HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Herr
Eduard Sabot

Evaluierung von Strategien zur Verringerung der Ladezeiten CMS-generierter Webseiten

2019

Fakultät: Medien

BACHELORARBEIT

Evaluierung von Strategien zur Verringerung der Ladezeiten CMS-generierter Webseiten

Autor/in:
Herr Eduard Sabot

Studiengang:
Mobile Media

Seminargruppe:
MD14w1-B

Erstprüfer:
Prof. Dr.-Ing. Frank Zimmer

Zweitprüfer:
Dipl.-Ing. Birger Jesch

Einreichung:
Flöha, 24.06.2019

Faculty of Media

BACHELOR THESIS

Evaluation of strategies to reduce loading times of CMS-generated websites

author:
Mr. Eduard Sabot

course of studies:
Mobile Media

seminar group:
MD14w1-B

first examiner:
Prof. Dr.-Ing. Frank Zimmer

second examiner:
Dipl.-Ing. Birger Jesch

submission:
Flöha, 24.06.2019

Bibliografische Angaben

Nachname, Vorname: Sabot, Eduard

Thema der Bachelorarbeit

Evaluierung von Strategien zur Verringerung der Ladezeiten CMS-generierter Webseiten

Topic of thesis

Evaluation of strategies to reduce loading times of CMS-generated websites

47 Seiten, Hochschule Mittweida, University of Applied Sciences,
Fakultät Medien, Bachelorarbeit, 2019

Abstract

In dieser Arbeit werden ausgewählte Strategien erläutert, die potenziell zu einer Verringerung der Ladezeiten von Webseiten führen könnten. Des Weiteren werden diese an einer Test-Webseite angewandt, um zu untersuchen, welche Auswirkung sie auf die Ladezeiten haben. Hierfür wird das Performance-Tool des Browsers Google Chrome verwendet.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ASCII - American Standard Code for Information Interchange

CERN - Europäische Organisation für Kernforschung

CMS - Content Management System

CPU - Central Processing Unit

CSS - Cascading Style Sheets

CSSOM - CSS Object Model

DNS - Domain Name System

DOM - Document Object Model

GIF - Graphics Interchange Format

GPU - Graphics Processing Unit

HTML - Hypertext Markup Language

HTTPS - Hypertext Transfer Protocol Secure

IETF - Internet Engineering Task Force

kB - Kilobyte

MP3 - MPEG Audio Layer III

MPEG - Moving Picture Experts Group

ms - Millisekunden

PNG - Portable Network Graphics

RFC - Request for Comments

SEO - Search Engine Optimization

TCP - Transmission Control Protocol

TRCP/IP - Transmission Control Protocol/Internet Protocol

URL - Uniform Resource Locator

W3C - World Wide Web Consortium

WMV - Windows Media Video

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Das Nutzererlebnis im Zusammenhang mit einer Webseite setzt sich aus mehreren Aspekten zusammen. Ein Aspekt davon ist die Performance der Webseite. Diese kann beeinflussen, ob ein Nutzer auf der Seite verweilt oder abspringt. Dabei können nur wenige Sekunden entscheidend sein:

Tabelle 1: Absprungrate von Nutzer bei erhöhten Ladezeiten[1]

Ladezeit	Absprungrate der Nutzer
1-3 Sekunden	32%
1-5 Sekunden	90%
1-6 Sekunden	106%
1-10 Sekunden	123%

Ein positives Beispiel bietet der Modeshop Etam[2]. Sie beschleunigten ihre Ladezeiten um 0.7 Sekunden und konnten damit einen 28%igen Anstieg der Seitenaufrufe bewirken.

Ein weiteres Beispiel stammt von Google[3]. Sie erhöhten ihre Anzahl der Suchergebnisse auf 30 pro Seite, was einen Trafficverlust von rund 20% verursachte.

[1] Vgl. Think with Google, Find Out How You Stack Up to New Industry Benchmarks for Mobile Page Speed - https://think.storage.googleapis.com/docs/mobile-page-speed-new-industry-benchmarks.pdf Stand: Februar 2017

[2] Vgl. Guillaume Thibaux, ETAM's page speed load time reduced by 700 ms - https://blog.quanta.io/etam/ Aufruf: 05.06.2019

[3] Vgl. Greg Linden, Marissa Mayer at Web 2.0 - http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html Aufruf: 05.06.2019

1.1 Zielsetzung

Ziel dieser Arbeit ist die fachgerechte Einschätzung der ausgewählten Strategien zur Verringerung der Ladezeiten im Hinblick auf moderne Webseiten im Web. Mögliche Ergebnisse könnten die Ladezeiten von Webseiten um mehrere Sekunden verkürzen.

1.2 Abgrenzung des Themas

Bildkomprimierung sowie die Überarbeitung von CSS und JavaScript wird ausgeschlossen. Ebenso wird das Nutzererlebnis vor und nach den Tests nicht bewertet.

Die Ladezeiten, die im direkten Zusammenhang mit Hardwareressourcen stehen, werden nicht bewertet. Das heißt auch, dass auf den Einsatz von verbesserter Hardware verzichtet wird, um die Ladezeiten zu verkürzen. Das gilt für den verwendeten Webserver und für den Computer, auf dem die Untersuchungen stattfinden.

2 Grundlagen

2.1 HTTP

HTTP ist ein zustandloses Übertragungsprotokoll, das dazu dient, Multimedia-Dokumente bzw. Webseiten zwischen einem Webserver und einem Webbrowser zu übermitteln.

