werden.

CSS-Definitionen sind für die Darstellung der Seiteninhalte verantwortlich und deshalb wichtig für den Render-Prozess einer Webseite. Diese sollten nach Möglichkeit im HEAD-Tag eingebunden werden, um dem Browser möglichst früh zur Verfügung zu stehen.

3.1.5 Redirects verhindern

Wird vom Benutzer eine Seite aufgerufen, die auf dem Server keinen Inhalt repräsentiert, gibt es zwei Möglichkeiten wie der Server darauf reagiert. Er antwortet mit einem „404 Page not Found“ Status Code oder aber er verweist auf eine andere Seite. In diesem Fall antwortet er mit dem Status Code „301 Moved Permanently“. In seinem Antwort-Header steht dann im Location-Attribut das neue Ziel. Der Browser stellt also erneut eine Anfrage an diese Adresse.

Diese Redirects werden auch häufig verwendet um den Besucher, abhängig von seiner Herkunft oder anderer Parameter, direkt auf eine spezielle Landing-Page zu dirigieren. Dabei sollte klar sein, dass jeder Redirect einen HTTP-Request und somit einen vollen Round-Trip noch innerhalb der First-Byte-Time bedeutet.

Ein typischer Redirect ist die Umleitung des Besuchers von <http://beispiel.com/> nach <http://www.beispiel.com/>.

3.1.6 Slash in Links

Bei der Auszeichnung von Hyperlinks, die auf ein Verzeichnis zeigen, sollte am Ende immer ein Slash angehangen sein. Also statt <http://www.beispiel.de/folder> sollte im HTML-Code immer <http://www.beispiel.de/folder/> stehen. Der Webserver interpretiert nämlich alles nach dem letzten Slash in einer URL als Datei, da er aber keine Datei namens „folder“ im Dateisystem finden kann, gibt er den Status-Code „301 Moved Permanently“ zurück und verweist dann auf die URL <http://www.beispiel.de/folder/>. Welche der Browser allerdings erneut anfragen muss. Dieser unnötige Round-Trip sollte gespart werden.

3.1.7 DOM-Elemente: Weniger ist mehr schneller!

Viele der heutigen Webprojekte werden sehr aufwändig und mit viel Liebe zum Detail von den Grafikabteilungen gestaltet. Diese komplexen Layouts umzusetzen, stellt die Frontend-Entwickler häufig vor eine Gewissensfrage. Denn sie wissen, je komplexer das DOM wird, desto länger braucht auch der Browser um dieses zu rendern.

Da die Designvorgaben so individuell verschieden sind, kann kein konkretes Vorgehensmodell für die Umsetzung definiert werden. Es ist allerdings ratsam, sich immer nach Best-Practices umzusehen und nicht wahllos immer noch ein DIV in das nächste zu schachteln. Je flacher die DOM-Hierarchie, desto schneller kann die Seite gerendert werden.

3.2 Bilder

Wie wichtig die Betrachtung der Bilder bei einer Optimierung ist, zeigt eine Untersuchung bei der die 10 populärsten Webseiten Deutschlands untersucht wurden. Demnach ist im Schnitt über die Hälfte (56,44%) des Gesamtvolumens dieser Webseiten auf Bilder zurückzuführen (siehe Anhang C).

Als eine gute Strategie mit Bildern umzugehen, haben sich zwei Phasen bewiesen.

1. In der ersten muss eine Entscheidung darüber getroffen werden wie viele Farben, welche Auflösung und welche Qualität das Bild haben soll. Die Einschränkung dieser Parameter sorgt für einen gewollten Verlust von Bildinformationen. Denn wenn Informationen, die nicht für die Darstellung im Web relevant sind wegfallen, reduziert sich die Größe des Bildes. Dies ist ein kreativer Prozess der individuell für jedes Bild entschieden werden sollte. Jedes Bild hat unterschiedliche Voraussetzungen, eine Grafik in der ein Balkendiagramm diverse Zahlen visualisiert ist anders zu behandeln als das letzte Urlaubsfoto.
2. Im zweiten Schritt, nachdem die Qualitätsentscheidung getroffen wurde, sollten noch verlustfreie Kompressionen angewandt werden, um auch die letzten unnötigen Bytes aus den Bildern zu verbannen. Für diese Aufgabe bietet es sich an, automatisiert vorzugehen, da die Qualität bei diesem Vorgang nicht beeinflusst wird.

3.2.1 Bild-Formate

Im Internet sind drei wichtige Bild-Formate verbreitet, alle haben ihre Vor- und Nachteile, welche individuell abgewogen werden müssen.

Das **JPEG-Format** ist ein verlustbehaftetes Format. Es ist möglich beim Speichern eine Qualitätsstufe von 0 bis 100 anzugeben. Dabei verliert das Bild allerdings sogar bei einer Stufe von 100 ein wenig Qualität. Deshalb sollte ein Bild, welches häufig geändert wird, in einem verlustfreien Format¹⁹ abgespeichert werden. Es wird für Bilder verwendet, in denen viele verschiedene Farben sind z.B. das letzte Urlaubsfoto oder aber auch Grafiken mit Farbverläufen.

Das **PNG-Format** war, nach Patentproblemen, als Ersatz für das GIF-Format gedacht. Es ist verlustfrei, unterstützt Alpha-Transparenz und bietet in den Versionen PNG8, PNG24 und PNG32 verschiedene große Farbpaletten an um die Dateigröße möglichst gering zu halten. Es wird häufig für Grafiken mit wenigen Farben, wie z.B. ein Diagramm oder Icon, verwendet.

GIF ist wie PNG ebenfalls ein verlustfreies Format und kommt in ähnlichen Situationen zum Einsatz. Der größte Unterschied liegt darin, dass es Animationen unterstützt, also kleine daumenkinoartige Bildabfolgen. Es unterstützt Transparenz, allerdings nur zwei Zustände, transparent oder nicht transparent. Die Alpha-Transparenz von PNG kennt an dieser Stelle noch Zwischenzustände.

Steht die Wahl zwischen PNG und GIF, sollte auf PNG zurückgegriffen werden, da dessen Komprimierungsalgorithmus etwas besser ist.

3.2.2 Automatisierte verlustfreie Kompression

Jedes Grafik-Format hat seine Eigenheiten, so ist es beispielsweise möglich, in einem JPEG-Bild den Ort, die Zeit, verwendete Kamera und noch viel mehr Informationen, abzuspeichern. Diese sind für die Darstellung eines Bildes auf einer Webseite aber nicht erforderlich. Um solche Bereinigungen durchzuführen gibt es Werkzeuge wie smush.it²⁰, welche diese Informationen aus den Bildern verbannen und somit die Größe weiter reduzieren.

¹⁹ verlustfreie Formate wie: PNG, GIF oder BMP

²⁰ <http://www.smushit.com/>

3.2.3 Größenangaben im img-Tag

Werden Bilder im HTML-Markup in einem IMG-Tag referenziert, sollten dabei immer Höhe und Breite mit angegeben werden. So wie in Quellcode 3 zu sehen, die Höhe und Breite sind über die Attribute *width* und *height* angegeben.

```
1 ...
2 
3 ...
```

Quellcode 3: HTML img-Tag

Wird das nicht getan, muss der Browser die Auflösung des Bildes während des Render-Prozesses erst ermitteln, bevor er es in das Layout einsetzen kann. Dieser Vorgang bedeutet zusätzlichen Aufwand und verzögert so den Seitenaufbau.

Außerdem sollten Bilder nicht skaliert eingesetzt werden, also in einer anderen Größe dargestellt werden, als im Original festgelegt. Das führt dazu, dass der Browser das Bild zunächst in die neue Größe umrechnen muss, bevor er es einsetzen kann.

3.2.4 Inline Bilder

Das Data-URI-Schema²¹ ist entwickelt worden, um kleine Daten-Objekte direkt in eine URL zu integrieren, ohne auf eine externe Quelle zu verweisen. Das bietet die Möglichkeit kleine Bilder im Base64-Format zu codieren und unmittelbar in das Markup zu integrieren. Durch dieses Verfahren können zeitaufwändige HTTP-Requests für kleine Bilder gespart werden.

```
1 
```

Quellcode 4: Data-URI Format innerhalb eines HTML *img*-Tags



²¹ Definiert von RFC 2397: <http://www.ietf.org/rfc/rfc2397>

Rechts unten im Quellcode 4 ist ersichtlich welches Bild dieser Code repräsentiert. Am Anfang des *src*-Attributes sind die Parameter für das Data-URI-Objekt zu erkennen, es handelt sich in diesem Fall um Daten vom Typ *image/gif* in *base64* Kodierung. Die Anwendung dieses Verfahrens ist auch in CSS-Dateien möglich.

Nachteile dieser Technologie:

- Externe Ressourcen im HTML können wiederverwendet werden was mit dieser Technik nicht möglich ist
- Der Internet Explorer unterstützt es erst ab der Version 8
- base64 codierte Bilder sind 33% größer als binär codierte

3.3 JavaScript

Konnten Webseiten früher noch gänzlich ohne JavaScript auskommen, sind die heutigen Anforderungen an eine moderne Webseite ohne diese Sprache nicht mehr denkbar. Sie wird verwendet um statistische Daten zu sammeln, dynamische Inhalte nachzuladen, visuelle Effekte zu erzeugen oder die Benutzerinteraktion dynamischer zu gestalten.

Es ergeben sich aus dieser Technologie viele Vorteile und der Erfolg spricht für sich, dennoch sollten beim Einsatz von JavaScript die Performancefresser nicht außer Acht gelassen werden.

3.3.1 Skripte extern

In Kapitel 5.1.1 „HTTP-Requests minimieren“ habe ich erwähnt, dass jeder Request den Overhead einer RTT mit sich bringt. Warum werden JavaScript-Dateien (und auch CSS-Dateien) dann eigentlich nicht einfach in das HTML-Dokument integriert und somit gar kein zusätzlicher Request erzeugt? Nun, diese Überlegung ist nicht falsch und kann durchaus auch einen Sinn ergeben, grundsätzlich ist es allerdings besser diese Dateien extern zu halten, da sie dann vom Browser des Users gecached werden können. Befinden sich diese innerhalb des HTML-Dokumentes, werden wie bei jedem Aufruf der Seite mitgeladen und machen somit das HTML-Dokument umfangreicher. Außerdem können Unterseiten von dem gecacheten JS und CSS profitieren, da diese bereits beim Aufruf der Startseite geladen wurden.

Sinn würde es machen ein JavaScript inline anzulegen, wenn beispielsweise auf einer Unterseite ein Bilder-Karussell integriert ist, welches sich nur auf dieser Seite befindet. Dann bräuchte es natürlich auch nur geladen werden, wenn diese Seite aufgerufen wird und nicht schon auf der Startseite.

Es muss also immer abgewogen werden für welche Seite das JavaScript relevant ist und ob es vielleicht für einen späteren Zeitpunkt noch einmal gebraucht wird. Die gleichen Überlegungen können auch auf die Einbindung von CSS-Dateien gemacht werden.

3.3.2 Skripte ans Ende

Stößt der Browser während des Parse-Vorgangs einer HTML-Datei auf eine JavaScript-Datei, so beginnt er damit, diese herunterzuladen und startet anschließend den JavaScript-Parser um den Code zu analysieren. Während dieses Prozesses stoppt der Browser aber das Parsen der HTML-Datei, denn es könnte ja sein, dass der JS-Code irgendetwas am DOM ändern will. JavaScript hat an dieser Stelle Priorität.

Das Problem dabei ist aber, dass dadurch der Seitenladevorgang ins Stocken gerät. Ist das JS im *head*-Bereich der Seite eingebunden, wird, solange der Browser sich damit beschäftigt, kein weiterer Dateidownload (z.B. CSS, Bilder, Flash) gestartet.

Aus diesem Grund gilt es als Best Practise, alle JavaScript-Dateien die nicht zeitkritisch sind, erst unmittelbar vor dem schließenden *body*-Tag einzubinden. Denn wenn der Parser hier angekommen ist, sind bereits alle Requests für die externen Dateien aus dem HTML-Dokument an den Server gestellt.

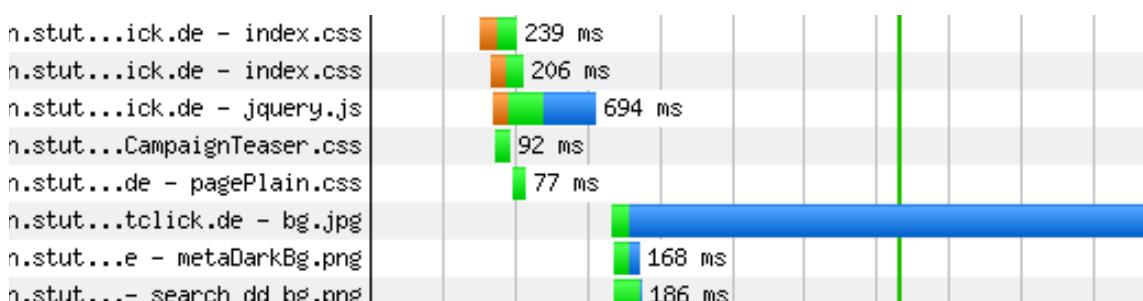


Abbildung 6: JavaScript blockiert Seitenladevorgang

Abbildung 6 zeigt ein Beispiel Wasserfall-Modell für eine blockierende JavaScript-Datei. Die Datei *jquery.js* wird geladen und als der Browser merkt, dass es sich um eine JavaScript-Datei handelt, startet er keinen weiteren Download, ehe er nicht mit dem Parsen fertig ist. Aus diesem Grund ist zwischen der CSS-Datei *pagePlain.css* und dem Bild *bg.jpg* eine Zeitspanne in der nichts passiert.

3.4 CSS

Cascading Style Sheet (CSS) ist eine Stylesheet-Sprache und wird verwendet, um das Aussehen von HTML-Dokumenten zu beschreiben. Sie ist vom W3C²² standardisiert²³ worden und liegt in mehreren Versionen vor. Die Version 1 ist mittlerweile obsolet, CSS2.1 wird empfohlen und die Version 3 befindet sich gerade im Standardisierungs-Prozess, kann aber in Verbindung mit HTML5 schon teilweise genutzt werden.

²² World Wide Web Consortium (W3C)

²³ <http://www.w3.org/Style/CSS/#specs>

Die CSS-Engine im Browser sorgt dafür, dass die festgelegten Regeln auf die Elemente innerhalb des HTML-Dokumentes angewendet werden. Dieser Prozess ist mittlerweile hochoptimiert und sorgt in der Praxis nur selten für große Performanceprobleme. Trotzdem ist es wichtig, ein Grundverständnis dafür zu haben, wie dieser Prozess funktioniert, worauf geachtet werden sollte und was zu Problemen führen kann.

Grundsätzlich gilt, für jede Regel die im CSS einer Seite definiert ist, versucht die CSS-Engine im DOM eine Entsprechung zu finden und diese anzuwenden. Also auch Regeln die auf gar kein Element im DOM zeigen werden überprüft und kosten Zeit. Häufig sind das Regeln, die für eine Unterseite definiert wurden aber schon beim Laden der Startseite mit ausgeliefert wurden. Oder aber sie wurden schlicht weg vom Entwickler im Code vergessen. Außerdem sollte vermieden werden Regeln zu definieren, die sich überschreiben, da dadurch ein unnötiger Aufwand für die Engine entsteht.

Effiziente Selektoren verwenden

Wenn der Browser das HTML-Dokument analysiert, baut er sich aus den Elementen einen virtuellen Baum, den sogenannten DOM-Tree²⁴, zusammen. Die CSS-Engine geht dann alle definierten Regeln durch und wendet sie auf die selektierten Elemente an. Wie lang nun dieses Auffinden der betreffenden Elemente dauert, kann vom Entwickler durch das Schreiben von effizienten Selektoren²⁵ gesteuert werden.

Dazu ist es zunächst wichtig zu verstehen, wie die Selektoren überhaupt von der Engine aufgelöst werden. Steve Souders betitelt dieses Kapitel in seinem Buch „Even Faster Websites“ mit „Rightmost First“, was soviel heißt wie „ganz rechts als erstes“.

„Rightmost First“

Steve Souders

Damit meint er, dass ein CSS-Selektor von ganz rechts angefangen nach links hin aufgelöst wird.

```
1 #main div p { font-color: red }
```

Quellcode 5: Beispiel CSS-Selector

²⁴ Document Object Model (DOM)

²⁵ Als Selektor wird der Teil einer CSS-Definition bezeichnet der bestimmt, auf welche Elemente die in den geschweiften Klammern stehenden Eigenschaften angewendet werden sollen.

Für das Beispiel im Quellcode 5 wird also zunächst das gesamte HTML-Dokument nach *p*-Tags gefiltert, danach schaut die CSS-Engine ob diese sich in einem *div*-Tag befinden und wenn dies ebenfalls zutrifft, wird im letzten Schritt noch geschaut ob sich die übrigen Elemente innerhalb des ID-Selektors *#main* befinden.

Gespräche mit Arbeitskollegen und Kommilitonen des Autors haben ergeben, dass gemeinhin die Annahme verbreitet ist, die Selektoren würden in Leserichtung aufgelöst, dies ist allerdings ein Irrglaube.

Mit einer optimierten Version lässt sich die CSS-Eigenschaft viel effizienter an die Ziel-Elemente heften. Dafür müssen die gewünschten *p*-Tags eine Klasse bekommen, welche dann als Selektor dient. Quellcode 6 zeigt das Resultat.

```
1 .paragraph-red { font-color: red }
```

Quellcode 6: Beispiel effizienterer CSS-Selektor

Der Aufwand ist sehr viel kleiner geworden, da das DOM nur noch einmal nach allen Elementen mit der Klasse *.paragraph-red* durchsucht werden muss.

Es folgt eine Liste von schnellen zu langsameren oder aber auch von speziellen zu allgemeineren Selektoren.

1. ID Rule: *#main*

Da es dieses Element nur einmal geben darf, kann die Suche durch das DOM abgebrochen werden, wenn das Element gefunden wurde.

2. Class Rule: *.link-red*

Das DOM muss einmal komplett nach allen Klassen durchsucht werden.

3. Tag Rules: *a; li; p;*

Verhalten sich wie Klassen, nur sind diese vordefiniert und haben einen Sinn im Markup.

David Hyatt hat es in seinem viel zitierten Blog Post als Best Practise ausgerufen, sich wenn möglich auf diese drei Arten von Selektoren zu beschränken um möglichst effiziente CSS-Selektoren zu schreiben (Hyatt David, 2000).

3.4.1 CSS-Sprites

Es macht keinen Unterschied ob ein Bild im HTML- oder CSS-Code referenziert ist, es wird für beide Varianten ein HTTP-Request gestellt und die Datei heruntergeladen. Bilder im CSS sind häufig aber nur kleine grafische Accessoires, wie Icons oder Buttons. Ihre Größe übersteigt einige Kilobyte meist nicht, weshalb der Round-Trip für einen HTTP-Request in diesem Fall sehr teuer ist.

Eine Lösung für dieses Problem stammt aus der „old school“ Computerspieleentwicklung, damals hatten die Programmierer mit Performanceproblemen zu kämpfen. Viele kleine Grafiken mussten adressiert und dann in den Speicher geladen werden. Um die Lese-Schreibzugriffe zu reduzieren, wurden dann viele kleine Grafiken zu einer großen zusammengefasst und nur die relevanten Bereiche des großen Bildes angezeigt.

CSS-Sprites funktionieren ganz ähnlich, mehrere Bilder werden zu einer Datei zusammengefasst. Und dann über das „Background-Position“-Attribut im CSS an die richtige Stelle positioniert.