2.1.1 Hintergrund

Ab dem Jahr 1989 arbeitete der britische Physiker und Informatiker Tim Berners-Lee, der Begründer des Webs, an der Entwicklung des HTTP. Der damals am CERN beschäftigte Schweizer konnte mit HTTP 0.9 seine ersten Erfolge vorweisen. Diese Urversion war ein großer Erfolg für das damals noch junge Internet.[4]

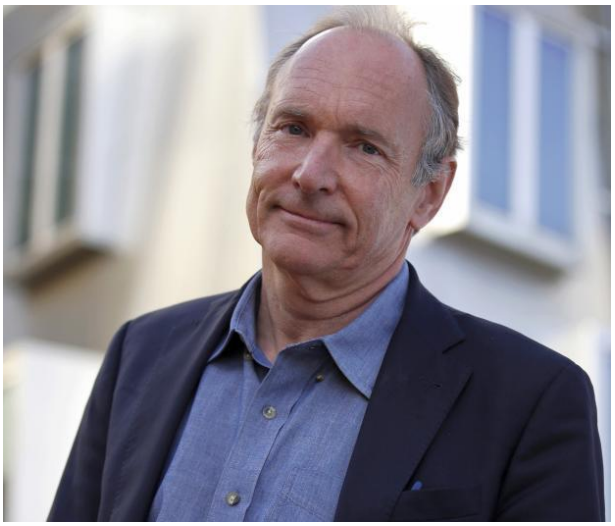

Abbildung 1: Tim Berners-Lee 2017

[4] Vgl. Munziger, Sir Tim Berners-Lee - http://www.munzinger.de/document/00000027854 Aufruf: 22.05.2019

Die Urversion besaß schon früh wesentliche Funktionalitäten, die noch heute das Internet von heute prägen. Zu diesen Funktionalitäten zählen:

- Anfrage-Antwort-Protokoll zwischen Client und Server
- Verwendung von ASCII-Zeichen für den Datenaustausch über TCP/IP-Verbindung
- Versendung von HTML-Dokumente durch Server
- Verbindungstrennung nach jeder Anfrage

2.1.2 Protokollversionen

HTTP/1.0 Durch die Publikation des RFC 1945[5] im Jahre 1996 wurde das ursprüngliche Protokoll zu HTTP/1.0 kodifiziert. Zu den Neuerungen zählen beispielsweise:

- Mehrere Zeilenumbrüche mit vorangestelltem Header[6] in einer Clientanfrage
- Vorangestellte Statusmeldung bei serverseitiger Antwort
- Serverseitige Antwort ermöglicht auch Headereinbindung mit Metainformationen
- Antwort nicht in HTML-Form erforderlich

Abschließend sollte noch erwähnt werden, dass zum jetzigen Stand jeder Server den Übertragungsstandard HTTP/1.0 besitzt.

HTTP/1.1 Im Jahre 1999 wurde mit der Publikation RFC 2616[7] der neue Standard für das HTTP beschlossen. Mit dieser Versionserweiterungen wurden viele Fehler der Vorversion behoben. Des Weiteren

[5] Siehe auch BERNERS-LEE, Request for Comments 1945 - https://tools.ietf.org/html/rfc1945

[6] Strukturierendes HTML-Element

[7] Siehe auch FIELDING, Request for Comments 2616 - https://tools.ietf.org/html/rfc2616

ergaben sich zahlreiche Leistungsverbesserungen, die die Usability[8] deutlich verbesserte. Zu den Leistungsverbesserungen zählen:

- Mögliche Aufrechterhaltung der Netzwerkverbindung
- Mehrere Anfragen und Antworten pro TCP-Verbindung
- Fortsetzung abgebrochener Verbindungen

HTTP/2 Erst im Mai 2015 wurde die neue Version beschlossen. Mit den Publikationen RFC 7540[9] und RFC 7541[10] wurde der Standard festgelegt. Zu den wichtigsten Antreibern der Weiterentwicklung zählen Google und Microsoft. HTTP/2 eröffnete eine Vielzahl von neuen Funktionen:

- Zusammenfassen mehrerer Anfragen
- Datenkompression wie beispielsweise Gzip[11]
- Binär kodierte Übertragung
- Push-Verfahren
- Streams[12] priorisieren

HTTP/3 Das von Google entwickelte HTTP-over-QUIC soll der Nachfolger von HTTP/2 werden.

[8] Benutzerfreundlichkeit bei Interaktion mit einem System

[9] Siehe auch BELSHE, Request for Comments 7540 - https://tools.ietf.org/html/rfc7540

[10] Siehe auch PEON & RUELLAN, Request for Comments 7541 - https://tools.ietf.org/html/rfc7541

[11] Kompressionsprogramm

[12] Bidirektionaler Datenstrom über eine bestehende TCP-Verbindung

2.1.3 Funktionsweise von HTTP

Alles beginnt mit der URL-Eingabe des Nutzers über einen Browser. Dieser Browser sendet zunächst eine Anfrage – auch Request genannt – an einen DNS-Server, um zu ermitteln, welche IP-Adresse der gesuchte Webserver besitzt. So eine Anfrage könnte folgendermaßen aussehen:

```
GET /index.php HTTP/1.1
Host: www.hs-mittweida.de
```

Abbildung 2: Anfrage an DNS-Server

Sobald dieser Prozess abgeschlossen ist, teilt der DNS-Server dem Browser die IP-Adresse des Webservers mit. Nun kann der Browser seine Anfrage an den zuständigen Webserver übermitteln. So eine Anfrage enthält mehrere Informationen. Zunächst enthält sie die Hauptinformationen wie die Host-Bezeichnung. Dazu enthält sie noch weitere Parameter wie die Bezeichnung des Browsers und die IP-Adresse.

Ist die Anfrage beim zuständigen Webserver angekommen, sucht er nach der angefragten Datei. Handelt es sich bei der Datei um eine statische Webseite – einer rein HTML-basierten Webseite – so dauert der Prozess nicht lange bis er sie gefunden hat. Bei dynamischen Webseiten bzw. CMS-generierten Webseiten dauert dieser Prozess etwas länger, da der Webserver zusätzliche Arbeit tätigen muss, um ein komplettes HTML-Dokument senden zu können. Vor dem Senden erstellt der Webserver aus der Datei oder aus den Dateien ein oder mehrere Pakete, die maximal 14 kB groß sind. Diese werden schließlich über das Internet an den Browser versendet.

Beim Browser angekommen geht er den HTML-Code schrittweise Zeile für Zeile durch und erstellt die Anzeige, die der Nutzer letztendlich auf seinem Bildschirm sieht. Falls der Browser auf Verweise auf Dateien stößt wie zum Beispiel Bilder, CSS- oder JavaScript-Dateien, so muss er erneut eine Anfrage an den zuständigen Webserver senden. Sobald der Browser alle notwendigen Informationen besitzt, um den vom Nutzer sichtbaren Bereich ohne Scrollen zu erstellen, beginnt

er die Webseite darzustellen. Diesen letzten Schritt bezeichnet man als Rendering.[13]

2.2 Content Management System

Ein Content Management System – kurz CMS – vereinfacht dem Anwender die Verwaltung und Erstellung einer Webseite auf Softwarebasis.

2.2.1 Funktionsweise von CMS

Ein CMS wird üblicherweise auf dem Server des jeweiligen Anbieters installiert. Die meisten Anbieter bieten diese Alternative der Webentwicklung in ihren Kunden-Portalen an. Die Voraussetzung für die meisten CMS ist eine aktive Verbindung zu einer MySQL-Datenbank, die aber auch in den meisten Fällen vom Anbieter zur Verfügung gestellt wird.

Ein großer Unterschied zu Webseiten, die ohne CMS erstellt werden, ist die Art der Generierung. Eine CMS-generierte Webseite wird dynamisch erstellt. Dynamisch erstellte Webseiten trennen strikt die inhaltlichen Informationen von den technischen Informationen. Das heißt, dass Bilder, Texte oder andere Medien erst beim Abruf einer Webseite anhand der Layout-Vorlagen, Programmierabhängigkeiten und der jeweiligen Skripte zusammengefügt und ausgegeben werden.

Zusammengefasst wird zunächst die Anfrage vom Webserver verarbeitet. Während er die Anfrage weiter an das CMS leitet, stellt er die technischen Informationen bereit. Auch statisch eingebundene Dateien wie beispielsweise Header-Bilder oder Icons werden vom Webserver bereitgestellt. Das CMS schickt in der Zwischenzeit eine Anfrage an die angebundene Datenbank, die die inhaltlichen

[13] Vgl. MEIER, GREGOR, Pagespeed Optimierung, Schritt für Schritt zur schnelleren Webseite, München, 2016 S. 13-17

Informationen weiterleitet. Sobald alle Daten beim Webserver eingetroffen sind, setzt er die Webseite zusammen und versendet sie an den anfragenden Browser.

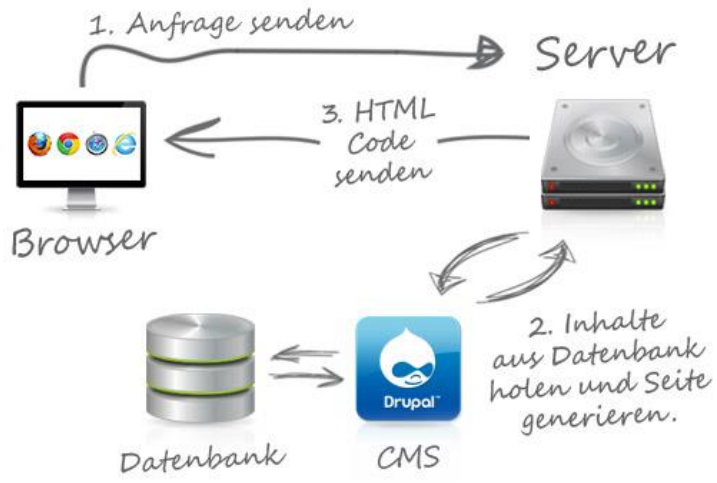

Abbildung 3:Dynamische Webseitenerstellung [Q10]

2.2.2 Vor- und Nachteile bei der Nutzung von CMS

Bei der Erstellung von Webseiten mithilfe von CMS gibt es verschiedene Vor- und Nachteile:

Vorteile:

- Einer der relevantesten Vorteile ist die mögliche Verwendung von Personen ohne Programmierkenntnisse. Durch die Trennung von Inhalt und Layout kann man Anwendungsmodule definieren. Sie setzen sie zusammen aus der Redaktion, der Entwicklung und die Veröffentlichung bzw. Administration. So können sich Experten verschiedener Bereiche zusammenfinden und ihre Stärken bündeln. Mithilfe von Rechtenverteilung fördern CMS diese Strukturen.
- Ein weiterer Vorteil der Trennung von Inhalt und Layout ist die Bearbeitung. Änderungen können viel schneller angewandt werden, weil der Inhalt sich dem Layout unterordnet und bestehen bleibt. Auch bei Änderungen am Layout wird der Inhalt sich unterordnen.
- Bei der Erstellung von Inhalten gibt es auch eine Vielzahl von Vorteilen. CMS bietet gute gestalterische Möglichkeiten an. So kann man bei der Texterstellung einen Texteditor verwenden oder Bilder über Bildmanager einbinden. Des Weiteren kann man einzelne Unterseiten besser voneinander trennen.

- Die meisten CMS bieten die Möglichkeit an Plug-Ins zu installieren, die dem Anwender die Anpassung eines CMS ermöglichen. Beispielsweise kann man voreingestellte Bildergalerien, Formulare oder ganze Shopsysteme erstellen und zu seiner Webseite hinzufügen.
- Letztendlich ist zu erwähnen, dass vor allem die Pflege von Webseiteninhalten die große Stärke eines CMS ist.

Aus den Vorteilen ergeben sich folgende Einsatzgebiete für CMS: Blogsysteme, Foren, Internetagenturen mit mehreren Webseiten, Nachrichtenseiten und Onlineshops

Nachteile:

- Ein wesentlicher Nachteil von CMS ist der hohe Aufwand der Ersteinrichtung. Dazu zählt auch die Installation des CMS, die Einrichtung beim Anbieter und die Anbindung zur Datenbank.
- Neben dem höheren Aufwand können auch die Hostingkosten höher sein. Jedoch gibt es auch Anbieter, die für die Einrichtung von CMS keine weiteren Kosten beanspruchen.
- Durch die Zusatzschritte wie Anfrage an CMS und Datenbank kann die Ladezeit von CMS-generierten Webseiten höher sein als bei statisch erstellten Webseiten

Hinsichtlich der Nachteile sollte man kein CMS verwenden, wenn man einen kleinen Internetauftritt plant, ein kleines Unternehmen repräsentiert oder man nur statische Informationen präsentiert.

2.3 Das Web Development Tool von Google Chrome

Für die Tests zur Ermittlung der einzelnen Ladezeiten wird das Web Development Tool von Google Chrome verwendet. Dieses Tool besitzt alle notwendigen Werkzeuge, die bei der Feststellung und der Interpretation der Ladezeiten benötigt werden. Zum einen wird das Sources-Tool verwendet, um festzustellen, welche Quellen die Test-Webseite verwendet. Solche Quellen sind beispielsweise CSS-Dateien, Schriftarten-Quellen, JS-Dateien oder ähnliches. Auch alle verwendeten Bilder sind dort gelistet.