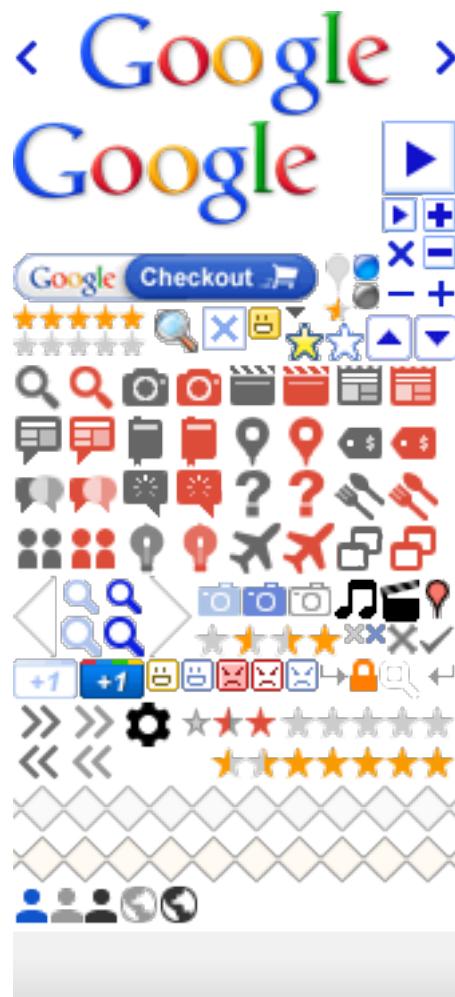


Abbildung 7: CSS-Sprite der Google Webseite

Wie in Abbildung 7 zu erkennen ist, macht auch Google von dieser Technik Gebrauch. Neben dem Logo sind diverse Buttons und Icons zu erkennen, die möglichst Platz sparend in dem Bild platziert wurden.

3.5 Netzwerk/Übertragungsweg

Auf dem Weg vom Server zum Client gibt es einige Stolpersteine, die ein Web-Entwickler kennen sollte, damit er das volle Potenzial aus der Datenübertragung ausschöpfen kann.

3.5.1 Komprimierung der Datenübertragung

Seit der Version 1.1 des HTTP-Protokolls ist es möglich, dass der Server die Daten welche vom Client angefordert werden, komprimiert ausliefert. Dadurch verringert sich die Menge der zu übertragenen Daten, was sich wiederum positiv auf die Seitenladezeit auswirkt.

Eine effiziente Komprimierung funktioniert aber nur für Dateien die nicht bereits in einem komprimierten Format vorliegen. Bilder, Flash oder PDF-Dateien sollten nicht noch zusätzlich komprimiert werden. Eine erneute Komprimierung würde die Dateien nicht verkleinern und somit kaum Zeit bei der Übertragung einsparen. Im Gegenteil, da die Komprimierung auf dem Server und die Dekomprimierung beim Client auch eine gewisse Zeit dauern, könnte sich das sogar negativ auf die Performance auswirken.

Dahingegen lassen sich HTML-, CSS- und JavaScript-Dateien sehr gut komprimieren, da sie reine Textdateien sind. Durchschnittlich können durch die Aktivierung der Komprimierung 70% des Datenvolumens gespart werden (Souders, 2007, S. 31).

Aktuelle Browser sind so konfiguriert, dass sie dem Server in ihrer Standardkonfiguration bereits mitteilen, dass er die Daten komprimiert zurück schicken soll. Dafür tragen sie in den HTTP-Request den Header *Accept-Encoding: gzip, deflate* ein. Der Server weiß dadurch, dass der Client mit den beiden Komprimierungsverfahren *gzip* und *deflate* umgehen kann.

In der Standardkonfiguration vieler Server wird daraufhin die HTML-Datei mit *gzip*-Kompression übertragen. Genauso sinnvoll obwohl oft nicht verwendet, ist es, die CSS- und JavaScript-Dateien komprimiert zu übertragen, dafür muss die Serverkonfiguration in der Regel nur auf die Komprimierung der beiden anderen Dateitypen ausgeweitet werden.

3.5.2 HTTP keep-alive

HTTP ist ein zustandsloses Datenübertragungsprotokoll und gehört der Anwendungsschicht etablierter Netzwerkmodelle²⁶ an. Darunter auf der Transportschicht wird für den Internetverkehr das TCP-Protokoll eingesetzt, dieses Protokoll baut anhand der IP-Adresse eine Verbindung zum Server auf. Über diese Verbindung werden dann die Datenpakete ausgetauscht. Um diese Verbindung aber nun nicht für jeden HTTP-Request erneut aufzubauen zu müssen, gibt es den HTTP-Header *Connection: Keep-Alive*, dieser zeigt dem Server, dass der anfragende Client die Verbindung gern bestehen lassen möchte, um darüber weitere Requests zu senden. Der Server entscheidet nun je nach Konfiguration, ob er mit *Connection: Keep-Alive* also positiv oder mit *Connection: close*, dem negativen Pendant, darauf antwortet.

Der Apache, einer der prominentesten Web-Server, antwortet in seiner Standardkonfiguration mit *Keep-Alive* und hält dann eine Verbindung 15s lang offen, bevor er sie wieder schließt.

3.5.3 Expires Header und der ETag

Aktuelle Webseiten werden immer aufwändiger und umfangreicher, das bedeutet für den Besucher bzw. seinen Browser, dass er beim ersten Aufruf einer Webseite zunächst eine Menge Bild-, JavaScript- oder CSS-Dateien herunterladen muss. Da diese Dateien sich im Leben einer Webseite aber nicht so häufig ändern, gibt es den Browser-Cache, indem diese statischen Dateien vorgehalten werden können, um sie später nicht erneut vom Server laden zu müssen.

Um Dateien in diesem Cache ablegen zu können, gibt es zwei Möglichkeiten. Die erste ist die Verwendung des **ETags**. Abbildung 8 visualisiert diesen Vorgang. Der Server bekommt eine Initialanfrage von einem Client (First View) und antwortet mit der angeforderten Datei. Zusätzlich wird in der Antwort der ETag im HTTP-Header gesetzt. Dieser Eintrag ist ein Hashwert²⁷, oft zusammengesetzt aus dem Dateinamen und einem Zeitstempel, und identifiziert die Datei eindeutig. Der Client speichert nun diese Datei in seinem Cache und verwendet sie.

Später, wenn der Benutzer die gleiche Datei noch einmal benötigt, also eine Repeat View erzeugt, stellt der Client erneut die Anfrage nach der Datei. Dieses Mal, weiß der Browser allerdings, dass er die Datei bereits im Cache hat und sendet den Inhalt des ETags im *if-none-match*-Tag des HTTP-Requests mit. Der Server vergleicht nun anhand des Hashwertes seine aktuelle Version mit der des Clients und kann somit feststellen, ob es eine neuere Version der Datei gibt. Wenn dem nicht so ist, gibt er den Status-

²⁶ oder auch Schichtenmodelle genannt, mehr dazu hier: <http://de.wikipedia.org/wiki/Schichtenmodell>

²⁷ Mit einer Hashfunktion wird aus einer unbestimmt großen Quellmenge an Daten eine kleine Ziellmenge (Hashwert) mit definierter Größe erzeugt.

Code „304 Not Modified“ zurück und überträgt die Daten nicht erneut. Ansonsten würde er die Datei mit dem neuen Hashwert im ETag übertragen.

Der Vorteil ist klar, es müssen weniger Daten übertragen werden und der Seitenaufbau geht schneller. Dennoch müssen zeitaufwändige HTTP-Requests für jede Datei gestellt werden.

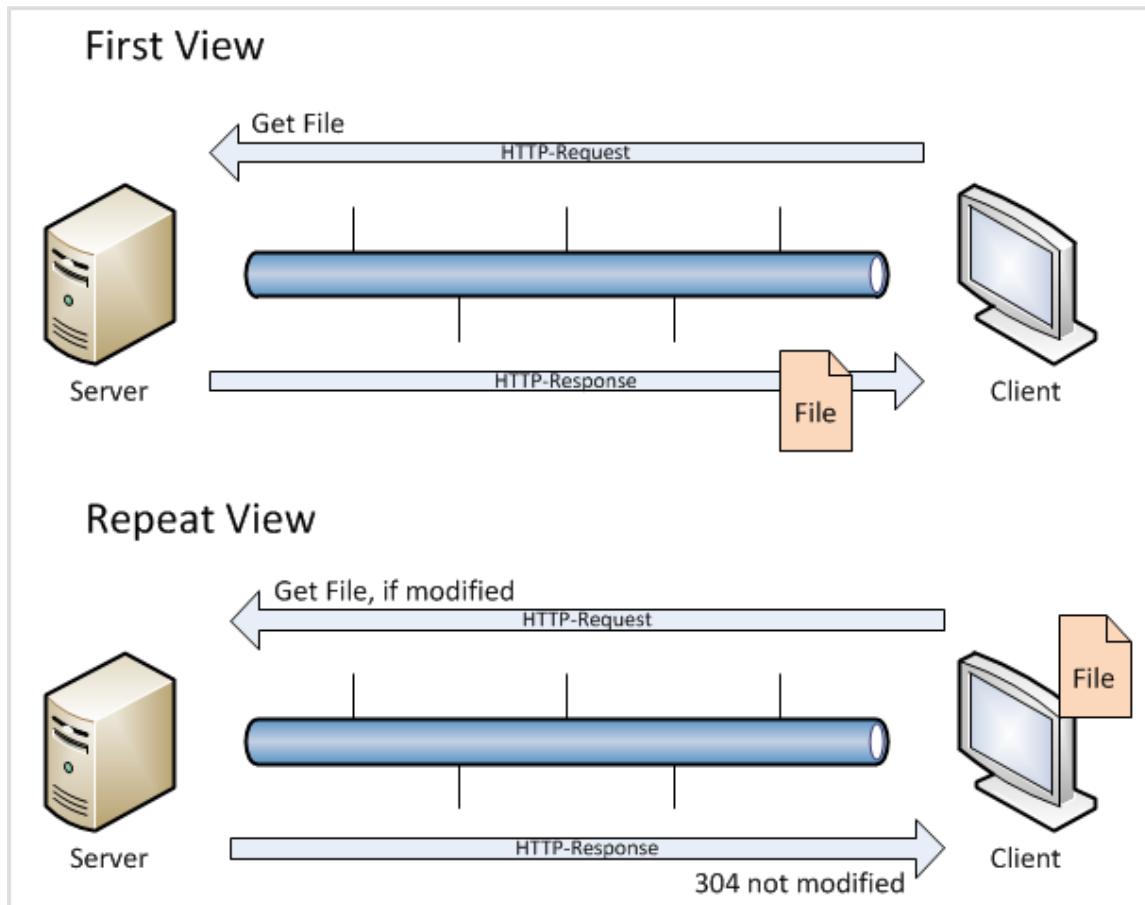


Abbildung 8: Cacheabfrage mit dem ETag

Die zweite Variante der **Expires Header** geht dieser Problematik aus dem Weg. Hier bestimmt nämlich der Server ein Verfallsdatum der statischen Dateien. Der Browser muss Dateien, die mit einem Expires-Header vom Server ausgeliefert wurden, solange nicht neu laden, bis das Verfallsdatum abgelaufen ist. Dadurch werden die „teuren“ HTTP-Requests auch noch eingespart. Im Idealfall muss bei einem Page Reload nur noch die HTML-Datei neu geladen werden. Abbildung 9 zeigt den Ablauf der First und Repeat View bei Einsatz des Expires Header, auffällig ist, dass bei der Repeat View gar kein HTTP-Request mehr gesendet werden muss.

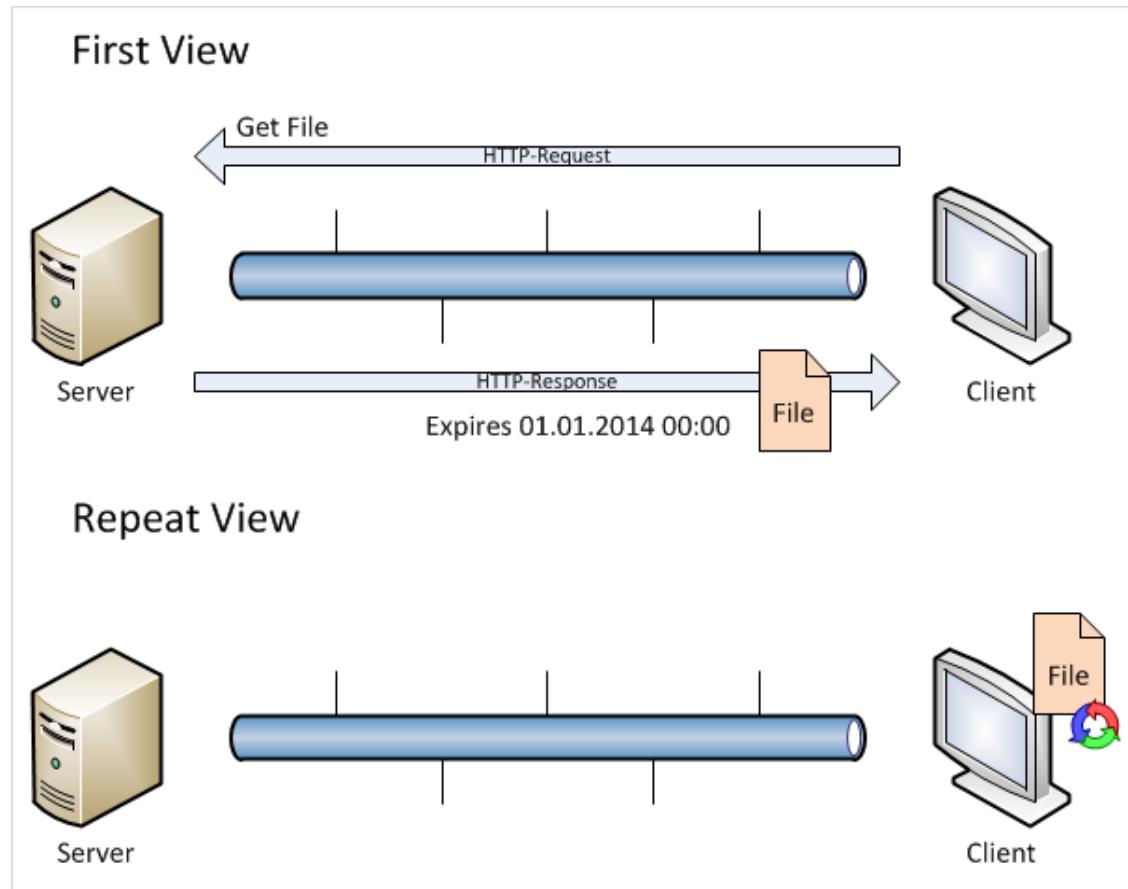


Abbildung 9: Cacheabfrage mit dem Expires Header

Ein Nachteil dieser Technologie ist, dass je nachdem wie weit das Verfallsdatum in der Zukunft gewählt wurde und wie lange diese Datei beim User im Browser-Cache erhalten bleibt, Änderungen schlecht verteilt werden können. Wenn also beispielsweise an einer CSS-Datei etwas geändert wurde, der Client aber noch auf eine alte CSS-Datei zurückgreift weil diese sich noch in seinem Cache befindet, dann wird er von den Änderungen nichts mitbekommen.

Deshalb werden statische Dateien häufig mit einer Versionsnummer versehen, um dem Browser bei einer Änderung vorzugaukeln, dass er eine völlig neue Datei herunterzuladen hat.

Es ist empfehlenswert seine Caching-Strategie mit dem Expires Header aufzubauen, denn dieser bietet den höchsten Performancegewinn und eine hohe Flexibilität, was Änderungen angeht. Allerdings sollte man immer im Hinterkopf behalten, dass sich das Caching erst bezahlt macht, nachdem die Seite einmal vollständig geladen wurde. Der erste Aufruf einer Seite kann sich aus Performancesicht also erheblich von allen weiteren Interaktionen mit der Seite unterscheiden.

Eine Studie von Tenni Theurer, Entwickler bei Yahoo!, hat ergeben, dass 40-60% aller Besucher der Webseite www.yahoo.com eine *empty-cache-experience*²⁸ machen und insgesamt 20% aller Seitenaufrufe ohne gefülltem Cache stattfinden (Theurer, 2007, Part 2). Das zeigt wie wichtig die Betrachtung des Seitenladevorgangs auch ohne den Browser-Cache ist. Der Cache sollte nach Meinung des Autors als tolles Feature angesehen werden aber niemals als Rechtfertigung für einen langsamen Seitenaufbau mit leerem Cache gelten.

3.5.4 Content Delivery Network

Ein Content Delivery Network (CDN) ist weniger ein Tweak als eine komplett eigene Strategie der Auslieferung von Daten einer Webseite an den Benutzer. Das Problem des klassischen Client-Server-Prinzips ist die Standortabhängigkeit und die Tatsache, dass ein einziger Server für alle Anfragen zuständig ist.

Durch die Ortsgebundenheit eines einzigen Servers variiert die Round-Trip-Time, je nachdem von wo aus die Seite abgerufen wird, sehr stark. Geht man innerhalb Deutschlands von einer RTT von 50ms aus und ruft eine deutsche Webseite auf, die aus 60 Komponenten besteht, so dauert der Verbindungsauflauf für die einzelnen Komponenten 3 Sekunden. Wenn nun ein Besucher aus den USA die Seite aufruft, dauert der Verbindungsauflauf für alle Komponenten ganze 7,5 Sekunden. Das liegt an der Zeit, welche die Datenpakete für die Strecke über den Atlantischen Ozean brauchen. Für die Berechnung wurde eine RTT von 125ms angenommen. Um nun aber allen Besuchern, egal von wo aus sie die Seite abrufen eine gute Performance zu bieten, ist ein CDN die Lösung.

Wie der Name bereits verrät, werden die Inhalte der Webseite auf ein Netzwerk von vielen einzelnen Servern auf der ganzen Welt, bzw. in dem Gebiet, wo die Seite hochverfügbar sein soll, verteilt. Dabei gibt es einen Haupt-Server und viele Cache-Server. Die Anfragen der Besucher werden immer an den am schnellsten erreichbaren Cache-Server geleitet. Dieser liefert dann die gewünschte Datei aus, wenn er sie im Cache hat. Hat er sie nicht im Cache wird die Anfrage an den Haupt-Server weitergeleitet und dann an den Besucher ausgeliefert. Für die nächste Anfrage dieser Datei ist der Cache-Server dann gewappnet und muss keine Anfrage mehr an den Haupt-Server stellen.

²⁸ heißt im Deutschen so viel wie „Leere-Cache-Erfahrung“

Es gibt Firmen, die sich ein solches CDN selbst aufbauen und warten. Der gewöhnliche und auch finanziell günstigere Weg ist allerdings einen CDN Service Provider zu nutzen. Die populärsten ihrer Art sind Akamai²⁹, Limelight Networks³⁰, EdgeCast³¹ und Amazon Web Services³².

3.5.5 Cookies

HTTP-Cookies haben viele Anwendungsgebiete, wie Personalisierung oder Authentifizierung. Die dafür nötigen Daten werden im HTTP-Header gespeichert und so zwischen Web-Server und Browser ausgetauscht.

Auf Abbildung 13 ist der Inhalt des Cookie-Attributes einer HTTP-Anfrage ersichtlich. In diesem Fall werden einige Checksummen und die PHP-Sessionid an den Server übermittelt.

Aus Performancesicht ist es wichtig die Größe der Cookies möglichst gering zu halten, da diese mit jedem Request übertragen werden und dadurch die Antwortzeit erhöhen. Einer Studie zufolge verzögert ein Cookie in der Größe von 3KB die Antwortzeit um 80ms bei DSL-Geschwindigkeit (Theurer, 2007, Part 3).

Cookie freie Domains für Komponenten

Beim Abrufen von statischen Komponenten wie Bilder, JavaScript- oder CSS-Dateien gibt es keinen Grund, warum Cookies dafür mitgesendet werden sollten. Stattdessen sollten statische Komponenten von Domains bezogen werden, auf denen keine Cookies gesetzt wurden. Das verkleinert die Größe jedes HTTP-Requests und beschleunigt somit die Antwortzeit.

²⁹ <http://www.akamai.com/>

³⁰ <http://www.limelightnetworks.com/>

³¹ <http://www.edgecast.com/>

³² <http://aws.amazon.com/>

4 Optimieren von Fein.de

Um die in den vorherigen Kapiteln theoretisch aufgearbeiteten Maßnahmen zur Steigerung der Performance in der Praxis anzuwenden, wurde im Rahmen dieser Arbeit die Webseite vom Elektrowerkzeuge-Hersteller Fein (<http://www.fein.de>) analysiert.