Die Darstellung sieht wie folgt aus:

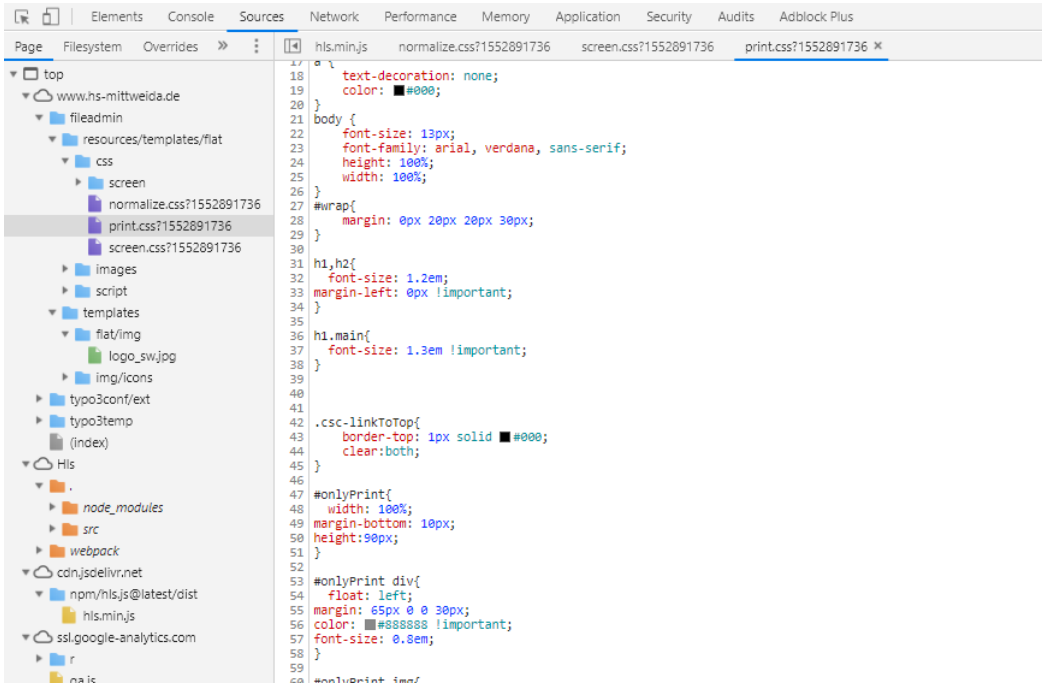

Abbildung 4: Beispiel Sources-Tool

Neben dem Sources-Tool wird primär das Performance-Tool verwendet. Dieses Tool zeigt, wie hoch die Ladezeit der Webseite ist. Dazu erhält man auch die Information, wie sich die Ladezeit zusammensetzt. Dafür unterteilt das Performance-Tool die Ladezeit in sechs verschiedene Kategorien: Loading, Scripting, Rendering, Painting, System bzw. Other und idle. Idle beschreibt die Zeit, die der Browser warten muss, bis die Prozesse an die CPU und GPU übergeben wurden und verarbeitet wurden. Aufgrund der wenigen Möglichkeiten Einfluss auf idle- und System-Zeiten nehmen zu können, wird darauf verzichtet diese auszuwerten.

Hier kurz die Erklärungen zu den vier Kategorien, die ausgewertet werden:

Loading: Das ist der Prozess, wo Daten heruntergeladen werden. Auch das Analysieren von Daten wird zum Loading kategorisiert.

Aufgabenbeispiele: Parse HTML, Receive Data, Finish Loading

Scripting: Das Scripting sorgt dafür, dass die verschiedenen Events bzw. Aufgaben ausgeführt werden.

Aufgabenbeispiele: domContentLoadedEventStart, requestStart, Compile Script

Rendering: Das Rendering erstellt das Layout und berechnet die Geometrie, die für das darauffolgende Painting benötigt wird.

Aufgabenbeispiele: Recalculate Style, Update Layer Tree, Layout

Painting: Beim Painting wird die Webseite anhand der berechneten Daten gezeichnet.

Aufgabenbeispiele: Paint, Composite Layers,

Um die Berechnung der vier Kategorien zu starten, öffnet man das Entwicklertool von Chrome über die Tastenkombination *Strg* + *Shift* + *I*. Dann öffnet man über den Reiter *Performance* das Performance-Tool. Mit der Tastenkombination *Strg* + *Shift* + *E* wird die Berechnung gestartet. Bei diesem Prozess wird die Seite neu geladen und parallel werden die Informationen gespeichert. Nachdem die Seite vollständig geladen wurde, kann man im unteren Bereich die einzelnen Kategorien einsehen und sich ein Überblick machen, wie viel Zeit benötigt wurde die verschiedenen Aufgaben auszuführen.

Die Zusammenfassung sieht folgendermaßen aus:

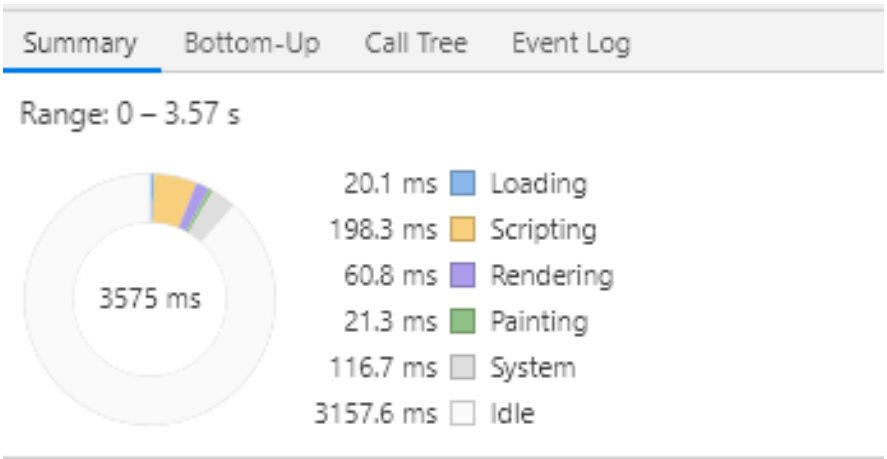

Abbildung 5: Beispiel einer Zusammenfassung im Performance-Tool

3 Strategien zur Verringerung der Ladezeiten

Im Nachfolgenden sind relevante Strategien aufgeführt, die Ladezeiten von Webseiten verringern können.