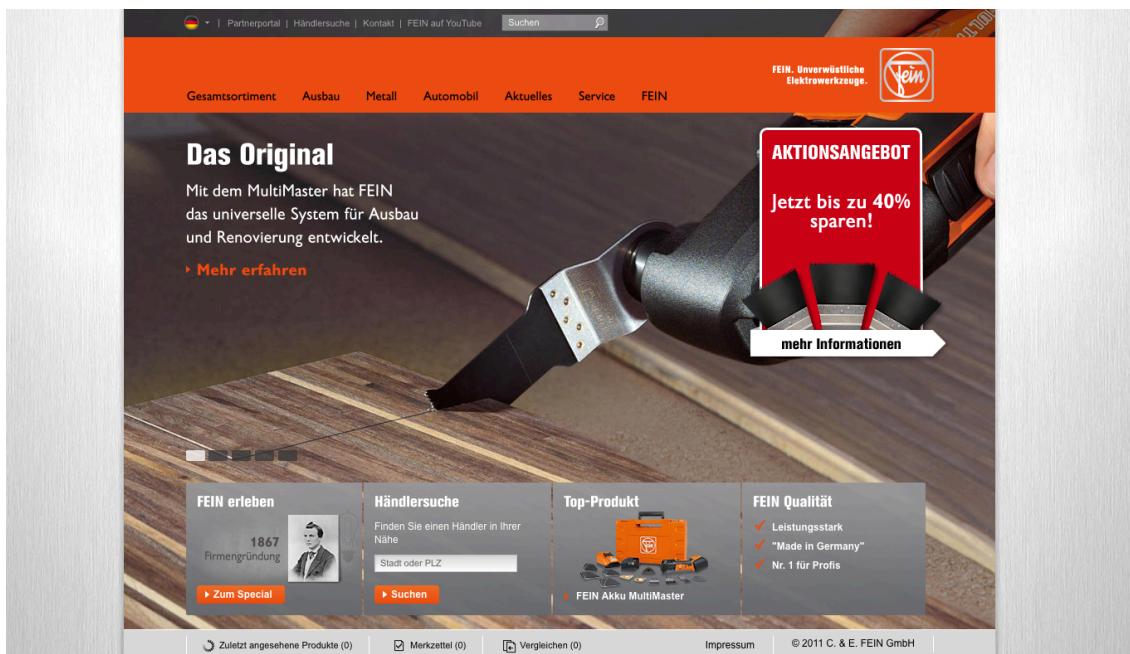


Abbildung 10: Startseite von Fein.de

Bei der Optimierung einer, wie in diesem Fall, bereits fertiggestellten Webseite, sollten nicht wahllos irgendwelche Maßnahmen umgesetzt werden. Nur eine umfassende Analyse der Performance-Schwachstellen führt zu einer effektiven Optimierung. Ein Zufallstreffer ist bei der Komplexität heutiger Webseiten eher unwahrscheinlich.

,,don't fiddle – analyse first“

Jakob Schröter

4.1 Auswertung

Dieses Kapitel schafft eine Faktengrundlage für eine objektive Analyse der Performance-Schwachstellen.

Die Fein-Webseite wird vom Internet-Service-Provider LF.net, vom Standort Stuttgart aus, bereitgestellt. Dafür steht ein eigener Server mit einem Linux-Betriebssystem zur Verfügung. Die Webseite wird von einem Apache-Web-Server³³ verwaltet und als Datenbank kommt MySQL³⁴ zum Einsatz. Für eine performante Suchfunktion werden bestimmte Daten von dem Solr-Indexer³⁵ ausgeliefert.

Mit Hilfe des Analyse-Tools econda³⁶ wurden Klickstatistiken aufgezeichnet. Das funktioniert auf die gleiche Weise wie Google Analytics³⁷, indem ein kleiner JavaScript-Code in die Seite eingebunden und dadurch von jedem Besucher beim Aufruf der Seite ausgeführt wird. Die dadurch übertragenen Daten werden von econda gespeichert und können über eine Weboberfläche, gut visualisiert, betrachtet werden.

Im Januar 2012 wurde die Webseite von 128.478 eindeutigen Besuchern aufgerufen, diese haben sich 569.386 Unterseiten von www.fein.de angeschaut. Im Schnitt ist ein Besucher 6,01 Minuten auf der Präsenz und besucht dabei 4,42 Unterseiten.

Die Seite ist im Moment in über 50 Sprachen verfügbar und wird von überall auf der Welt aufgerufen (s. Abbildung 11 „Zugriffe pro Land – Januar 2012 – Fein.de“).

³³ <http://httpd.apache.org/>

³⁴ <http://www.mysql.de/>

³⁵ <http://lucene.apache.org/solr/>

³⁶ <http://www.econda.de/>

³⁷ <http://www.google.com/analytics/>

4.1.1 Klickpfade

Unter anderem werden von econda auch Klickpfade aufgezeichnet, diese zeigen wie Benutzer durch die Seite navigieren und an welcher Stelle sie diese wieder verlassen.

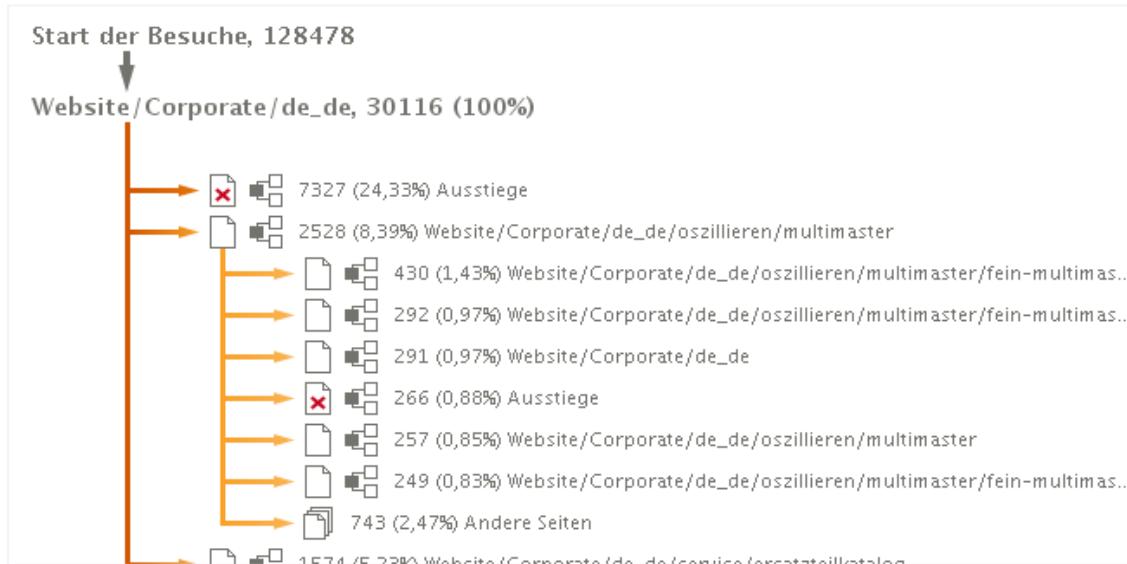


Abbildung 11: econda-Ausschnitt der User-Klickpfade von Fein.de

In der Abbildung 12 ist zu sehen, dass zwischen dem 01.01.2012 und 31.01.2012 insgesamt 128.478 Besucher auf die Seite gegangen sind, davon haben 30.116 die deutschsprachige Seite besucht. 24,33% davon verlassen die Seite direkt wieder und alle anderen verteilen sich recht gleichmäßig auf dem Angebot von Fein. Die Absprungraten der Unterseiten weisen auf kein ungewöhnliches Verhalten der Nutzer hin. Interessant ist allerdings die Frage: Aus welchem Grund ist die Anzahl der Benutzer welche die Seite direkt wieder verlassen, so hoch? Dies kann anhand dieser Statistik nicht genau bestimmt werden. Eventuell haben sie bereits gefunden was sie gesucht haben oder aber sie sind mit etwas unzufrieden. Auf jeden Fall ist es ein Anhaltspunkt für weitere Nachforschungen.

4.1.2 Geschwindigkeit messen

Um später Vergleichswerte dafür zu haben, wie viel die Optimierungen an der Seite gebracht haben, wird die Testumgebung (<http://willi.fein.stuttgart.weitclick.de>) von Fein.de einem Geschwindigkeitstest unterzogen. Dafür wird die Zeit vom ersten Request an den Server bis zum Ende des Render-Prozesses (Load Time) gemessen. Es kommt das Tool webpagetest.org³⁸ zum Einsatz, damit ist es möglich, einige Parameter wie z.B. Standort, Verbindungs geschwindigkeit und verwendeter Browser, zu konfigu-

³⁸ Erläuterung zu diesem Tool im Anhang A „Tools zum Messen ...“

rieren. Dies ist nötig, da die Fein-Webseite zu einem großen Teil auch aus dem Ausland abgerufen wird, die Server aber in Deutschland stehen.

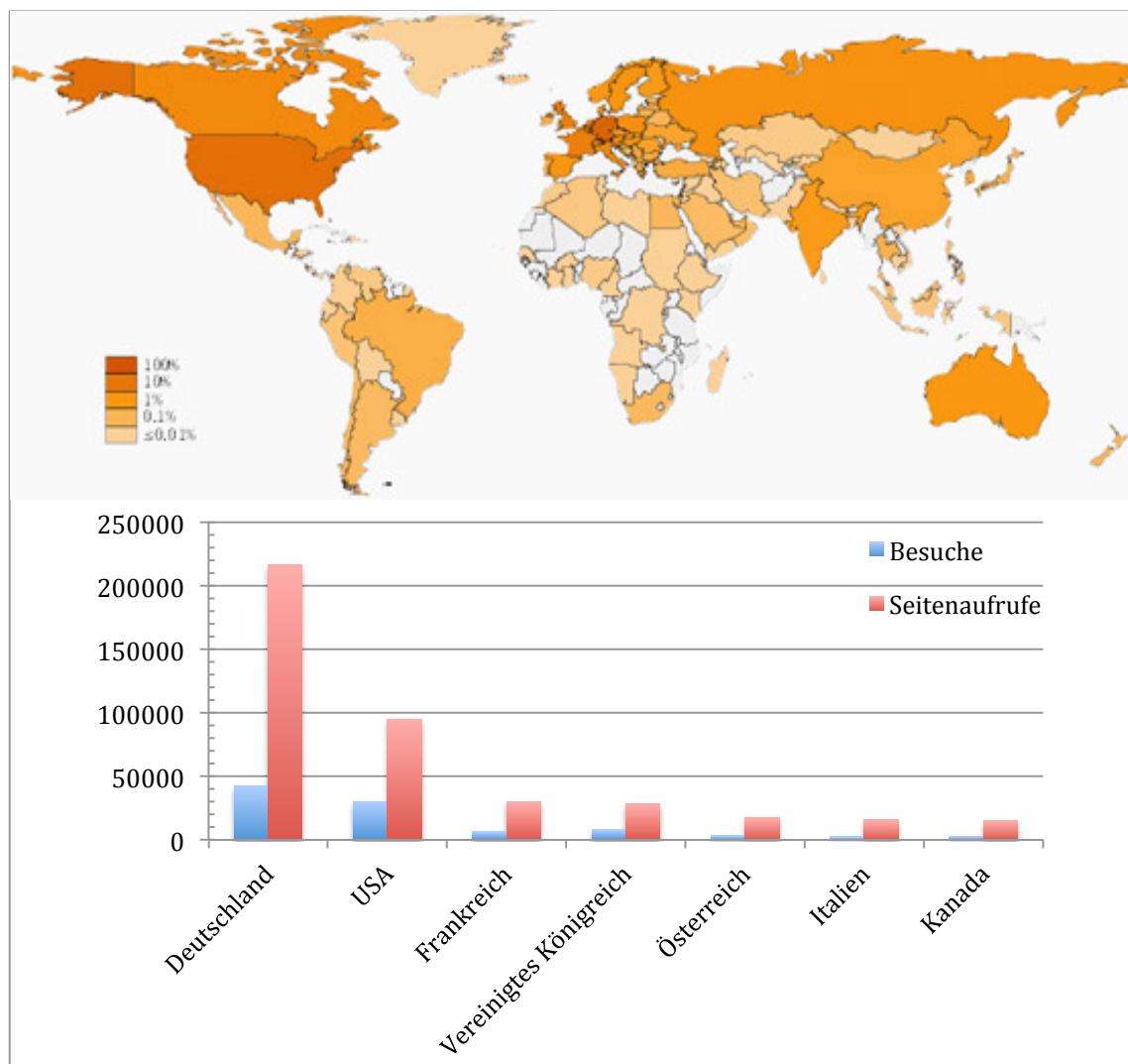


Abbildung 12: Zugriffe pro Land – Januar 2012 – Fein.de

Wie aus der econda-Auswertung in der Abbildung 13 ersichtlich, kommen die mit Abstand meisten Zugriffe aus Deutschland und den USA. Aus diesem Grund konzentriert sich die Geschwindigkeitsmessung auch auf die beiden Standorte Frankfurt (DE) und Los Angeles (US).

Ein weiterer wichtiger Faktor für die Messung ist der verwendete Browser, hier kann wiederum econda Aufschluss darüber geben, welche Browser am häufigsten verwendet werden.

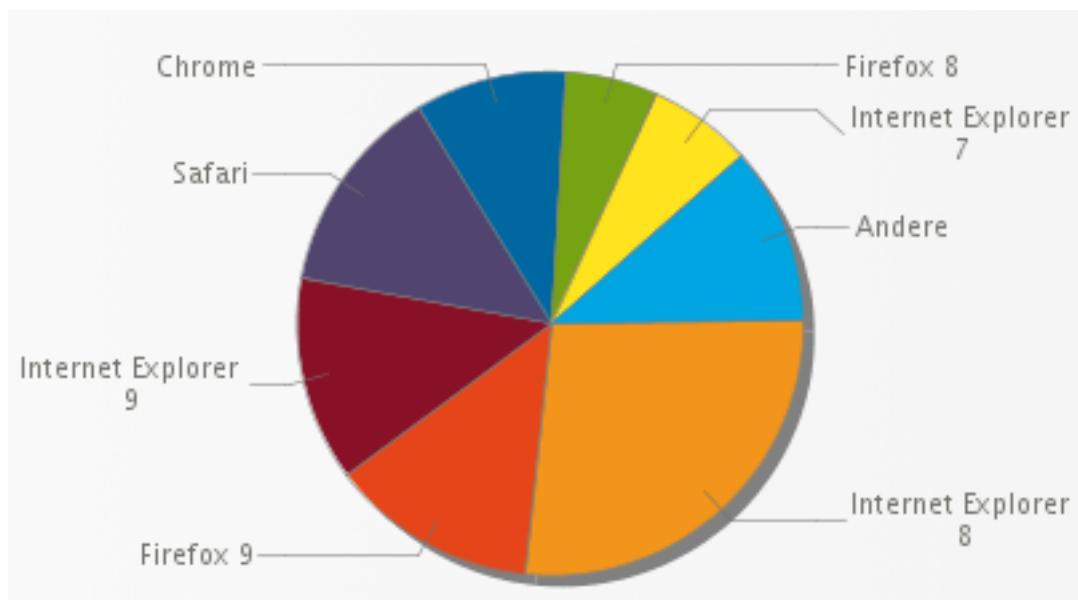


Abbildung 13: Besucher pro Browser - Januar 2012 - Fein.de

Der Internet Explorer ist demnach mit fast der Hälfte aller Zugriffe der am meisten genutzte Browser auf Fein.de. Für den Test wurde er daher in der am stärksten vertretenen Version 8 (27%) verwendet. Zusätzlich wurden auch noch Tests mit dem Internet Explorer 7 gemacht da dieser ebenfalls einen signifikanten Benutzeranteil (7%) hat und zudem eine technische Ausnahme darstellt. Er ist der einzige Browser dessen maximale Anzahl von Verbindungen zu einem Host auf zwei beschränkt ist.

Um möglichst realistische Bedingungen für den Test zu schaffen, wurde für die Bandbreite die durchschnittliche Internetverbindungsgeschwindigkeit des jeweiligen Landes verwendet. Dies ist für Deutschland eine Geschwindigkeit von 5,3Mbit pro Sekunde (Mbps) und für die USA 5,8Mbps (Akamai, 2011). Außerdem wurde die Round-Trip-Time (RTT)³⁹ auf typische Werte für die Entfernung gesetzt, innerhalb Deutschlands 50ms und von den USA nach Deutschland 125ms (Wikipedia - Paketumlaufzeit, 2012).

³⁹ Die Round-Trip-Time gibt die Zeit an die ein Datenpaket in einem Rechnernetz benötigt, um von der Quelle zum Ziel und zurück zu gelangen.

Tabelle 5: Parameter für den Performancetest mit webpagetest.org

Parameter	Wert
Bandbreite	DE 5.3Mbps / 5427Kbps
	US 5.8Mbps / 5939Kbps
Round Trip Time (RTT)	DE 50ms US 125ms
Standorte	Frankfurt, DE
	Los Angeles, US
Browser	Internet Explorer 8
	Internet Explorer 7

Für jede der vier Parameterkonstellationen wird die Zeit einmal ohne (First View) und einmal mit gefülltem Cache (Repeat View) gemessen. Jede dieser acht Stichproben wird fünfmal erhoben aber jeweils nur das Ergebnis als relevant betrachtet, welches sich im Vergleich in der Mitte ausrichtet. Damit sind evtl. Seiteneffekte durch erhöhte Paketverluste oder schlechtes Routing der Pakete weitgehend ausgeschlossen.

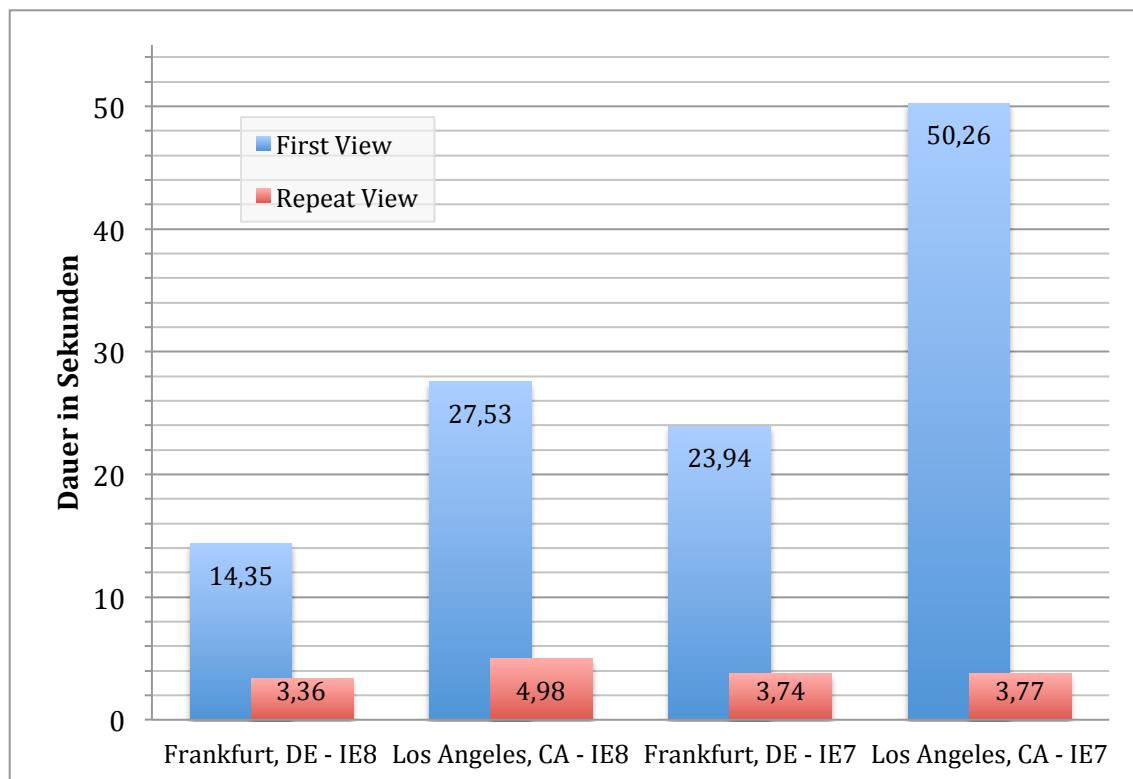


Abbildung 14: Performancetest von Fein.de (Testumgebung) - Urzustand

Bei Betrachtung des Ergebnisses (Abbildung 15) können gleich mehrere interessante Beobachtungen gemacht werden. Der Internet Explorer 8 ist im Verhältnis zu Version 7 deutlich schneller. Das macht sich besonders für die First View bemerkbar. Für den Aufruf der Seite aus Frankfurt braucht der IE7 knapp 6s länger als der 8er und für Los Angeles sind es sogar 14s. Grundsätzlich fällt auf, dass der Initialaufruf der Seite sehr lange dauert, im Idealfall knapp über 14s (IE8, Frankfurt) und im Schlechtesten über 50s (IE7, Los Angeles).

Die Repeat View ist in allen Tests signifikant schneller als die First View, sie liegt relativ konstant bei einer Ladedauer von 3s bis 4s. Daraus lässt sich schließen, dass das Browser Caching gut funktioniert. Und das Hauptaugenmerk der Analyse der First View gelten sollte.

Zu den Messbedingungen sei gesagt, dass diese Werte mit einer durchschnittlich schnellen Internetverbindung ermittelt wurden und eine Tageszeitabhängigkeit bei der Erhebung der Proben spürbar war. Deshalb wurde sich bemüht, die Stichproben immer in den Abendstunden zu nehmen um möglichst gleich bleibende Bedingungen zu schaffen. Grundsätzlich dienen die Messungen dazu, eine Tendenz und ungefähre Auswirkung festzustellen, dafür ist die in diesem Szenario gewählte Genauigkeit ausreichend.

4.1.3 Zieldefinition

Als Ziel dieser Performanceoptimierung wurde forciert, dass die Seite innerhalb des definierten Testszenarios nicht mehr länger als 15s braucht, um vollständig geladen zu sein. Das bedeutet automatisch, dass sich die Performance des Internet Explorer 7 deutlich verbessern muss, da dieser das größte Problem darstellt.

Die Definition eines Ziels bestimmt die Mittel mit denen eine Optimierung durchgeführt wird und gibt eine Richtung für die Analyse vor. Je ehrgeiziger das Ziel definiert wird, desto aufwändiger/teurer wird die Umsetzung.

4.1.4 Viele HTTP-Requests

Außer der Ladezeit wertet webpagetest.org auch noch den Ladevorgang einer Webseite aus. Alle Komponenten aus diesem Prozess werden in einem Wasserfallmodell dargestellt. Auf der Y-Achse sind alle Komponenten (Bilder, JS, CSS, etc.), die für diese Seite geladen wurden, zu sehen, jede Komponente steht für einen HTTP-Request. Die X-Achse visualisiert den zeitlichen Verlauf insgesamt und durch die Länge der Balken für jede Komponente einzeln. Die vertikale grüne Linie markiert den Zeitpunkt, an dem der Browser mit dem rendern der Seite beginnt und die blaue wann er damit fertig ist. Letzterer ist der Moment, an dem die Messung endet, da ab hier die Webseite vollständig angezeigt wird und bedienbar ist.

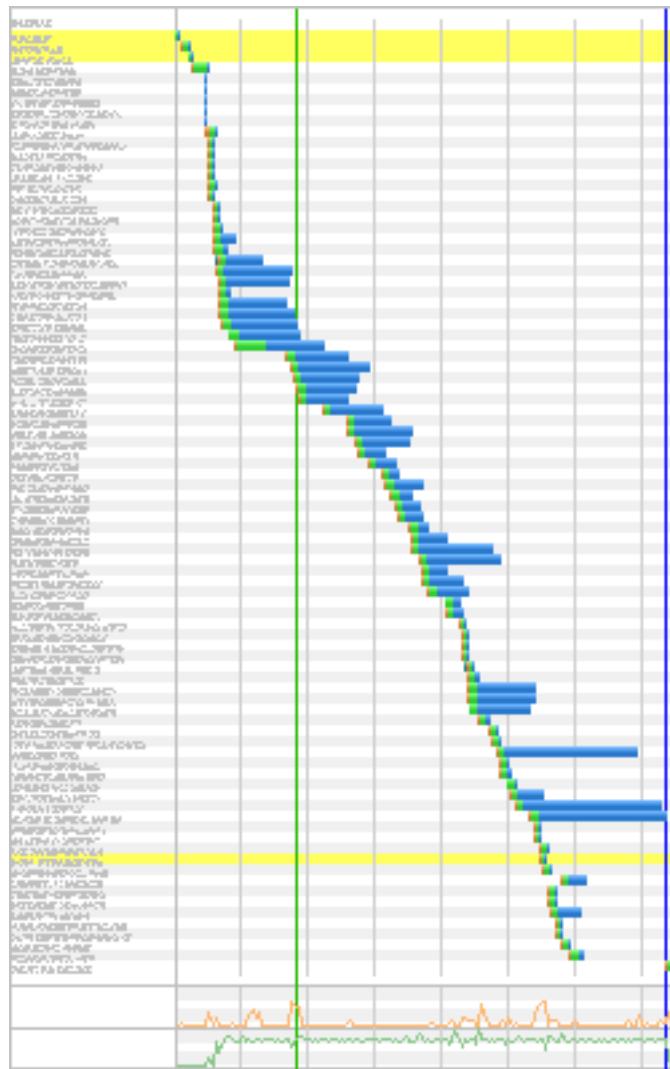


Abbildung 15: Wasserfall-Modell von Fein.de vor der Optimierungsmaßnahme

Abbildung 16 zeigt die vollständige Übersicht von einem Wasserfall-Modell, diese sieht für alle Messungen aus diesem Test ähnlich aus und kann natürlich auch detaillierter betrachtet werden. Unter dem Diagramm sind noch zwei Linien-Diagramme zu sehen, das obere stellt die Auslastung der CPU des Client-Computers dar und die untere die Auslastung der Bandbreite.

Insgesamt werden 82 HTTP-Requests für den Initialaufruf der Startseite von Fein.de an den Server übermittelt. Dieser Wert ist nicht zwingend schlecht, aber multipliziert man die RTT von den USA nach Deutschland von 125ms (in diesem Testszenario) mit der Anzahl der Requests, so werden über 10s allein dafür verwendet, die Anfragen an den Server zu senden. An dieser Stelle lässt sich also viel Potenzial vermuten.

4.1.5 HTTP Keep-Alive nicht verwendet

Bei näherer Betrachtung des Wasserfall-Modells fällt auf, dass immer wieder eine neue Verbindung zum Server aufgebaut wird. Zu erkennen an dem orangen, Teil der Balken.

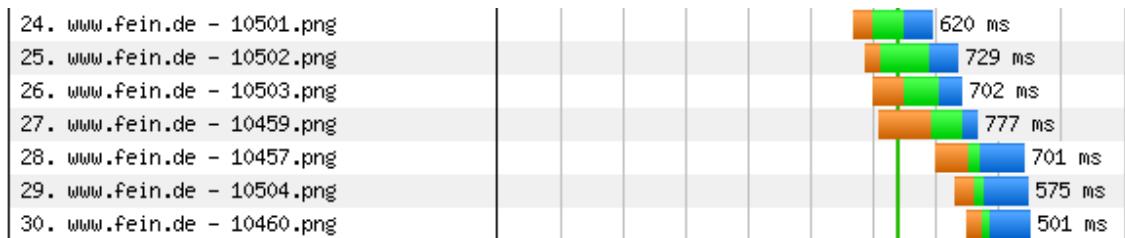


Abbildung 16: Wasserfall-Modell-Ausschnitt mit inaktivem HTTP Keep-Alive

Ein Blick in den Request-Header den der Browser für ein Bild verwendet, zeigt, dass das Problem hier nicht zu finden ist, denn die Anfrage wird mit „Connection: keep-alive“ gesendet.

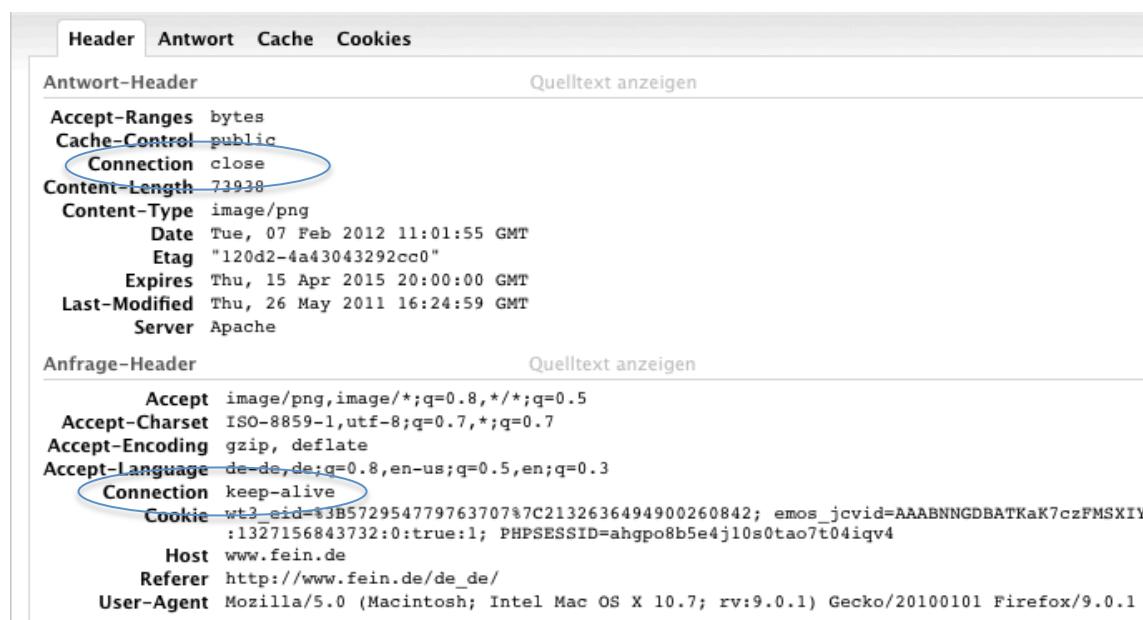


Abbildung 17: Anfrage- und Antwort-Header eines HTTP-Requests (Keep-Alive Off)

Die Betrachtung des Response-Headers zeigt allerdings, dass die Verbindung nach einem Request vom Server immer wieder geschlossen wird. Besser wäre es, die Verbindung vom Server offen halten zu lassen und erst am Ende wieder zu schließen.

4.1.6 Unnötiger Redirect

Gibt der Benutzer in den Browser die URL <http://fein.de> ein, so finden drei Redirects statt, bis der Browser die endgültige URL weiß. Der Server quittiert jede Anfrage mit dem HTTP-Statuscode 301 „Moved Permanently“ und gibt damit zum Ausdruck, dass es für die angeforderte URL eine andere gibt, die verwendet werden soll.

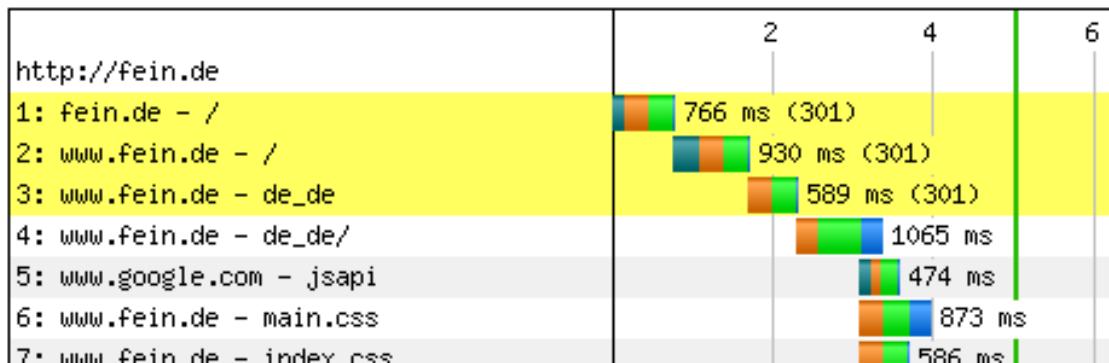


Abbildung 18: Wasserfall-Modell - Redirect Fehler

Der erste Redirect von Fein.de auf www.fein.de ist nicht zu vermeiden. Würden beide URLs funktionieren, hätte das eine Herabstufung der Vertrauenswürdigkeit, mit der Begründung des mehrfach vorhandenen Inhaltes (duplicate content), bei den Suchmaschinen zur Folge. Schließlich würden folgende URLs den gleichen Inhalt ausliefern:

- http://www.fein.de/de_de/fein/karriere/fein-als-arbeitgeber-t61/
- http://fein.de/de_de/fein/karriere/fein-als-arbeitgeber-t61/

Der zweite Redirect verweist den Browser auf die Ebene „de_de“. Da hier aber offensichtlich in der Konfiguration vergessen wurde, den schließenden Slash zu setzen, greift das Standardverhalten des Webservers. Er interpretiert alle URLs ohne den schließenden Backslash als Dateien. Im Root-Verzeichnis kann er aber keine Datei „de_de“ finden, also nimmt er an, dass nach dem Verzeichnis verlangt wird und verweist auf „de_de/“. Von dieser URL bekommt der Browser nun schließlich die angeforderte HTML-Datei. Dieser Redirect (Zeile 3) ist unnötig und sollte deshalb vermieden werden.

4.1.7 Großteil des Traffics von einer Domain

In der Abbildung 20 ist ein Kuchendiagramm zu sehen, welches alle Domains und die davon bezogene Datenmenge beim Aufruf von Fein.de visualisiert.

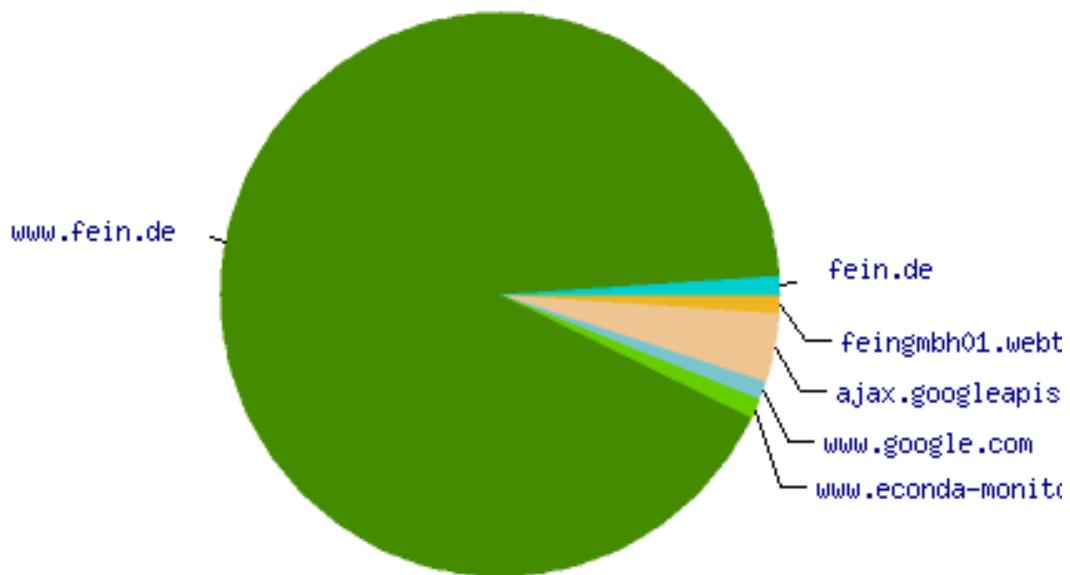


Abbildung 19: Datenmenge pro Domain im Verhältnis zum Gesamtvolumen (vor der Optimierung)

Auffällig ist die Menge an bezogenen Daten von der Domain www.fein.de. Dieses Ungleichgewicht sorgt für mehrere Seiteneffekte.

Parallele Downloads

Theoretisch wirkt sich diese Tatsache am stärksten auf die Performance im Internet Explorer 7 aus. Wie im Kapitel 5.1.3 „DNS-Lookups minimieren und HTTP-Requests parallelisieren“ angesprochen, baut dieser Browser maximal zwei Verbindungen pro Host-Server auf. Die Folge ist, im Vergleich mit aktuelleren Browsern, eine höhere Laufzeit, ein Blick auf das Wasserfall-Modell bestätigt das.

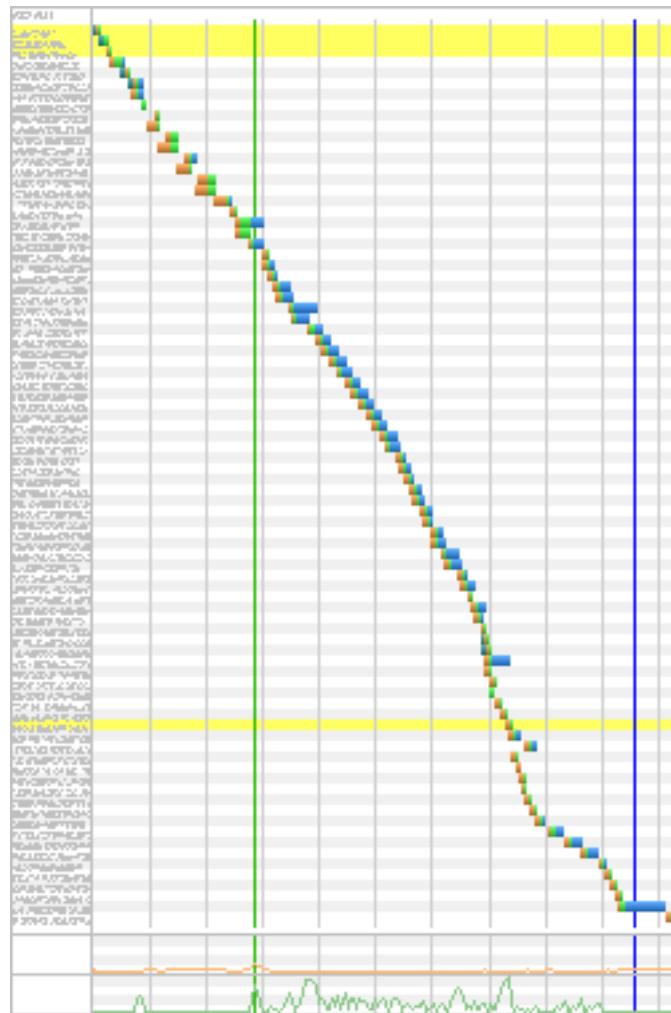


Abbildung 20: Wasserfallmodell von Fein.de (Internet Explorer 7)

Verglichen mit dem Wasserfall-Modell des Internet Explorer 8 (Abbildung 16) ist deutlich zu sehen, dass maximal zwei Komponenten gleichzeitig heruntergeladen werden und sich die Gesamtladezeit dadurch deutlich verlängert.

DNS-Lookups

Da jeder DNS-Lookup eine gewisse Zeit dauert (siehe dazu Kapitel 5.1.3 „DNS-Lookups minimieren und HTTP-Requests parallelisieren“), sollten diese sparsam eingesetzt werden. Aus der Abbildung 20 ist ersichtlich, dass das größte Datenvolumen über nur eine Domain (www.fein.de) bezogen wird und die restlichen fünf Domains nur einen Bruchteil des Gesamt-Traffics ausmachen. Eine Verteilung des Datenvolumens auf mehrere Domains wäre hier, gerade auch in Hinsicht auf den Internet Explorer 7, sinnvoll.

Cookies

Des Weiteren verrät ein Blick in den Anfrage-Header der HTTP-Requests, dass für die Domain www.fein.de mehrere Cookies gesetzt werden.



```
Anfrage-Header
Accept: image/png,image/*;q=0.8,*/*;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Cache-Control: no-cache
Connection: keep-alive
Cookie: wt3_eid=%3B572954779763707%7C2132636494900260842; emos_jcvid=AAABNNGDBATKaK7czFMSXIY_veh6PUYX:2:AAABNQC2VN:1:1327156843732:0:true:1; PHPSESSID=estfg793acubd4kreh55a73lo5; emos_clickmonitor=; emos_clickmonitor_sid=; emos_jcsid=AAABNQC2VNQno0j27PMs05js1s2Md014:69:AAABNSDUGyOxYJdgR3tOz**g6Z4OQPsm:1327695692579; wt3_sid=%3B572954779763707
Host: www.fein.de
Pragma: no-cache
Referer: http://www.fein.de/de_de/
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:9.0.1) Gecko/20100101 Firefox/9.0.1
```

Abbildung 21: HTTP-Anfrage-Header von Fein.de mit gesetzten Cookies

Diese Cookies werden für statistische Auswertungen und die PHP-Session-ID gesetzt. Es wirkt sich negativ auf die Geschwindigkeit aus, dass diese Cookies bei jeder Anfrage mitgesendet werden. Also auch wenn statischer Content, wie z.B. Bilder, CSS- oder JavaScript-Dateien, vom Server abgefragt wird. Dieser Overhead summiert sich natürlich bei 82 Requests pro Seitenaufruf. Mehr dazu im Kapitel 5.5.5 „Cookies“.