3.1 Bilder

Auch wenn sich der Trend wandelt und immer weniger Bilder auf Webseiten verwendet werden, sind Bilder immer noch eine der größten Datenquellen einer Webseite. Aufgrund dieser Tatsache sollte man sich gut überlegen, welches Format am geeignetsten ist.

3.1.1 Moderner Bildtyp - WebP

Zu den am meist verwendeten Bildtypen unserer Zeit zählen unumstritten JPEG[14]- und PNG-Dateien. JPEG wurde mit der Norm ISO/IEC 10918-1[15] im Jahre 1992 vorgestellt und PNG wurde vom IETF mit der RFC 2083[16] im Jahre 1996 verabschiedet. Trotz ihres Alters konnten sich bisher keine Alternativen der Datenkompressionen gegen diese beiden Methoden entscheidend durchsetzen.

Allerdings gibt es seit September 2010 eine neue Möglichkeit für Komprimierung – das Format WebP. Dieses Format wurde eigens von Google entwickelt. Im Vergleich zu JPEG schafft WebP eine höhere Bildqualität bei gleicher Datenmenge. Jedoch bietet diese Methode ausschließlich eine Farbunterabtastung von 4:2:0.[17]

> Eine Gegenüberstellung von JPEG- und WebP-Bildern findet man unter: *https://developers.google.com/speed/webp/gallery1*.

[14] Methode der Bildkompression

[15] Siehe auch RODRIGUEZ HERRERA, The JPEG standar - https://w3.ual.es/~vruiz/Docencia/Apuntes/Coding/Image/03-JPEG/index.html Aufruf: 27.05.2019

[16] Siehe auch BOUTELL, Request for Comments - https://tools.ietf.org/html/rfc2083 Stand: März 1997

[17] Vgl. ONLINE-UMWANDELN, Informationen zum WEBP Dateiformat - https://online-umwandeln.de/dateiformat/webp/ Stand: 21.04.2019

WebP-Dateien kann man über Online-Konverter wie *https://image.online-convert.com/convert-to-webp* oder *https://webp-converter.com/* erstellen. Eine weitere Alternative ist der offzielle Konverter von Google zum Download unter *https://developers.google.com/speed/webp/*.

Die nachfolgende Tabelle zeigt, welche Browser WebP vollständig unterstützen:

Tabelle 2: Browser, die WebP unterstützen[18]

Internet Explorer	**Microsoft Edge**	**Mozilla Firefox**	**Google Chrome**	**Apple Safari**	**Opera**
Keine Zulassung	Verwendung seit Edge 18 13. Nov. 2018	Verwendung seit Firefox 65 29. Jan. 2019	Verwendung seit Chrome 32 14. Jan. 2014	Keine Verwendung	Verwendung seit opera 19 28. Jan. 2014

Als Entwickler sollte man bei der Verwendung von WebP stets Fallback-Methoden verwenden, um sicherzustellen, dass die Verwendung von nichtunterstützenden Browsern keine Einschränkungen hinsichtlich der Usability besitzen.

[18] Vgl. Can I use? - https://caniuse.com/#feat=webp Stand: 23.05.2019

Hierfür gibt es drei Methoden:

```
<picture>
<source src=ihrbild.webp type=image/webp>
<source src=ihrbild.jpg type=image/jpg>
<img src=foo.png alt="Hier ALT-Text eingeben">
</picture>
```

Abbildung 6: Fallback - HTML5

```
#background-image: image("ihrbild.webp" format('webp'), "ihrbild.jpg");
```

Abbildung 7: Fallback - CSS mit leerem div-Container

```
<IfModule mod_rewrite.c>
RewriteEngine On
RewriteCond %{HTTP_ACCEPT} image/webp
RewriteCond %{DOCUMENT_ROOT}/$1.webp -f
RewriteRuke ^(path/to/your/images.+)\.(jpe?g|png)$ $1.webp
[T=image/webp,E=accept:1]
</IfModule>
<IfModule mod_headers.c>
Header append Vary Accept env=REDIRECT_accept
</IfModule>
AddType image/webp .webp
```

Abbildung 8: Fallback in der .htaccess-Datei

Letztere Methode ermöglicht dem Entwickler Bilder wie gewohnt als JPEG oder PNG einzubinden. Falls sich eine WebP-Datei mit identischer Bezeichnung im hinterlegten Pfad befindet, wird diese vorzugsweise verwendet.[19]

[19] Vgl. MEIER, Pagespeed Optimierung, S. 36-37

3.1.2 Bilder verzögert laden

Bei dieser Strategie wird der Anwender ein wenig ausgetrickst. Hier verwendet man ein Platzhalterbild. Dabei kann man zwischen zwei Methoden wählen: Entweder man verwendet ein eigenes Bild, das klein ist und von Grund auf eine geringe Ladezeit aufweist, oder man verwendet ein Bit-Code anstelle eines Bildes.

Diese Methoden sehen wie folgt aus:

```
<img src="platzhalter.jpg" data-src="echtes-bild.jpg">
```

Abbildung 9: Platzhalterbild

```
<img src=""
    date-src="echtes-bild.jpg">
```

Abbildung 10: Bit-Code als Bild

Der Platzhalter erfüllt die Aufgabe, dass er beim Durchlaufen des Quellcodes den Browser nur wenige Zeit kostet, um zur nächsten Zeile zu gelangen. Das hat den wesentlichen Vorteil, dass die gesamte Webseite schneller lädt.

Mit dem nachfolgenden JavaScript-Code, den man am Ende des Quellcodes vor den schließenden Body-Tag setzt, gibt man an, dass die Bilder, die eigentlich angezeigt werden sollen, nach dem Anzeigen der Webseite automatisch geladen werden sollen:[20]

```
<script>
    function init() {
        var imgDefer = document.getElementByTag('img');
        for (var i=0; i<imgDefer.length; i++) {
            if (imgDefer[i].getrAttribute('data-src')) {
                imgDefer[i].setAttribute('src',
                imgDefer[i].getrAttribute('data-src'));
            } } }
    window.onload = init;
</script>
```

Abbildung 11: JavaScript-Code zum verzögerten Laden

3.2 Datenkomprimierung

Datenkomprimierung ist ein wesentlicher Bestandteil, um Ladezeiten zu verringern. Das primäre Ziel einer Datenkomprimierung ist der Gewinn an Speicherplatz mithilfe von Codierung. Der Gewinn von Speicherplatz ermöglicht eine schnellere Verarbeitung, Übertragung und Darstellung der Daten. Der Grundsatz der Datenkomprimierung beschreibt, dass die alternative, neue Datei effizienter sein soll als die Ausgangsdatei.