4.1.8 Unsortiertes HTML-Dokument

Wie im Kapitel 5.1.4 „Seitenaufbau planen“ erläutert, kann ein überlegtes und strukturiertes HTML-Dokument die User Experience des Seitenladevorgangs erheblich verbessern. Dabei kommt es darauf an, dem Benutzer möglichst schnell erste Ergebnisse dieses Vorgangs zu liefern und dementsprechend zunächst alle Inhalte, die für die erste Darstellung der Seite wichtig sind zu laden und erst am Ende des Dokumentes Komponenten wie z.B. JavaScript-Dateien für Animationen.

„Vom Wichtigen zum Unwichtigen“

In Falle von Fein.de werden einige JavaScript-Dateien unnötigerweise bereits im Kopfbereich der Seite geladen. Diese könnten auch am Ende geladen werden, um so den für die Darstellung wichtigen Komponenten wie Bild- und CSS-Dateien, den Vortritt zu lassen.

Im Hauptmenü werden einige Bilder eingebunden, die für die erste Darstellung der Seite aber nicht erforderlich sind. Sie werden erst sichtbar, wenn der Benutzer das Menü aufklappt. Hier könnte mit JavaScript dafür gesorgt werden, dass diese Bilder erst geladen werden, wenn die Seite komplett fertig dargestellt ist.

4.1.9 Ineffiziente Bildformate

Wie in Kapitel 5.2 „Bilder“ angesprochen, bestehen aktuelle Webseiten ungefähr zur Hälfte aus Bilddaten. Die Abbildung 13 visualisiert, wie sich das bei der Fein-Webseite verhält.

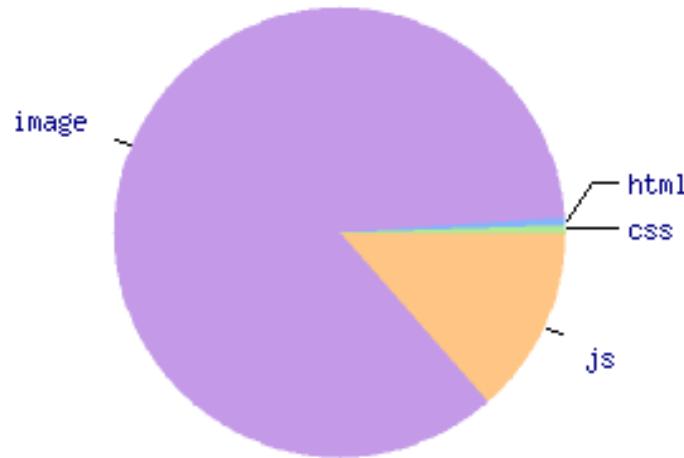


Abbildung 22: Zusammensetzung des Inhaltes der Fein-Webseite

Hier scheint die Webseite von Fein ein wenig aus dem Raster zu fallen, denn über drei Viertel des Seitenvolumens besteht aus Bildern. Diese Erkenntnis lässt bereits ein hohes Optimierungspotenzial erahnen.

Es gibt verschiedene Bildformate, die im Web-Bereich gebräuchlich sind. Diese haben je nach Anwendungsfall Vor- und Nachteile. Näheres dazu in Kapitel 5.2.1 „Bild-Formate“.

Bei genauerer Betrachtung fällt auf, dass sehr viele Bilder im PNG-Format gespeichert sind, obwohl diese aus vielen Farben bestehen. Für diese Art von Bildern ist das JPEG-Format viel effizienter. Ein kleiner Test bestätigt diese Theorie, dafür wurden zwei dieser im PNG-Format gespeicherten Bilder ins JPEG-Format konvertiert und die Dateigrößen verglichen. Die Tabelle 6 zeigt das Ergebnis, eine Effizienzsteigerung von fast 80% zeigt ein hohes Optimierungspotenzial auf.

Tabelle 6: Optimierungspotenzial der Bilder im PNG-Format

	10504.png	288296.png
Miniaturlansicht		
Größe als png	57kb	89kb
Größe als jpg	13kb	19kb
Optimierungsgrad	77%	79%

Weiterhin fällt insbesondere das Bild flag.png auf, es wird als ein CSS-Sprite genutzt, hat eine Größe von 232kb und die Abmessungen 350px × 1945px.



Abbildung 23: Ausschnitt aus dem CSS-Sprite flag.png

Die Abbildung 24 zeigt einen Ausschnitt dieses Bildes, es befinden sich die Flagge des jeweiligen Landes in zwei verschiedenen Größen und eine dazu gehörige Landesbezeichnung auf dem Bild. Das PNG-Format macht in diesem Fall Sinn, da die Alpha-Transparenz genutzt wird. Überflüssig ist allerdings die textuelle Beschreibung der

Flaggen, sicherlich soll diese zur Orientierung der Entwickler dienen, technisch gesehen macht sie die Datei allerdings unnötig groß.

Ein CSS-Sprite macht technisch gesehen auch nur dann Sinn, wenn auch ein Großteil der Bilder von der einbindenden Seite gebraucht werden. In diesem Fall profitieren nur andere Sprachversionen von dieser Seite. Es müsste also eine große Nutzerzahl geben die sich die Seite in vielen Sprachen anschauen um zu rechtfertigen, dass die Flaggen als ein Ganzes geladen werden. Allerdings ist so ein Nutzerverhalten nicht anzunehmen, weshalb es mehr Sinn ergibt die Ländericons einzeln für jede Sprache zu laden und dadurch den Ladeprozess zu beschleunigen.

4.2 Umsetzung

In diesem Kapitel werden die identifizierten Probleme aus dem letzten Kapitel im Einzelnen beseitigt. Nach jeder Optimierung wird eine Geschwindigkeitsmessung vorgenommen und mit der vorhergehenden verglichen. So wird ersichtlich, wie effizient sich jede Verbesserung auf die Gesamtperformance auswirkt.

Die Reihenfolge der Maßnahmen orientiert sich daran, wie aufwändig diese sind.

4.2.1 Aktivieren von HTTP Keep-Alive

Die wohl am schnellsten umsetzbare Optimierung ist die Aktivierung von HTTP Keep-Alive. Dazu wird in der Konfiguration des Apache2-Webservers das Flag Keep-Alive einfach von „Off“ auf „On“ gesetzt und der Webserver neu gestartet.

Schon bei der Betrachtung des Wasserfall-Modells ist sofort ersichtlich, dass die Verbindung am Anfang einmal aufgebaut wird und dann von allen Ressourcen genutzt wird. Die ersten fünf Komponenten (3. bis 7.) bauen noch separat Verbindungen auf, da sie parallel gestartet werden und zum Startzeitpunkt noch keine Verbindung besteht.

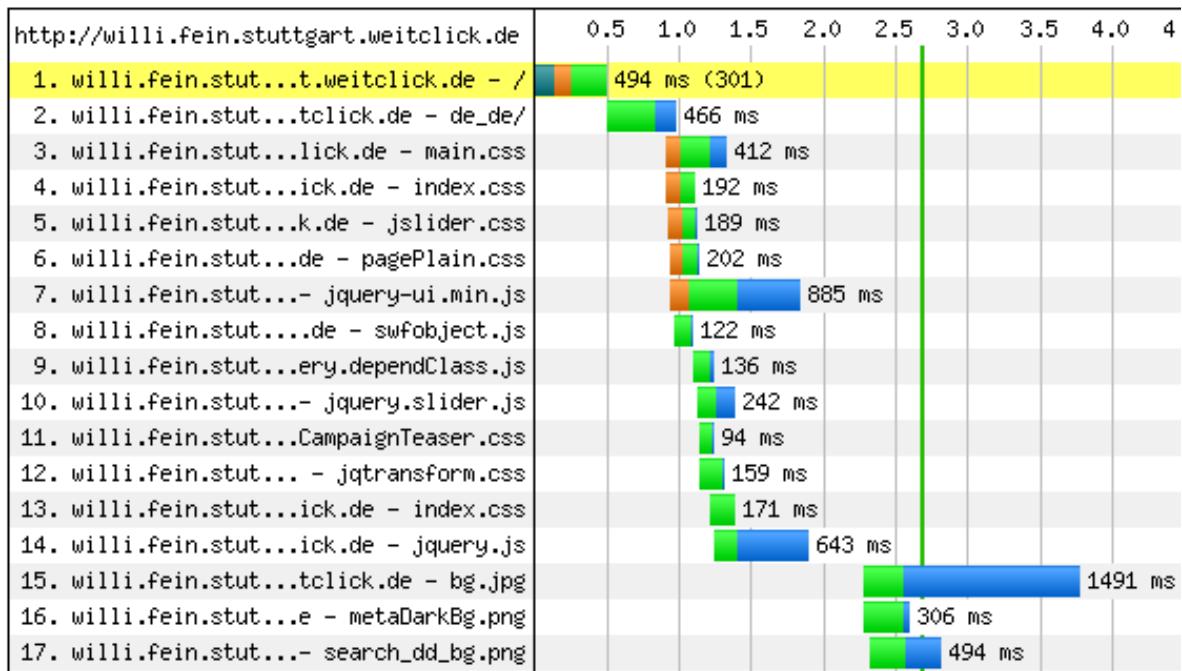


Abbildung 24: Wasserfall-Modell-Ausschnitt mit aktivem HTTP Keep-Alive

Auf die Gesamtladezeit des First-Views wirkt sich die Maßnahme ganz enorm aus, eine Steigerung von 30,12% kann hier im Vergleich zur Initialmessung verbucht werden. Für die Repeat View ergibt sich eine kaum spürbare Verbesserung der Performance um 1%, da hier nur wenige Requests gestartet werden und sich deshalb diese Maßnahme kaum bemerkbar macht.

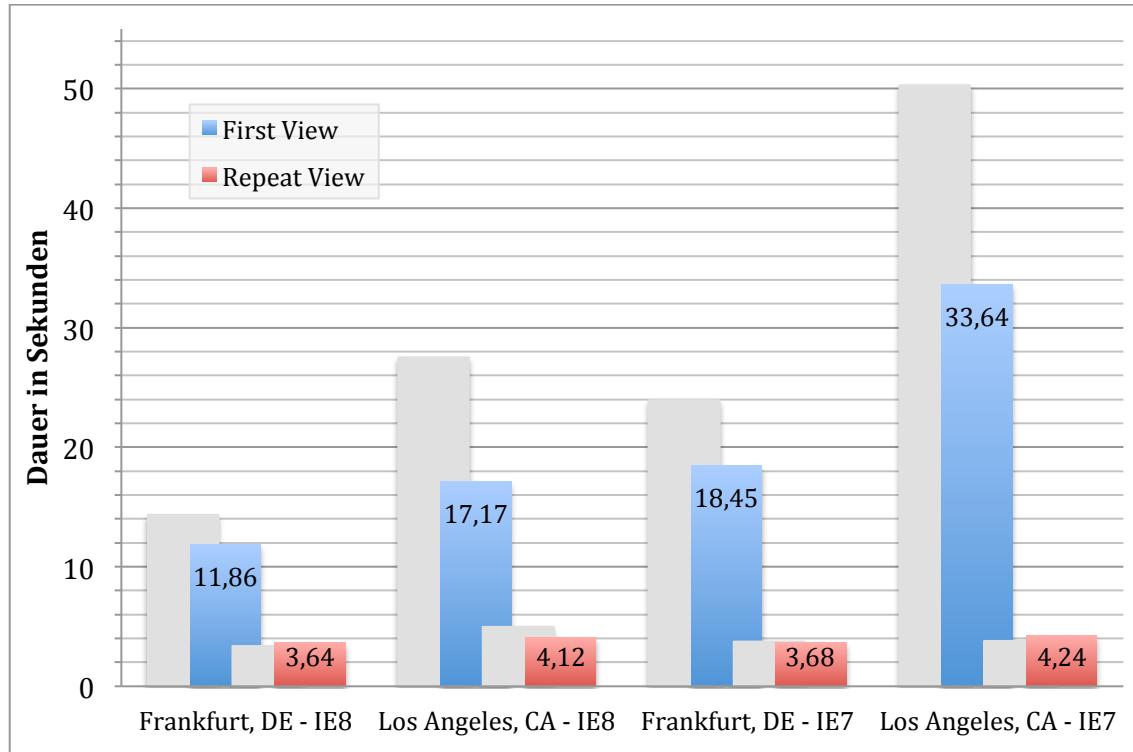


Abbildung 25: Performanceanalyse nach dem Einschalten von HTTP Keep-Alive

4.2.2 Redirect-Fehler beheben

Da beim Aufruf von www.fein.de zunächst auf www.fein.de/de_de und erst dann auf www.fein.de/de_de/ weitergeleitet wird, kann ohne den Zwischenschritt ein Redirect gespart werden. Mit der Modifizierung einer Codezeile ist auch dies eine Optimierung ohne großen Aufwand.

Interessant ist die Effektivität dieser Maßnahme, sie beschleunigt die First View um weitere 17,41% und die Repeat View um 15%.

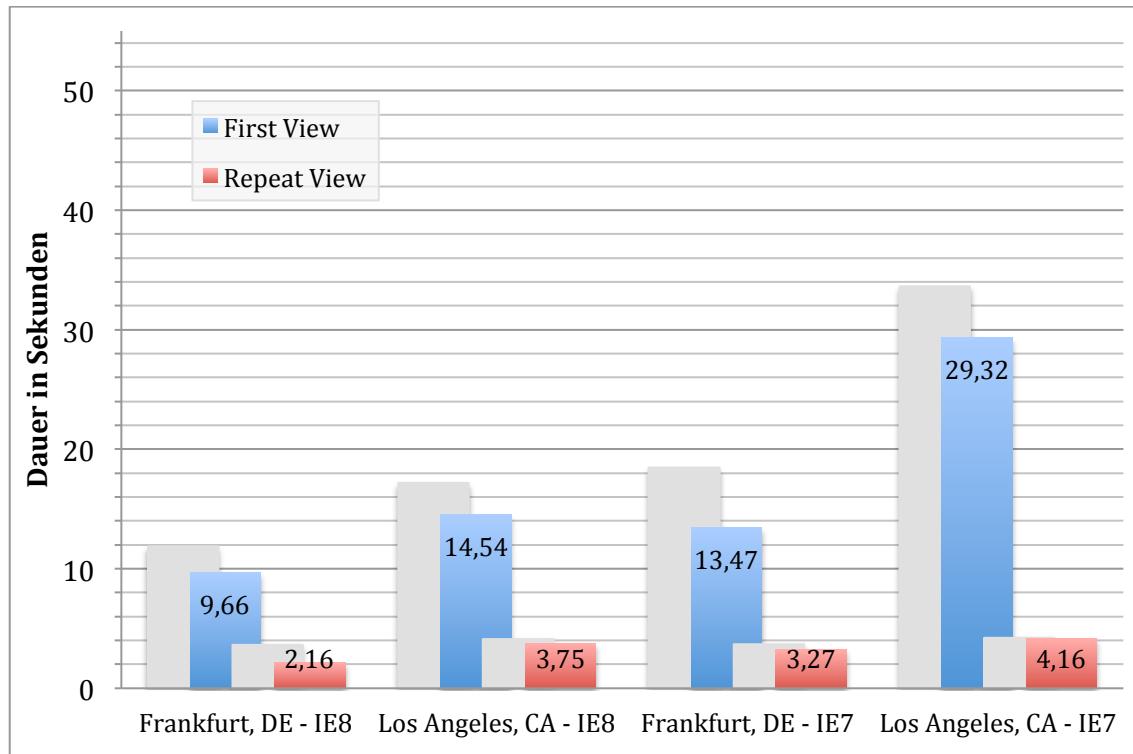


Abbildung 26: Performanceanalyse nach Behebung des Redirect-Fehlers

4.2.3 Bilder optimieren

Zunächst ist das Länderflaggen-CSS-Sprite entfernt und stattdessen nur die jeweils benötigte Flagge eingebunden worden. An dieser Stelle war die Idee eines CSS-Sprites zwar grundsätzlich gut, da ein Wechsel der Flagge aber nur in Verbindung mit dem Laden einer anderen Seite passiert, sollte eine andere Flagge auch erst dann nachgeladen werden.

Anschließend sind dann noch die PNG-Bilder, bei denen es Sinn gemacht hat, sie ins JPEG-Format umgewandelt zu haben.

Diese Änderungen haben bewirkt, dass die Größe der Seite von 2,4MB auf 1,2MB reduziert wurde. Nun könnte man annehmen, dass die Halbierung der Seitengröße auch eine Halbierung der Seitenladezeit nach sich zieht. Ein Blick auf das Ergebnis der Geschwindigkeitsauswertung (Abbildung 28) zeigt allerdings das dem nicht so ist.

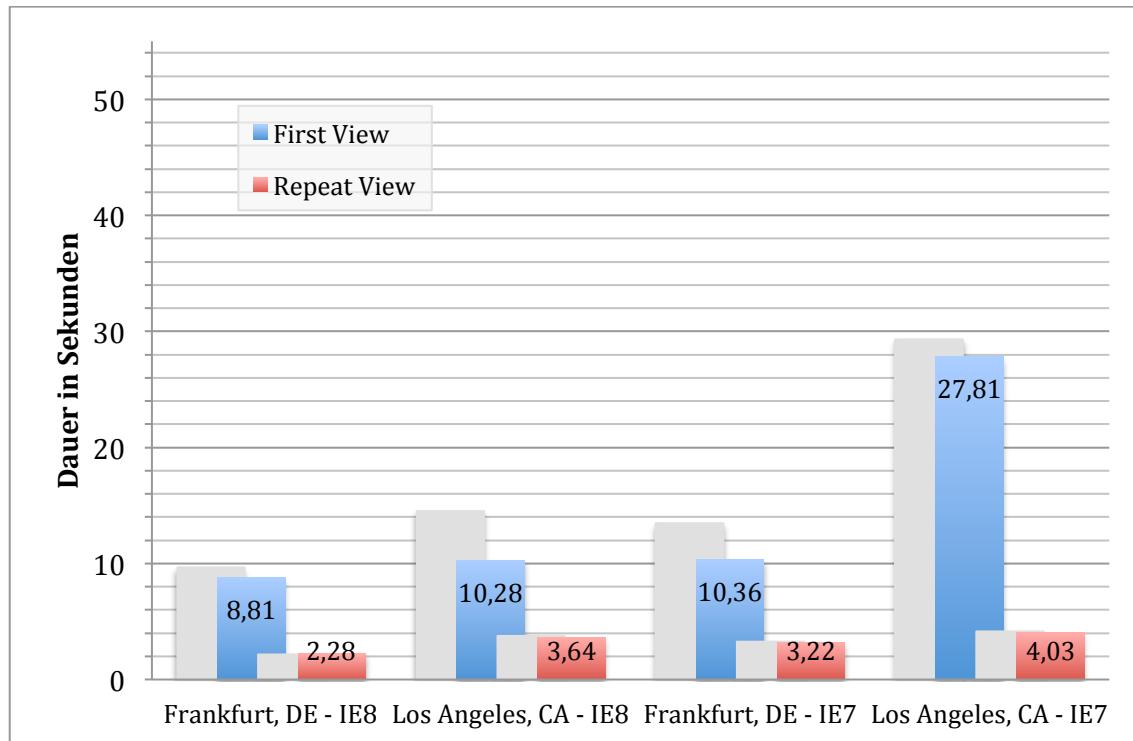


Abbildung 27: Performanceanalyse nach dem Optimieren der Bilder

Die First View ist demnach nun 14,53% schneller und auf die Repeat View hat diese Maßnahme eigentlich keinen Einfluss, da nur statischer Content optimiert wurde, der für diese Abfrage bereits im Cache des Browsers liegt. Dass die Halbierung der Seitengröße keinen so großen Einfluss auf das Gesamtergebnis hat, zeigt wie maßgeblich die Anzahl der HTTP-Requests für die Ladezeit ist, denn diese sind ja gleich geblieben.