Heutzutage finden die Methoden der Datenkomprimierung eine Vielzahl von Anwendungsgebieten:

- Rundfunk und Fernsehen insbesondere der Archivierung
- Mobilfunk
- Banken

[20] Vgl. MEIER, Pagespeed Optimierung, S. 44-45

- Mittelgroße und große Unternehmen
- Versicherungen
- Langzeitarchivierung medizinischer Daten
- Meteorologische und andere statistische Daten

In der Datenkomprimierung gibt es zwei verschiedene fundamentale Prinzipien: ***Redundanzreduktion*** und ***Irrelevanzreduktion***.

Bei der Redundanzreduktion versucht man schon vorhandene Informationen mithilfe von Deduktion[21] abzuleiten. Hier ein Beispiel:

```
Ausgangstext: AUCH EIN GROSSER BAUM IST EIN BAUM
Kodiertext: AUCH EIN GROSSER BAUM IST -4 -3
```

Abbildung 12: Beispiel Redundanzreduktion

In dem Beispiel werden die Wörter „EIN“ und „Baum“ referenziert, weil sie schon einmal im Text verwendet wurden. Somit konnte man einfach vier Zeichen einsparen.

Bei den Methoden der Redundanzreduktion wird vorausgesetzt, dass die Ausgangsinformation nicht verfälscht wird. Das bedeutet, wenn die Information komprimiert wird, muss sie anschließend auch verlustfrei rekonstruiert werden können. Deshalb verwendet man nur verlustfreie Datenkompressionstechniken. Bekannte verlustfreie Techniken sind: ZIP-Kompression, RAR-Kompression, PNG, GIF.[22]

Wiederrum bezieht sich die Irrelevanzreduktion auf Informationen, die mithilfe von verlustbehafteten Kompressionstechniken Verfälschungen hervorrufen, die akzeptiert werden. Beispielsweise wird ein Bild, dass 1600 Pixel breit ist auf

[21] Lateinisch: deductio - ‚Abführen, Fortführen, Ableiten‘

[22] Vgl. BILLO, Was ist Datenkompression - https://www.storage-insider.de/was-ist-datenkompression-a-764167/ Aufruf: 23.05.2019

einem Laptop ähnlich wahrgenommen, wie ein identisches Bild, das 600 Pixel breit ist und auf einem Smartphone angeschaut wird. Mit dieser Methode kann man bei kleineren Datenmengen vergleichbare Benutzerfreundlichkeit erzielen.[23]

Ein anderes Beispiel zeigt auch die Wichtigkeit von starken Verfälschungen: Falls ein Smartphone-Nutzer mit einer schlechten Internetverbindung ein Video auf einer Webseite anschauen möchte, akzeptiert er eine starke Verfälschung des Videos, um überhaupt ein Video angezeigt zu bekommen.

Die Irrelevanzreduktion wird vor allem bei Bildern, Audio-Dateien und Video-Dateien verwendet. Zu den bekanntesten verlustbehafteten Kompressionstechniken zählen: JPEG, MP3, MPEG-1, WMV.

Im Folgenden werden zwei Module beschrieben – Gzip und Deflate. Diese beiden Module werden vom Webserver verwendet, um eine Webseite zu komprimieren noch bevor sie versendet wird.

[23] Vgl. BILLO, Was ist Datenkompression? - https://www.storage-insider.de/was-ist-datenkompression-a-764167/ Aufruf: 23.05.2019

3.2.1 Gzip

Basierend auf dem LZ77-Algorithmus[24] komprimiert das Komprimierungsprogramm Gzip verlustfrei. Dabei werden doppelte Datenbestandteile analysiert und verwiesen.[25]

Voraussetzung für die Nutzung ist die Aktivierung des Webserver-Anbieters und das Einfügen der folgenden Code-Zeilen in die .htaccess-Datei:

```
<ifModule mod_gzip.c>
mod_gzip_on Yes
mod_gzip_dechunk Yes
mod_gzip_item_include file .(html?|txt|css|js|php|pl)$
mod_gzip_item_include handler ^cgi-script$
mod_gzip_item_include mime ^text/.*
mod_gzip_item_include mime ^application/x-javascript.*
mod_gzip_item_exclude mime ^image/.*
mod_gzip_item_exclude rspheader ^Content-Encoding:.*gzip.*
</ifModule>
```

Abbildung 13: Code für Gzip-Aktivierung

[24] Siehe auch MÜHLINGHAU, STEFAN, Der LZ77 Algorithmus - http://www.gm.fh-koeln.de/~hk/lehre/ala/ws0506/Praktikum/Projekt/D_rot/Ausarbeitung.pdf Aufruf: 25.05.2019

[25] Vgl. DEUTSCH, Request for Comments 1952 - https://tools.ietf.org/html/rfc1952 Stand: Mai 1996

3.2.2 Deflate

Das Modul Deflate wird in der Regel in Kombination mit dem Modul Gzip verwendet. Deflate verwendet neben dem LZ77-Algorithmus auch die Huffman-Kodierung[26].[27]

Wie auch bei Gzip wird bei der Nutzung vorausgesetzt, dass der Webserver Deflate aktiviert hat und der folgende Code in die htaccess-Datei hinzugefügt wird:

```
<IfModule mod_deflate.c>
AddOutputFilterByType DEFLATE text/plain
AddOutputFilterByType DEFLATE text/html
AddOutputFilterByType DEFLATE text/xml
AddOutputFilterByType DEFLATE text/css
AddOutputFilterByType DEFLATE application/xml
AddOutputFilterByType DEFLATE application/xhtml+xml
AddOutputFilterByType DEFLATE application/rss+xml
AddOutputFilterByType DEFLATE application/javascript
AddOutputFilterByType DEFLATE application/x-javascript
</IfModule>
```

Abbildung 14: Code für Deflate-Aktivierung

[26] Siehe auch HUFFMAN, DAVID, A Method for the Construction of Minimum-Redundancy Codes - http://compression.ru/download/articles/huff/huffman_1952_minimum-redundancy-codes.pdf Stand: 1952