4.2.4 Content-Verteilung auf eine weitere Domain

Um die Inhalte auf eine weitere Domain zu verteilen, wurde ein zusätzlicher virtueller Host⁴⁰ eingerichtet. Die Domäne des neuen vHosts (<http://static.willi.fein.stuttgart.weitclick.de>) zeigt dabei auf das gleiche Verzeichnis im Dateisystem wie die Haupt-Domäne (<http://willi.fein.stuttgart.weitclick.de>). Von dieser wird nun knapp über ein Viertel der Gesamtdatenmenge bezogen (Abbildung 25), und zwar ausschließlich Bilder. Im Vergleich zu vorher (Abbildung 15) ist eine größere Verteilung der Daten deutlich zu erkennen.

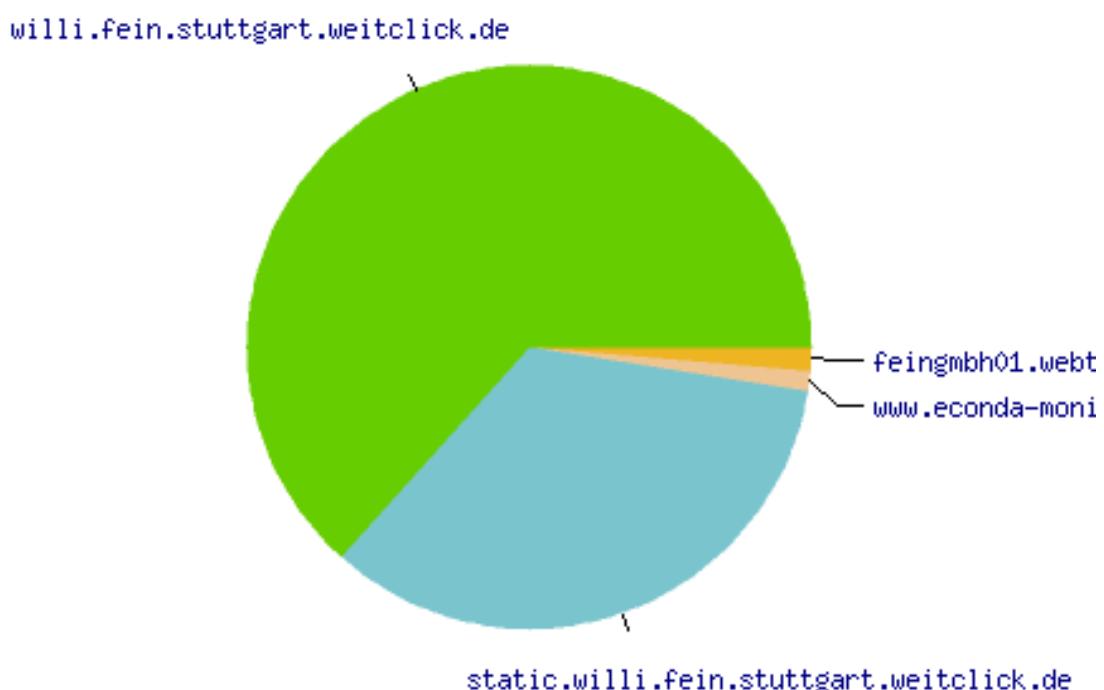


Abbildung 28: Datenmenge pro Domain im Verhältnis zum Gesamtvolumen (nach der Optimierung)

Ein Blick auf die Geschwindigkeitsanalyse zeigt, wie viel Zeit nun dadurch gespart wurde. Mit etwas über 1% ist die Abweichung der Repeat View nicht relevant, was allerdings zu erwarten war, da sich diese Maßnahme nur auf Bilder auswirkt, welche für die Repeat View aber bereits im Cache liegen. Die First View allerdings wurde um 22,12% verbessert. Wobei insgesamt keine so deutliche Verbesserung beim Internet Explorer 7 zu verzeichnen ist wie erhofft, lediglich der IE7 aus Los Angeles macht einen deutlich schnelleren Eindruck mit einer Verbesserung von 6s.

⁴⁰ Als virtueller Host (oder auch vHost) wird eine Technologie genannt, die es ermöglicht mehr als ein Webangebot auf einer einzigen Maschine zu betreiben.

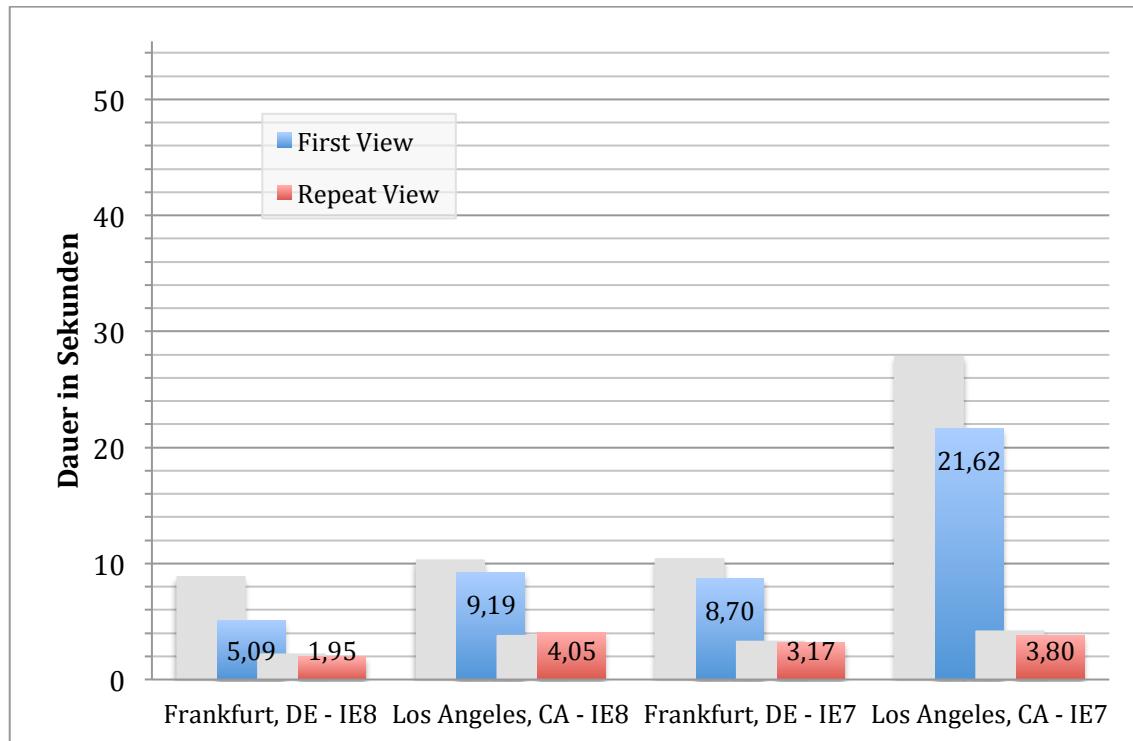


Abbildung 29: Performanceanalyse nach der Content-Verteilung auf eine weitere Domain

Das Wasserfall-Modell des IE7 aus Frankfurt (Abbildung 31) zeigt dann aber doch, dass die Maßnahme ihr gewünschtes Ziel erreicht hat. Es werden nun bis zu vier Komponenten gleichzeitig heruntergeladen.

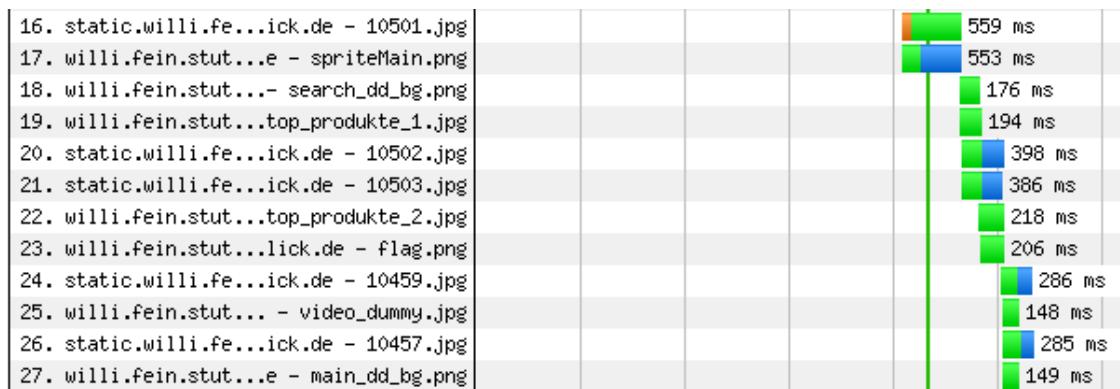


Abbildung 30: Wasserfall-Modell nach Content-Verteilung auf eine weitere Domain

4.2.5 Minimierung der HTTP-Requests

Damit die Anzahl der HTTP-Requests für den Initialaufruf der Seite auf das Nötigste beschränkt ist, wurden drei Dinge geändert. Zunächst wurden einige JavaScript-Dateien zu Einer zusammengefasst und an das Ende des HTML-Dokuments verlegt. Damit

konnten elf Requests gespart werden. Bei den CSS-Dateien konnten mit dieser Maßnahme 5 Requests gespart werden.

Weiterhin sind die Bilder für die Illustration des Menüs durch ein einheitliches Platzhalterbild ersetzt worden. Da diese Bilder erst in Erscheinung treten wenn das Menü aufgeklappt wird, ist es sinnvoll diese erst dann zu laden, wenn die Seite komplett fertig gerendert ist. Diese Herangehensweise wird „Lazy Loading“ genannt und mit Hilfe von JavaScript umgesetzt.

Als Ergebnis dieser Mühen ist eine Reduktion der HTTP-Requests um mehr als 50% möglich gewesen, statt 82 werden nun nur noch 40 Requests für die Initialdarstellung der Startseite benötigt.

Tabelle 7: Aufteilung der HTTP-Requests vor und nach der Optimierung

	insgesamt	Bilder	CSS	JS	HTML
Vorher	82	51	10	17	4
Hinterher	40	25	5	6	4

Auf die Geschwindigkeit hat diese Maßnahme eine sehr positive Auswirkung, wie in Abbildung 23 ersichtlich, wird dadurch die First View um 26,64% beschleunigt und die Repeat View um 12,09%.

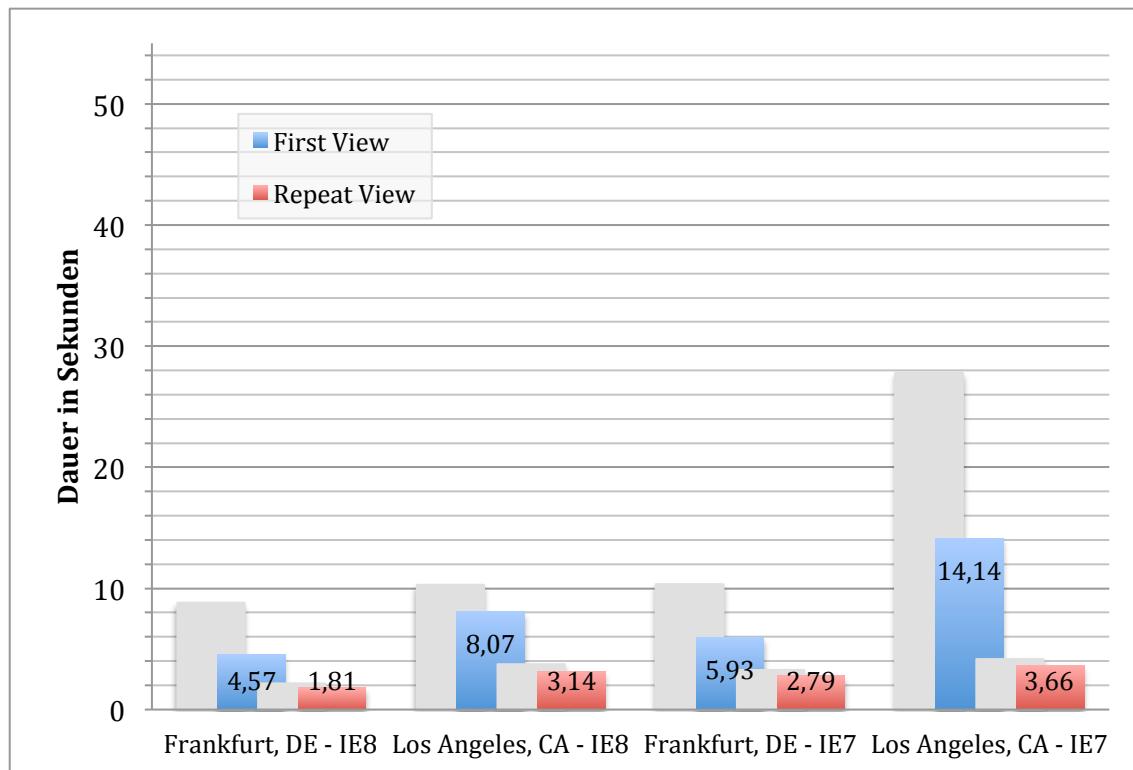


Abbildung 31: Performanceanalyse nach Content-Verteilung auf eine weitere Domain

Interessanterweise hat der Internet Explorer 7 von dieser Maßnahme stark profitiert. Innerhalb Deutschlands ist er dadurch um fast 3s und aus den USA um mehr als 7s schneller geworden. Offensichtlich kann dieser schlechter mit vielen Requests umgehen als aktuellere Browser.

4.3 Ergebnis

Bei der Betrachtung der Ladezeit vor und nach der Umsetzung (Abbildung 33) ist eine starke Verbesserung der First View zu verzeichnen, wohingegen die Repeat View keine großen Performanceverbesserungen aufweisen kann. Das hängt mit der Zielsetzung und der dadurch resultierenden Ausrichtung der ergriffenen Maßnahmen zusammen.

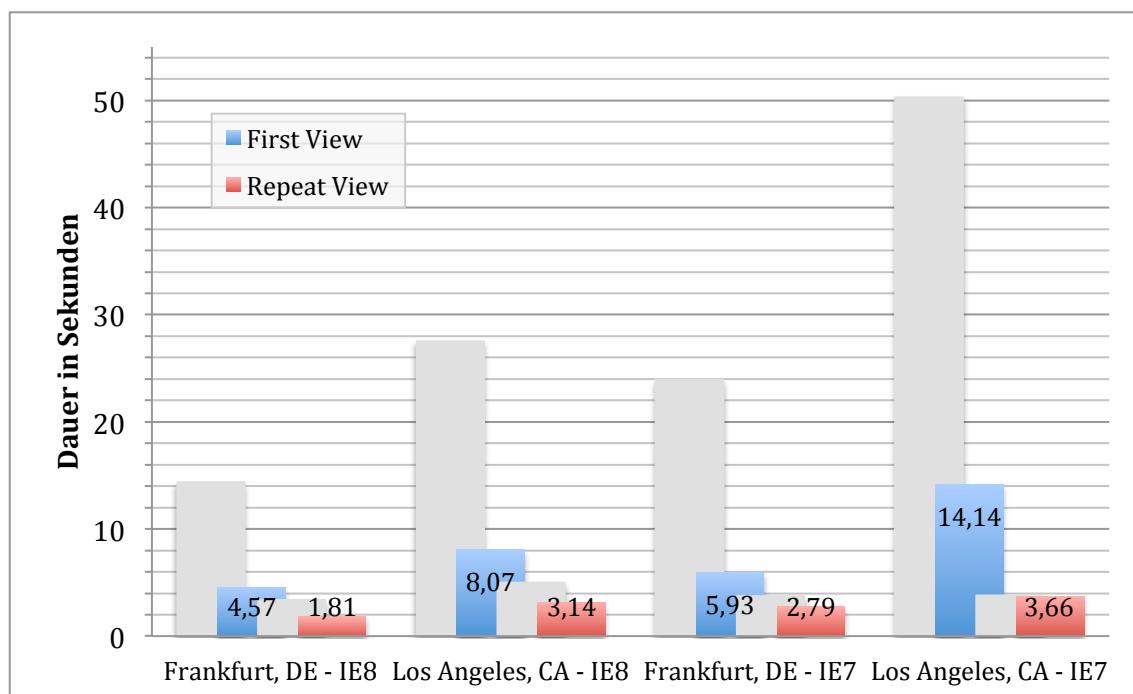


Abbildung 32: Vergleich der Ladezeiten vor und nach den Optimierungsmaßnahmen

Die Zielsetzung, dass die First View in diesem Testszenario nicht mehr länger als 15s brauchen und der Internet Explorer 7 im Focus der Optimierung stehen sollte, wurde erreicht. Die längste Seitenladedauer beträgt nun 14,14s für den IE7 aus den USA. Damit ist dieser 36,12s schneller als noch vor der Optimierung. Aber auch innerhalb Deutschlands ist der IE7 viel schneller als zuvor, es dauert nun keine 6s mehr (vorher 23,94s) bis die Seite fertig dargestellt wird.

Die Repeat-View war bereits vor der Optimierung schon zügig weshalb darauf kein gesonderter Focus gerichtet war. Rückwirkend betrachtet hat nur die Beseitigung des Redirect-Fehlers (6.3.2 „Redirect-Fehler beheben“) sich effektiv auf die Repeat-View ausgewirkt.

Alles in allem war diese Maßnahme sehr wichtig für das Projekt, da die beiden Schwachstellen „Internet Explorer 7“ und „Abruf aus dem Ausland“ im Vorfeld nicht als Probleme bekannt waren. Interessant wird es sein, wie sich die Klickpfade der Besucher in Zukunft, nach der Anwendung dieser Maßnahmen, auf das Produktivsystem auswirken werden. Es wird sich zeigen ob, nach der Übernahme dieser Änderungen auf das Produktivsystem, die Mutmaßungen aus dem Kapitel 6.2.1 „Klickpfade“ richtig waren oder nicht.

Durch die Optimierung der Bilder konnten 50% (1,2MB) der zu übertragenden Datenmenge für jeden First View eingespart werden, das bedeutet eine monatliche Traffic-Ersparnis von 150,56GB bei 128.478 eindeutigen Besuchern (Januar 2012). Das wirkt sich nicht nur positiv auf die Belastung des Servers und die Geschwindigkeit der Webseite aus, es kann sogar Geld damit gespart werden. Denn die Kosten für das Hosting der Webseite werden pro übertragenes GB berechnet.

Abschließend lässt sich noch sagen, dass ein gesonderter Blick auf die Performance enorm wichtig für ein erfolgreiches Webprojekt ist. Je früher in einem Projekt die Probleme erkannt werden, desto leichter ist es diese zu beheben.

5 Ausblick

Der Internet-Boom ist nach wie vor ungebrochen und eine große Menge von Menschen arbeiten daran es stetig zu verbessern. In Zukunft wird nach Meinung des Autors das Thema Performance erheblich an Bedeutung gewinnen, da die Webanwendungen welche sich bereits jetzt technisch auf hohem Niveau befinden, schon bald den Anspruch erheben werden ihre Desktop-Pendants zu ersetzen. Ein Beispiel ist das klassische E-Mail-Programm (z.B. Outlook oder Mail), welches schon heute durch online Systeme wie Google-Mail⁴¹ unter Druck gerät.

In Zukunft wird es egal sein, an welchem Computer der Benutzer sich gerade befindet, wichtig für die Nutzung seiner gewohnten Programme wird nur der Internetanschluss sein. Auch wird die Software nicht mehr vom Benutzer selbst gewartet sondern nur noch an einem zentralen Ort vom Hersteller, dadurch können Innovationen und auch Sicherheitslücken viel schneller produktiv gehen.

Für so einen Evolutionsschritt muss allerdings nicht nur die Menge an Features, sondern vor allem auch die Performance der Anwendungen stimmen. Ansonsten steigen die Nutzer vermutlich nicht auf ein neues System um, weil die Frustration zu groß wäre.