[27] Vgl. DEUTSCH – Request for Comments 1951 - https://tools.ietf.org/html/rfc1951 Mai 1996

3.3 Kritische Redering-Pfad

Der kritische Rendering-Pfad umfasst das Rendering des Webseitenabschnitts, den der Anwender zu Beginn des Öffnens sieht. Diesen Bereich nennt man auch „above the fold“. Gerade für den Ersteindruck des Anwenders ist es wichtig, dass dieser Bereich schnellstmöglich geladen wird. Auch aus SEO-technischen Gründen empfiehlt es sich, diese Ladezeit so gut es geht zu verringern. Suchmaschinen wie Google priorisieren einen gutstrukturierten kritischen Rendering-Pfad.[28]

Abbildung 15: Above the fold [Q23]

3.3.1 Erstellung des kritischen Rendering-Pfads

Alle modernen Browser erstellen den kritischen Rendering-Pfad auf die gleiche Weise. Sie müssen zunächst eine DOM- und eine CSSOM-Baumstruktur erstellen. Das ist der Grund, weshalb HTML- und CSS-Elemente schnellstmöglich bereitgestellt werden müssen. Schrittweise sieht eine Baumstruktur folgendermaßen aus:

[28] Vgl. MEIER, Pagespeed Optimierung, S. 93

Tabelle 3: Einzelne Schritte zur Erstellung einer Baumstruktur

Bytes →	Characters →	Token →	Knoten →	DOM
3C 62 6F 64 79 3E 48 65 6C 6F 2C 20 3C 73 70 61 6E 3E 77 6F 72 6C 64 21 §C 2F 73 70 61 6E 3E 3C 2F 62 6F 64 79 3E	<html> <head>…<head> <body><p>Hello <web- performance …	StartTag: html StartTag: head … EndTag: head StartTag: body …	html head meta body p Hello	DOM- Baumstruktur CSSOM- Baumstruktur

Zunächst muss der Browser die Rohbytes erfassen und sie nach der vorgegebenen Dateicodierung, beispielsweise UTF-8, in Zeichen umwandeln. So entstehen Zeichenfolgen, die man auch als Characters bezeichnet. Diese muss er wiederrum in Tokens umwandeln. Tokens müssen für den Browser eindeutig sein. Die Standardisierung für HTML5 gibt das W3C vor.[29]

Aus den vorliegenden Tokens werden nun Objekte bzw. Knoten erstellt, die alle wichtigen Eigenschaften und Regeln beinhalten. Dieser Prozess wird auch als Lexing bezeichnet. Abschließend müssen alle Knoten noch zu einer Baumstruktur zusammengefasst werden.[30]

Nachdem die beiden Baumstrukturen erstellt sind, erfolgt die Zusammenführung zur Rendering-Baumstruktur, die alle notwendigen Informationen beinhaltet. Dabei werden alle sichtbaren Knoten der DOM-Baumstruktur übernommen. Für

[29] Siehe auch W3C, W3C Recommendation, https://www.w3.org/TR/html5/ Stand: 14. Dezember 2017

[30] Vgl. Google, Objektmodell erstellen, https://developers.google.com/web/fundamentals/performance/critical-rendering-path/constructing-the-object-model?hl=de Aufruf: 01.05.2019

jeden dieser Knoten werden identische Knoten in der CSSOM-Baumstruktur gesucht und die zugewiesenen Regeln übernommen.[31]

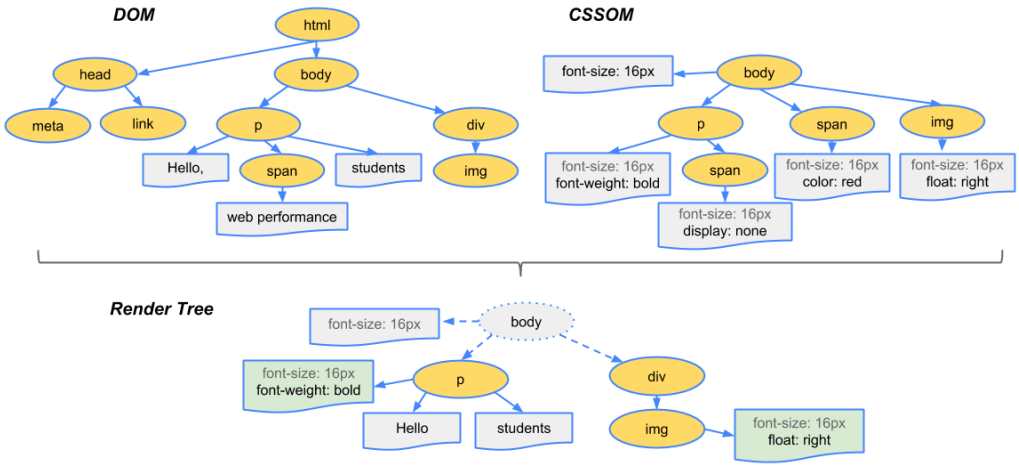

Abbildung 16: Zusammenführung zur Rendering-Baumstruktur [Q24]

Bevor die Darstellung auf dem Gerät erscheint, benötigt der Browser das Layout der einzelnen Knoten. Für das Layout gibt es vordefinierte Größen, die vom W3C-Consortium standardisiert wurden, falls sie nicht explizit vom Entwickler definiert wurden. Mit dem Layout kann der Browser letztendlich die Geometrie der Knoten ermitteln.[32]

Schlussfolgernd kann man die Ladezeit des kritischen Rendering-Pfads am besten verringern, wenn man die notwendigen Ausgansinformationen für die Baumstrukturen minimiert und die nichtkritischen Informationen aufschiebt.