Zusammen mit dieser Entwicklung wird eine gute Performance auch immer mehr zu einem Qualitätsmerkmal werden und dadurch als Verkaufsargument auch in der Marketingabteilung ankommen.

Ein weiteres großes Thema heute und auch in Zukunft ist das mobile Internet. Prognosen sagen voraus, dass die mobile Nutzung des Internets mit ihrem derzeitigen Wachstum die Desktop-Nutzung in den nächsten Jahren überholen wird.

In diesem Bereich ist das Thema Performance noch viel brisanter, da den Nutzern bisher nicht die vom Desktop gewohnte Verfügbarkeit und Bandbreite zur Verfügung steht. Das bedeutet, Inhalte die für einen Desktop-Browser funktionieren, müssen für die mobile Darstellung speziell aufbereitet werden.

Als Abschluss dieses Ausblickes werden nun noch einige Technologien vorgestellt, die zwar noch in der Entwicklung stecken, aber vielleicht in Zukunft das Internet aus der Performancesicht vorantreiben.

⁴¹ <http://mail.google.com/>

5.1 HTTP-Pipelining

Mit HTTP-Pipelining ist es möglich, über einen HTTP-Request gleich mehrere Komponenten einer Webseite zu beziehen. Es müssen also nicht mehr alle Komponenten einzeln vom Server nachgeladen werden. Diese Technologie bricht mit dem strikten Muster „Request senden, auf Response warten“, stattdessen können parallel mehrere Requests über eine HTTP-Verbindung an den Server gesendet und empfangen werden.

Die Idee ist genauso alt wie die von Keep-Alive, nämlich seit dem HTTP1.1 Standard der bereits 1999 verabschiedet wurde. Diese Technologie wurde aber niemals offiziell von einem Browser unterstützt, da es noch zu viele Schwierigkeiten mit Proxy-Servern und dem Caching gibt.

Dennoch ist sie nicht ganz vergessen worden, im Firefox ist es beispielsweise möglich diese Funktion zu aktivieren. Dazu muss die Konfiguration, mit der Eingabe von *about:config* in die Adresszeile, geöffnet werden. Anschließend können alle relevanten Einstellungen mit der Eingabe von *network.http.pipelining* herausgefiltert werden.

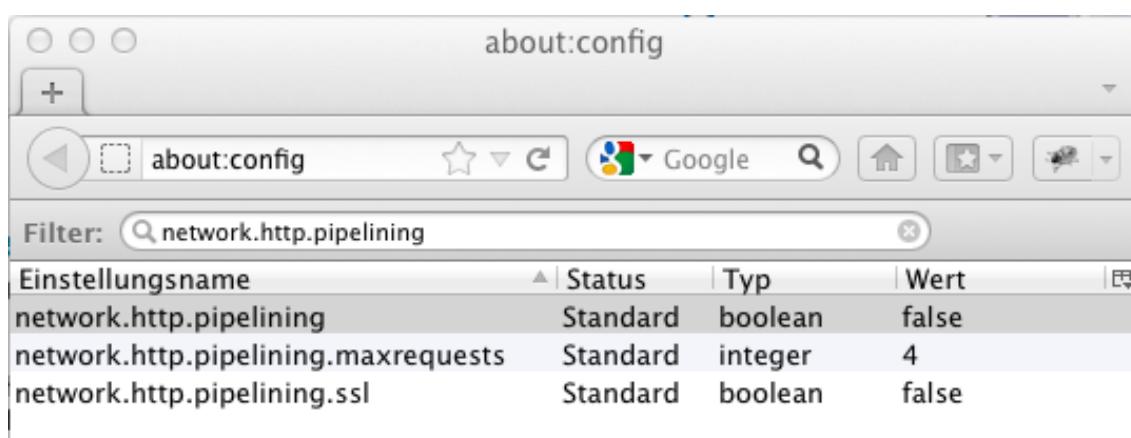


Abbildung 33: Aktivieren von HTTP-Pipelining im Firefox

Mit einem Doppelklick auf den Wert des Attributes *network.http.pipelining* aktiviert man das Pipelining und mit dem Attribut *network.http.pipelining.maxrequests* kann die Anzahl der Requests pro HTTP-Connection definiert werden.

Für einen Test muss natürlich noch ein Server angefragt werden, bei dem das Pipelining ebenfalls aktiviert ist.

Eine gute Quelle für detailliertere Informationen ist:

<http://www.igvita.com/2011/10/04/optimizing-http-keep-alive-and-pipelining/>

Diese Technologie sollte in Zukunft im Auge behalten werden, denn vielleicht erlebt sie ja noch eine Renaissance.

5.2 HTML5 Application Cache

Mit dem neuen HTML5 Standard wird das bisherige Browser-Cache-System, wie in Kapitel 5.5.3 „Expires Header und der ETag“ beschrieben, erweitert. Damit ist es möglich gezielt bestimmte Dateien zu cachen und offline Verfügbar zu machen, aus diesem Grund wird diese Technologie auch als Offline-Storage bezeichnet.

Bisher war es so, dass der Browser die Kontrolle darüber hatte, was sich in seinem Cache befindet und wie lange. Diese wird nun an den Entwickler übergeben, welcher mit Hilfe einer Manifest-Datei das Verhalten des Caches steuern kann.

Mit dieser Technik ist es möglich ganze Anwendungen offline verfügbar zu machen. Somit wird der große Nachteil, immer eine Verbindung zum Internet zu benötigen, gegenüber den Desktop-Anwendungen nichtig.

Gesteuert wird der Cache über eine Manifest-Datei, die über das Attribut *manifest* im *html*-Tag der HTML-Seite referenziert wird.

```
1 <html manifest="http://www.example.com/example.mf">
2 ...
3 </html>
```

Quellcode 7: HTML5 Application Cache - *manifest*-Attribut

Innerhalb dieser Datei können die Einstellungen zur Steuerung des Cacheverhaltens gemacht werden. Quellcode 8 zeigt eine Beispielkonfiguration. Die Zeile 1 ist zwingend erforderlich und gibt dem Parser zu verstehen, dass es sich hierbei um eine Manifest-Datei handelt.

Zeile 2 ist nur ein Kommentar mit der Versionsnummer dieser Datei. Wichtig in diesem Zusammenhang ist, dass der Browser den Cache nur wieder leert, wenn die Manifest-Datei sich ändert. Änderungen an den statischen Inhalten bewirken kein Erneuern des Caches. Deshalb wird so eine Versionsnummer häufig dafür verwendet, dem Browser zu sagen, dass er den Cache neu aufbauen soll.

In Zeile 4 wird mit *CACHE*: die Sektion eingeleitet, in der alle Dateien aufgelistet werden (Zeile 5 - 9), die in den Cache aufgenommen werden sollen.

Die Sektion *NETWORK*: in Zeile 11 definiert Dateien, die nur verwendet werden können, wenn der Benutzer online ist.

Und in Zeile 16 werden mit *FALLBACK*: alternativ Dateien angegeben, wenn die Seite im Offline-Modus verwendet wird. Z.B. können im Offline-Modus keine Bilder nachgeladen werden, sodass statt der Bilder aus dem Pfad *images/large/* nur das Bild *images/offline.jpg* angezeigt werden soll (Zeile 18).

```
1 CACHE MANIFEST
2 # 2010-06-18:v2
3
4 CACHE:
5 /favicon.ico
6 index.html
7 stylesheet.css
8 images/logo.png
9 scripts/main.js
10
11 NETWORK:
12 login.php
13 /myapi
14 http://api.twitter.com
15
16 Fallback:
17 /main.py /static.html
18 images/large/ images/offline.jpg
19 *.html /offline.html
```

Quellcode 8: HTML5 Application Cache - Manifest-Datei

Das Problem bei dieser neuen Technik ist, dass die Internet Explorer diese nicht durchgängig unterstützen. Deshalb wird der Application-Cache zunächst im Mobile-Bereich verstärkt eingesetzt, da dort die Browser alle sehr aktuell sind und das Problem keine Internet-Verbindung zur Verfügung zu haben noch größer ist.

5.3 WebP

WebP wird von Google unter einer Open Source Lizenz entwickelt und ist ein neues verlustbehaftetes Grafikformat, optimiert für die Anwendung im Web. Es ist dem JPEG-Format sehr ähnlich, verspricht allerdings eine 30% bessere Komprimierung als JPEG bei gleicher Qualität. Außerdem beherrscht es Alpha-Transparenz wie das PNG-Format und Animationen wie das GIF-Format.

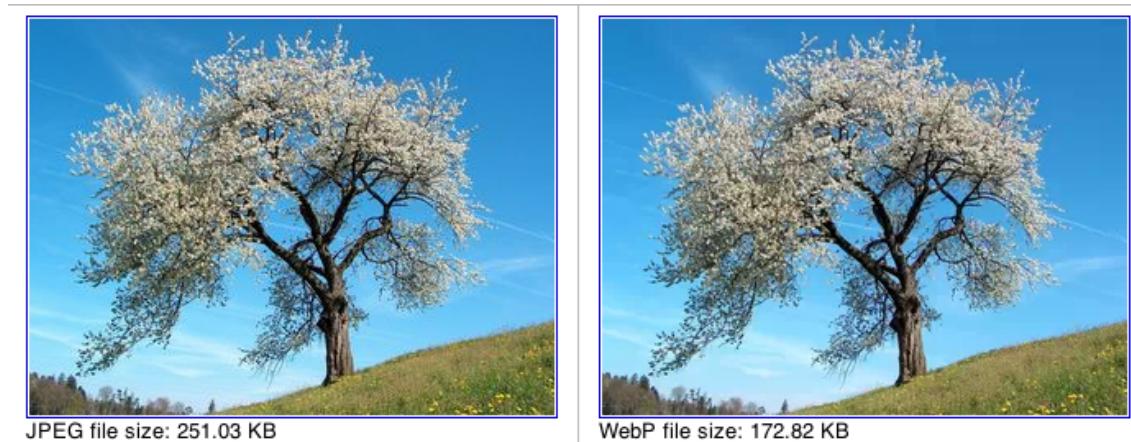


Abbildung 34: WebP vs. JPEG

Abbildung 33 zeigt wo WebP seine große Stärke ausspielen kann, nämlich im Komprimieren von Fotografien. Das WebP ist um 31% kleiner als das JPEG-Bild und das ohne erkennbaren Qualitätsverlust.

Der Haken an dem Format ist, dass es zurzeit nur vom Chrome-Browser unterstützt wird. Es bleibt zu hoffen, dass andere große Browserhersteller bald auch dieses Format unterstützen.

5.4 SPDY

SPDY (gesprochen SPeDY) ist ein experimentelles Web-Protokoll mit dem Ziel, das Internet schneller zu machen. Es wird von Google im Rahmen der "Let's make the web faster"-Initiative⁴² entwickelt.

Rahmenbedingungen des Projektes im Einzelnen:

- Die Seitenladegeschwindigkeit soll um 50% gesenkt werden.
- Um die Integration in die bestehende Internetlandschaft möglichst unkompliziert zu halten setzt SPDY wie gehabt auf das Transport-Protokoll TCP.

⁴² <http://code.google.com/speed/>

- Damit Autoren von Webseiten ihre Inhalte nicht anpassen müssen, braucht lediglich der Browser und der Server eine Unterstützung für SPDY.
- Um andere Entwicklergruppen an diesem Projekt teilhaben zu lassen und es dadurch mehr zu fördern, ist das Projekt Open Source⁴³.

Einige konkrete technische Ziele:

- Es sollen viele gleichzeitige HTTP-Requests über eine TCP-Verbindung laufen können.
- Um die Bandbreite besser ausnutzen zu können, sollen die HTTP-Header komprimiert und auf das Nötigste reduziert werden.
- SSL soll grundsätzlich für den Transportweg eingesetzt werden. In Zukunft soll kein unverschlüsselter Datentransfer mehr stattfinden.
- Der Server soll befähigt werden, von sich aus Inhalte an den Client zu senden (Push).

Die Umsetzung

SPDY baut eine Session-Layer auf das SSL-Protokoll, um mehrere verzahnte Datenströme über eine TCP-Verbindung leiten zu können. Dabei bleibt das klassische HTTP-GET, –POST Prinzip erhalten, allerdings wird ein neues Paketformat für die Kodierung und Übertragung der Daten etabliert. Die Datenströme sind Bi-Direktional und können sowohl vom Server als auch vom Client initiiert werden.

Entwicklungsstand

Bisher wurde ein Server entwickelt, der SPDY- und HTTP-Verbindungen über SSL und TCP aufbauen kann. Außerdem ein modifizierter Google Chrome-Browser, der auch beide Technologien versteht und mit dem Server kommunizieren kann. Und um die Performance in diesem Szenario messen zu können, eine Testumgebung.

Der Ansatz dieses neuen Protokolls ist deshalb sehr vielversprechend, da es nach Meinung des Autors für genug Performancegewinn sorgt, um die Browser- und Web-Server-Hersteller dazu zu bewegen, ihre Systeme kompatibel zu machen. In Labortests wurden bis zu 64% schnellere Seitenladezeiten gemessen (chromium.org, 2012).

⁴³ Open Source bedeutet so viel wie „quelloffen“ und beschreibt eine Bewegung der Softwareentwicklung, die es jedem Menschen ermöglicht, an einem Projekt teilzuhaben.

6 Fazit

Durch das Lesen der Fachliteratur, das Besuchen von Vorträgen und den Gesprächen mit Experten habe ich als Autor dieser Arbeit eine Menge dazugelernt und für mich eine neue Facette des Web-Entwicklerberufes erschlossen.

Ich bin der Meinung, jeder der gut in diesem Berufsfeld sein möchte, sollte sich auch mit dem Thema Performance auseinandersetzen, da die Qualität der eigenen Arbeit durch die Berücksichtigung einfacher Best Practices erheblich gesteigert werden kann.

Durch die praktische Umsetzung der theoretischen Kenntnisse kann ich nun nach einer Analyse einschätzen, an welchen Stellen eine Webseite ihre Geschwindigkeitsprobleme hat und entsprechende Maßnahmen dagegen ergreifen. Darüber hinaus wird bei der Entwicklung neuer Webseiten der Performancegedanke von vornherein eine größere Beachtung finden.

Ein Beispiel: Würde ich jetzt eine Webseite auf seine Performanceschwächen hin analysieren, so würde ich, wenn überhaupt, erst ganz zum Schluss einen Blick auf die CSS-Selektoren⁴⁴ werfen und schauen ob diese vielleicht ineffizient gewählt wurden. Denn ich weiß, dass hier nicht so viel Zeit gut zu machen ist. Entwickle ich allerdings in Zukunft eine neue Webseite, so werde ich direkt effiziente CSS-Selektoren verwenden, weil es für mich keinen größeren Aufwand bedeutet.

⁴⁴ 5.4.1 „Effiziente Selektoren verwenden“

Anhang A: Verwendete Tools

Für diese Arbeit wurden einige Tools benutzt um Geschwindigkeitsberechnungen durchzuführen, unter die „Haube“ von Webseiten zu schauen oder Vorgänge zu visualisieren. Nachfolgend werden diese kurz vorgestellt.

Page Speed

Page Speed ist ein Open Source Projekt von Google und analysiert Webseiten auf ihre Geschwindigkeitsschwachstellen hin. Auf der Grundlage dieser Auswertung gibt das Tool dann Tipps, an welchen Stellen die Seite besonders viel Zeit verliert und was man dagegen tun kann.

Mittlerweile ist es nicht nur mehr eine Browser-Extension für den Chrome-Browser, seine Aufgabe erledigt Page Speed jetzt auch direkt online oder als Grundlage für weitere frei verfügbare Tools, wie z.B. WebPagetest.org.

The screenshot shows the Google Developers Page Speed Online interface. At the top, there's a navigation bar with the Google Developers logo, a search bar containing "Search Google Developers", and a magnifying glass icon. Below the header, the URL "Page Speed — www.fein.de/ [Edit](#)" is displayed. The main content area is titled "Overview". It shows a summary: "The page Elektrowerkzeuge für den professionel... got an overall Page Speed Score of 97 (out of 100). [Learn more](#)". There are three sections of recommendations: "Medium priority (1)" with one item: "JavaScript später parsen"; "Low priority (9)" with nine items including "Bilder optimieren", "CSS (klein) inline einfügen", and "Umleitungen minimieren"; and "Experimental rules (2)" with two items. A note states: "⚠ Note that Page Speed Online followed a redirect from <http://www.fein.de/> and analyzed the page at http://www.fein.de/de_de/.". Another note says: "💡 This Page Speed report is generated for this page as it appears in desktop browsers. To get suggestions on how to optimize the page for mobile devices, generate a [mobile report](#)." Below this is a "Suggestion Summary" section with a note: "Click on the rule names to see suggestions for improvement." It lists three types of suggestions: "High priority", "Medium priority", and "Low priority". The "Low priority" section includes a note: "💡 Experimental rules These suggestions are experimental, but do not affect the overall Page Speed score. Consider these items if you want to make your site faster without impacting the overall score." At the bottom, there's a footer note: "« Experimental rules These suggestions are experimental, but do not affect the overall Page Speed score. Consider these items if you want to make your site faster without impacting the overall score. »".

Abbildung 35: Page Speed Online Übersicht

Am Ende einer Analyse bekommt die getestete Seite eine Punktebewertung von 0 bis 100 wobei 100 der besten Bewertung entspricht.

Diese Tools sind sehr hilfreich, man könnte denken, sie würden einem die Arbeit einer detaillierten Performanceanalyse bereits abnehmen. Das Problem dabei ist allerdings, dass sie immer nach einem strikten Muster vorgehen und somit kleine Abweichungen ihrer Vorgehensmodelle nicht verstehen. Beispielsweise wurden im Falle von Fein.de die Bilder zwar angemerkt, weil noch Optimierungspotenzial in den PNG-Bildern steckte. Dass diese aber im ganz falschen Format waren und mit dem JPEG-Format ein viel höherer Komprimierungsgrad erreicht werden konnte, hat Page Speed nicht entdeckt.

Für einen ersten Überblick und eine schnelle Analyse haben diese Tools ihre Daseinsberechtigung, sie ersetzen aber nicht den differenzierten Verstand eines Fachmannes, der die Zusammenhänge der einzelnen Performancestellschrauben kennt.

Mehr Informationen hier: <http://code.google.com/intl/de-DE/speed/page-speed/>

YSlow

YSlow ist das Yahoo!-Pendant zu Googles Page Speed. Es ist für diverse Browser als Plugin verfügbar und kann sogar über die Kommandozeile aufgerufen werden. Auch YSlow bewertet die getestete Webseite, allerdings mit den Noten A-F, wobei A der Bestnote entspricht.

The screenshot shows the YSlow interface with the following details:

- Header:** Home, Grade (selected), Components, Statistics, Rulesets (YSlow(V2)), Edit, Help.
- Grade:** A Overall performance score 92. Ruleset applied: YSlow(V2). URL: http://www.yahoo.com/
- Filter:** ALL (23) FILTER BY: CONTENT (6) | COOKIE (2) | CSS (6) | IMAGES (2) | JAVASCRIPT (4) | SERVER (6)
- Grade F on Make fewer HTTP requests:**
 - F Make fewer HTTP requests**
 - A Use a Content Delivery Network (CDN)**
 - A Avoid empty src or href**
 - B Add Expires headers**
 - A Compress components with gzip**
- Grade F Summary:**
 - This page has 4 external Javascript scripts. Try combining them into one.
 - This page has 26 external background images. Try combining them with CSS sprites.

Abbildung 36: YSlow Übersicht

Mehr Informationen hier: <http://developer.yahoo.com/yslow/>

Eine Besonderheit von YSlow ist die Integration einer Bildoptimierungsfunktion. Diese heißt Smush.it, wird auch von Yahoo entwickelt und ist eine Sammlung von Techniken um die Dateigröße von Grafiken für den Einsatz im Web zu optimieren.

The screenshot shows the Smush.it results page. At the top, it says "Smushed 17.50% or 90.71 KB from the size of your image(s). How did we do it? See the table below for more details." Below this, there is a yellow button labeled "Download Smushed Images" and a checkbox labeled "Keep directory structure in zip file". A table titled "Smushed Images" lists three files: "yarddraw.jpg", "start.png", and "rolling_table_tennis_ball.JPG", along with their original sizes, saved sizes, percentage savings, and status.

Image	Result size	Savings	% Savings	Status
yarddraw.jpg	74.35 KB	22.65 KB	23.35%	
start.png	250.54 KB	55.35 KB	18.09%	
rolling_table_tennis_ball.JPG	102.71 KB	12.71 KB	11.01%	

Abbildung 37: Smush.it Übersicht

Für die Optimierung können Bilder hochgeladen oder über eine URL angegeben werden. Nach der Optimierung werden die Ersparnisse pro Bild angezeigt und das neue zum Download angeboten.

Mehr Informationen hier: <http://www.smushit.com/>

WebPagetest

WebPagetest ist das Tool mit dem die meisten Messungen in dieser Arbeit durchgeführt worden sind. Es baut auf dem zuvor vorgestellten Tool Page Speed auf und erweitert es. So ist es damit möglich Messungen, von unterschiedlichsten Orten mit individueller Verbindungsgeschwindigkeit, durchzuführen.

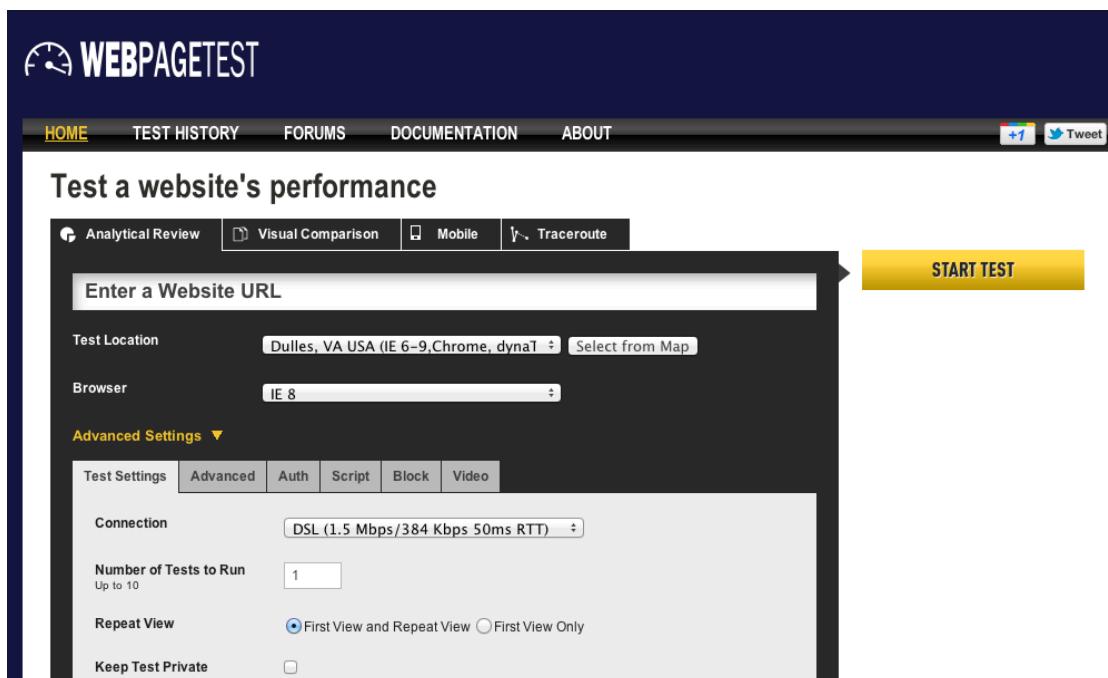


Abbildung 38: WebPagetest Übersicht

Dieses Tool ist deshalb so gut, weil es alle wichtigen Informationen gut visualisiert und bis zu 10 Tests direkt hintereinander durchführt, um Messungsspitzen zu filtern.

Die wichtigsten Visualisierungen sind das Wasserfall-Diagramm und die Kuchendiagramme über die Zusammensetzung der Webseite. Es gibt aber auch noch die Möglichkeit, sich ein Video vom Aufbau der Seite erzeugen zu lassen.

Mehr Informationen hier: <http://www.webpagetest.org>

PCAP Web Performance Analyzer

Dieser Online-Service visualisiert Dateien im *har*-Format. Er ist wichtig, um sich die Auswertungen von WebPagetest im Nachhinein noch einmal anschauen zu können.

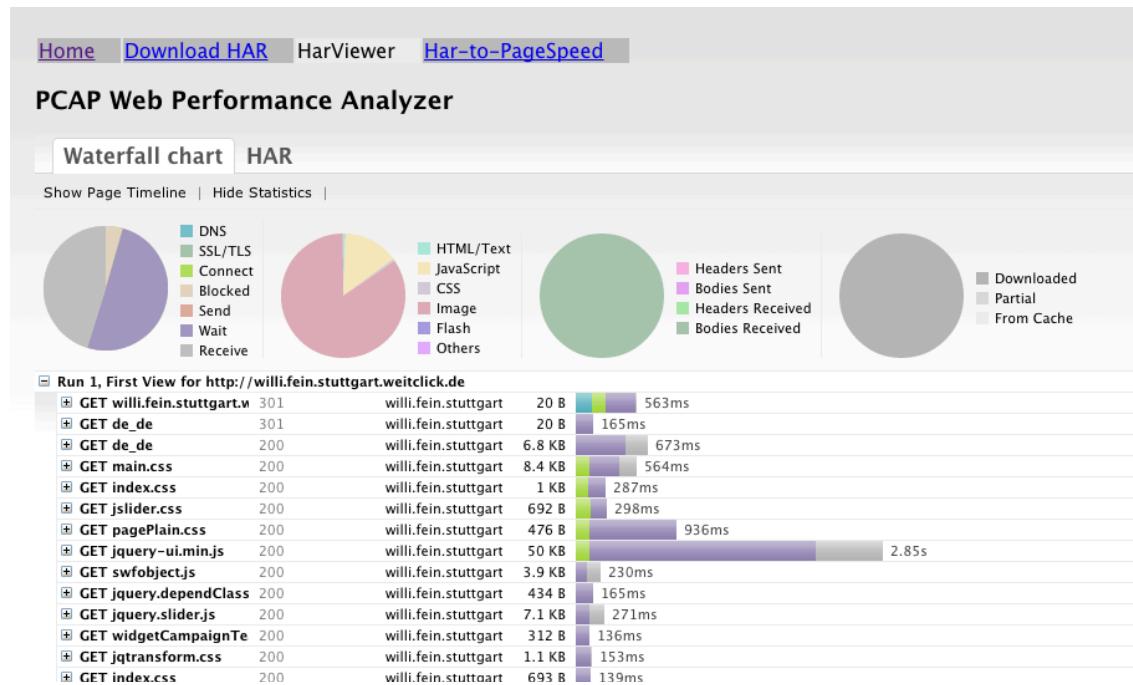


Abbildung 39: PCAP Web Performance Analyzer Übersicht

Neben dem Wasserfall-Diagramm und einigen Statistiken über die Zusammensetzung der analysierten Webseite, bietet das Tool auch die Möglichkeit, direkt zu dieser Auswertung eine Page Speed Analyse anzeigen zu lassen.

Mehr Informationen hier: <http://pcapperf.appspot.com/>

Firebug für den Firefox

Das Plugin „Firebug“ für den Firefox ist wohl eines der wichtigsten Tools, um einen ersten Blick unter die „Haube“ einer Webseite zu werfen. Es bietet die Möglichkeit, den Ladevorgang einer Webseite live mit zu verfolgen, um sich anhand eines Wasserfall-Modells einen ersten Eindruck zu verschaffen.

Darüber hinaus kann der HTML-Quellcode der Seite eingesehen und manipuliert werden, es gibt eine Übersicht der CSS-Eigenschaften, welche ebenfalls direkt geändert werden können und eine JavaScript-Console sorgt für ein bequemes Debugging.

Alles in allem gehört dieses Tool schon seit Jahren zur Grundausstattung jedes Web-Entwicklers und stellt die Basis für jegliche Optimierungsansätze dar. Es gibt Bemühungen seitens der Browser-Hersteller, diese Funktionalität nativ in die Browser zu bringen, bisher reicht deren Funktionsumfang allerdings noch an den von Firebug heran.

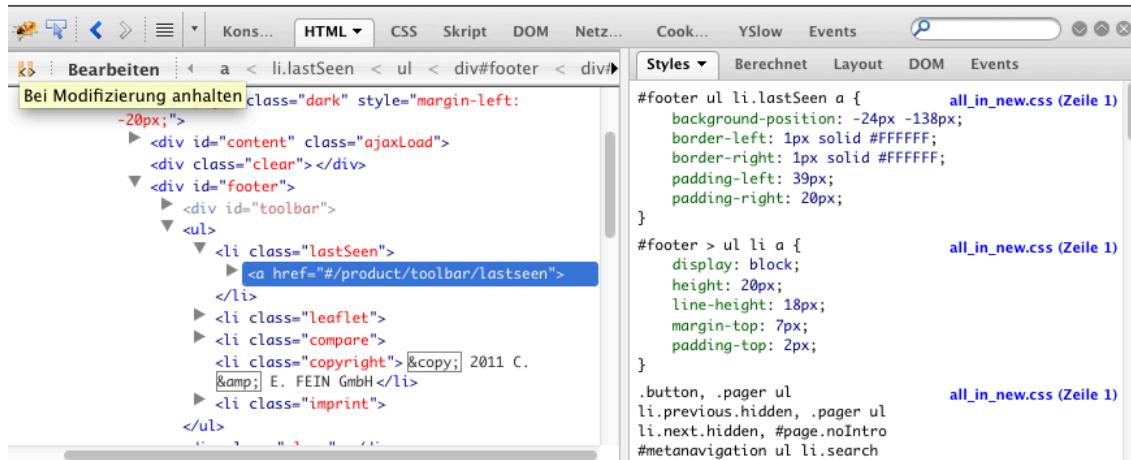


Abbildung 40: Firebug Übersicht

Firebug zeichnet auch die Zeit auf, wie lange eine Seite zum Laden braucht. Für diese Arbeit wurde sich aber nicht darauf verlassen, da so keine einheitlichen standortabhängigen Testergebnisse hätten erzielt werden können.

Mehr Informationen hier: <http://getfirebug.com/>

SpeedTracer

SpeedTracer ist eine Erweiterung für den Google Chrome-Browser und analysiert den Ladevorgang einer Webseite live und sehr detailliert. Möchte man beispielsweise erfahren, wie lange genau es dauert das HTML zu parsen, zeigt einem SpeedTracer solche Informationen.

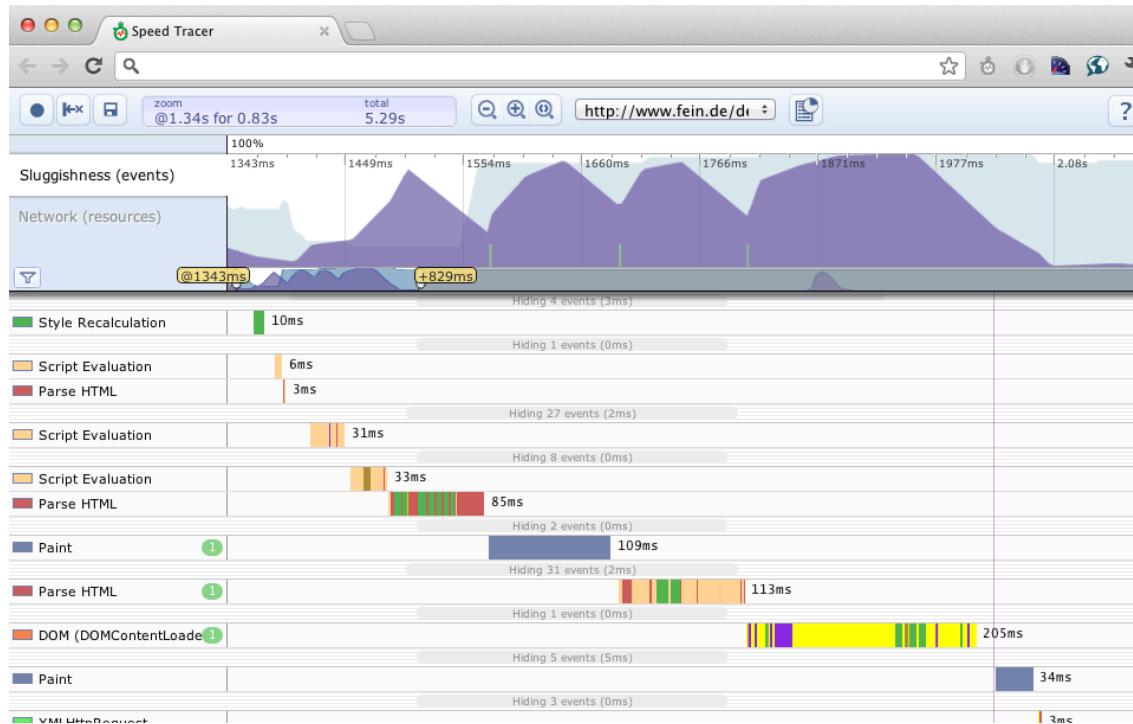


Abbildung 41: SpeedTracer Übersicht

Für die Installation ist es allerdings notwendig eine spezielle Entwicklerversion vom Chrome zu installieren.

Mehr Informationen hier:

<http://code.google.com/intl/de-DE/webtoolkit/speedtracer/index.html>

Anhang B: Inhalt der CD

Dieser Arbeit wurde eine Daten-CD beigelegt, auf dieser befinden sich folgende Dateien:

- Diese Arbeit im PDF-Format (Passwort: p3rform4nc3)
- Testergebnisse im *har*-Format, mit PCAP Web Performance Analyzer (s. Anhang A) anschaubar
- Auswertung der Testergebnisse im Excel-Format

Anhang C: Bildanteil der Top 10 Webseiten

Dieser Versuch wurde am 10.02.2012 von Willi Kampe durchgeführt. Die Liste mit den zehn meistbesuchten deutschen Webseiten wurde auf <http://www.alexa.com> recherchiert. Es wird untersucht, wie das Verhältnis von Bilddaten zum Gesamtvolumen der Webseite ist.

In den Top 10 von Alexa ist auf Platz 3 die Seite Google.com, da diese aber bereits auf Platz 1 in ihrer deutschen Variante untersucht wurde, ist sie für diesen Test nicht berücksichtigt worden. Dementsprechend sind alle anderen Seiten nachgerückt.

Tabelle 8: Top 10 Webseiten und deren Bildanteil

	total weight (KB)	image size (KB)	Anteil der Bilder am Gesamtvolumen
www.google.de	228,8	61,9	27,05%
www.facebook.com	261,0	52,0	19,92%
www.youtube.com	516,7	314,4	60,85%
www.ebay.de	448,2	339,5	75,75%
www.wikipedia.org	87,5	68,4	78,17%
www.amazon.de	384,6	275,9	71,74%
www.spiegel.de	1325,3	583,7	44,04%
www.yahoo.de	447,1	145,3	32,50%
www.bild.de	2075,3	1707,4	82,27%
www.web.de	394,3	284,4	72,13%

Im Mittel errechnet sich ein Bildanteil von 56,44%. Knapp über die Hälfte der Gesamtgröße dieser Webseiten besteht also aus Bildinformationen.

Glossar

CSS-Selektor: Als solcher wird der Teil einer CSS-Definition bezeichnet, welcher für die Selektion der betreffenden DOM-Elemente zuständig ist. Also einfach ausgedrückt, wird damit alles das bezeichnet, was vor der öffnenden geschweiften Klammer einer CSS-Definition steht.

Doctype: Die "Document Type Definition" oder DOCTYPE gibt die Struktur eines Dokumentes vor.

DOM (Document Object Model): Ist eine Spezifikation des W3C für ein Programmier-Interface, es erlaubt Programmen und Skripten den Inhalt, die Struktur und das Aussehen von HTML- und XML-Dokumenten zu verändern. Eine DOM Implementation in einem Web-Browser stellt Skripten zusätzlich noch grundlegende Funktionalitäten für den Zugriff auf Fenster oder die Historie zu Verfügung.

Eye Catcher: Als Eye Catcher bezeichnen die Grafiker Bilder, die eine positive Emotion in Verbindung mit der Marke auslösen sollen.

First View: Beschreibt in dieser Arbeit den ersten Seitenaufruf bei dem noch kein gefüllter Browser-Cache oder aufgelöster DNS-Hostname beim Client vorhanden. Dieser erste Aufruf ist im „Leben“ einer Webseite sehr wichtig. Ist es doch dieser erste Eindruck der sich beim Benutzer sehr stark meinungsbildend auswirkt.

Hashwert: Mit einer Hashfunktion wird aus einer unbestimmt großen Quellmenge an Daten eine kleine Zielmenge (Hashwert) mit definierter Größe erzeugt.

Open Source: Bedeutet so viel wie „quelloffen“ und beschreibt eine Bewegung der Softwareentwicklung, die es jedem Menschen ermöglicht, an einem Projekt teilzuhaben.

Relaunch: Ein Relaunch beschreibt das Ablösen eines Systems, was bisher verwendet wurde, durch ein Neues.

Repeat View: Beschreibt in dieser Arbeit das erneute Aufrufen einer Webseite, bei dem bereits Mechanismen wie der Browser- oder DNS-Cache ihre Wirkung zeigen. Dieser zweite Aufruf einer Seite ist meist deutlich schneller die First View, da der statische Content sich bereits im Cache befindet.

Round-Trip-Time: Sie gibt die Zeit an, die ein Datenpaket in einem Rechnernetz benötigt, um von der Quelle zum Ziel und zurück zu gelangen.

Literaturverzeichnis

Akamai (2011): Volume 4, Number 2 - The State of the Internet - 2ND Quarter, 2011 Report

Belshe, M. (2010): More Bandwidth Doesn't Matter (Much)

WWW: <http://www.belshe.com/2010/05/24/more-bandwidth-doesnt-matter-much/> (13.01.2012)

Brutlag, J. Google Inc. (2009): Speed Matters for Google Web Search

Abrufbar aus dem Internet: <http://code.google.com/speed/files/delayexp.pdf> (02.02.2012)

Chromium.org (2012)

WWW: <http://dev.chromium.org/spdy/spdy-whitepaper> (12.02.2012)

Dixon, P. (2009): Shopzilla's Site Redo - You Get What You Measure

WWW: <http://velocityconf.com/velocity2009/public/schedule/detail/7709> (12.12.2011)

Googleblog (2010): Using site speed in web search ranking

WWW: <http://googlewebmastercentral.blogspot.com/2010/04/using-site-speed-in-web-search-ranking.html> (02.01.2012)

Hyatt, D. (2000): Writing efficient CSS for use in the Mozilla UI

WWW: https://developer.mozilla.org/en/Writing_Efficient_CSS (12.02.2012)

ImmobilienScout24 (2012): Daten und Fakten zu ImmobilienScout24

Anrufbar aus dem Internet: http://www.static-immobilienscout24.de/MungoBlobs/2012/01/19/141611_012012.pdf (15.02.2012)

Lehmann, T., von Deetzen, A. & Kampe W. (2012). Persönliches Interview, geführt vom Verfasser bei ImmobilienScout24 in Berlin (18.01.2012).

Miller, R.B. (1968): Response time in man-computer conversational transactions

Abrufbar aus dem Internet: <http://theixdlibrary.com/pdf/Miller1968.pdf> (03.02.2012)

Souders, S. (2007): High Performance Web Sites

Theurer, T. (2007): Performance Research, Part 2: Browser Cache Usage – Exposed!

WWW: <http://www.yuiblog.com/blog/2007/01/04/performance-research-part-2/> (15.02.2012)

Theurer, T. (2007): Performance Research, Part 3: When the Cookie Crumbles

WWW: <http://www.yuiblog.com/blog/2007/03/01/performance-research-part-3/> (15.02.2012)

Theurer, T. (2007): Performance Research, Part 4: Maximizing Parallel Downloads in the Carpool Lane

WWW: <http://www.yuiblog.com/blog/2007/04/11/performance-research-part-4/>
(15.02.2012)

Wikipedia - Paketumlaufzeit (2012)

WWW: <http://de.wikipedia.org/wiki/Paketumlaufzeit> (31.01.2012)

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbstständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ort, Datum

Unterschrift