3.3.2 CSS unterteilen

Ein erster Schritt ist die CSS-Dateien in kritischen und nichtkritischen Abschnitten zu unterteilen. Dafür gibt es ein Online-Tool namens *Critical Path CSS Generator* von Jonas Sebastian Ohlsson. Dieses Tool filtert die nichtkritischen Abschnitte

[31] Vgl. Google, Erstellung der Rendering-Baumstruktur, Layout, und Paint - https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-tree-construction?hl=de Stand: 27.04.2019

[32] Vgl. W3schools, HTML Layouts - https://www.w3schools.com/html/html_layout.asp Stand: 29.04.2019

eines eingefügten CSS-Quellcodes. Den gefilterten Code muss lediglich sichern und in den Header-Bereich eingebunden werden. Abschließend muss man noch den Ausgangscode vor dem schließenden Body-Tag in die index.php einbinden.[33]

Abbildung 17: Critical Path CSS Generator [Q27]

3.3.3 JavaScript asynchron laden

JavaScript (kurz: JS) ist eine dynamische Scriptsprache. Gerade in der Werbentwicklung ist JS mittlerweile unverzichtbar geworden. Mithilfe von JS kann man dynamische Inhalte erstellen, mit denen der Nutzer interagieren kann. Beispiele für JS-Inhalte sind:[34]

- Verarbeiten von Formularen
- Mausaktionen
- Dialogfenster

[33] Vgl. OHLSSON, JONAS SEBASTIAN, Critical Path CSS Generator, https://jonassebastianohlsson.com/criticalpathcssgenerator/#what-is Stand: 24.04.2019

[34] Vgl. Mozilla, JavaScript-Grundlagen - https://developer.mozilla.org/de/docs/Learn/Getting_started_with_the_web/JavaScript_basis Aufruf 02.05.2019

- Animierte Bilder
- Galerien

JavaScript besitzt die Fähigkeit die DOM- und die CSSOM-Baumstruktur abzufragen und zu ändern. Diese Fähigkeit hat den entscheidenden Nachteil, dass das Ausführen dieser Strukturen blockiert werden kann. Dabei ist die Position, also wo JavaScript eingebunden wird, ausschlaggebend. Geht der Browser den Quellcode durch, unterbricht er die DOM-Erstellung bei jeden Script-Tag, den er einliest. Erst sobald die Ausführung abgeschlossen ist, würde die Erstellung fortgeführt werden.

Ein weiterer wichtiger Punkt ist der Zeitpunkt, wann die Ausführung von JavaScript überhaupt erfolgen kann. Diese hat nämlich die Beendigung der Ausführung der CSSOM-Baumstruktur als Voraussetzung. Anhand der Abhängigkeiten von DOM-, CSSOM-Baumstrukturen und JS kann schlussfolgern, dass man JS asynchron laden sollte, um einen schnellen kritischen Rendering-Pfad zu gewährleisten.

Um JS asynchron zu laden muss man lediglich das Schlüsselwort *async* zum Script-Tag hinzufügen. Dieses Schlüsselwort gibt dem Browser an, dass die DOM-Erstellung nicht unterbrochen werden soll und dass das eingebundene JS nachrangig geladen werden soll. Da die CSSOM-Baumstruktur im Vornherein vorrangig gegenüber dem JS ist, hat das keine Folgen.[35]

[35] Vgl. Google, Interaktivität dank JavaScript - https://developers.google.com/web/fundamentals/performance/critical-rendering-path/adding-interactivity-with-javascript?hl=de Aufruf: 02.05.2019

Das Vorgehen würde wie folgt aussehen:

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <link href="style.css" rel="stylesheet">
    <title>Kritischer Rendering-Pfad: JS async</title>
  </head>
  <body>
    <p>Ladezeiten verringern <span>jetzt</span></p>
    <div><img src="foto1.jpg"></div>
    <script src="js.js" async></script>
  </body>
</html>
```

Abbildung 18: JS asynchron laden

3.4 Der Zwischenspeicher im Browser – Cache

Der Cache ist ein schneller Zwischenspeicher (auch Puffer-Speicher genannt), der dem Browser ermöglicht Informationen wie Bilder, Text und JS-Zeilen abzuspeichern und diese wieder abzurufen, wenn der Nutzer die Webseite erneut aufruft. Durch diese Methode kann man je nach Webseite die Ladezeit deutlich senken.

Der Nutzer selbst hat die Möglichkeit das Verhalten des Cache zu bestimmen. Er kann beispielsweise festlegen, wie groß der verwendete Gesamtspeicher des Browsers sein soll, ob Informationen überhaupt zwischengespeichert werden sollen oder auch die Dauer bestimmen, wie lange Informationen abgespeichert werden dürfen. Darüber hinaus kann der Nutzer den Cache jederzeit löschen.

Informationen von der Webseite werden erst bereitgestellt, wenn der Entwickler entsprechende Codezeilen in die htaccess-Datei hinzufügt. Dabei hat er auch die Möglichkeit anzugeben, welcher Typ von Informationen verwendet werden darf. Auch dem Entwickler ist vorbehalten anzugeben, wie lang die Speicherung erfolgen darf.

Der benötigte Code mit allen Arten von Typen sieht wie folgt aus:

```
<IfModule mod_expires.c>
ExpiresActive ON
ExpiresDefault "access plus 1 month"
ExpiresByType text/html "access plus 1 month"
ExpiresByType image/gif "access plus 1 month"
ExpiresByType image/jpeg "access plus 1 month"
ExpiresByType image/png "access plus 1 month"
ExpiresByType text/css "access plus 1 month"
ExpiresByType text/javascript "access plus 1 month"
ExpiresByType application/x-javascript "access plus 1 month"
ExpiresByType text/xml "access plus 1 seconds"
</IfModule>
```

Abbildung 19: Code für Cache-Aktivierung

Bei diesem Code wird beschrieben, dass alle HTML-, CSS-, JS- und XML-Dateien vom Typ Text gespeichert werden soll. Des Weiteren sollen auch Bilder vom Typ GIF, JPEG und PNG gespeichert werden. Diese Information haben eine Laufzeit von einem Monat. Zusätzlich werden auch JS-Applikationen gespeichert.

Ebenfalls werden Informationen zwischengespeichert, die keinen dieser Typen entsprechen, aber vom Browser als wichtig oder zeitintensiv eingestuft werden.

3.5 Einstellungen im Joomla!

Es gibt einige Methoden, die im CMS Joomla! dafür sorgen, dass die Ladezeiten verringert werden. Wie in anderen CMS gibt es auch in Joomla! die Möglichkeit Erweiterungen von Drittanbietern zu installieren, die versprechen die Ladezeiten positiv zu beeinflussen. Hier werden jedoch nur die Einstellungen präsentiert, die man im Joomla! ohne Erweiterungen übernehmen kann.

3.5.1 Caching aktivieren

Joomla! bietet die Möglichkeit die Informationen, die nur vom CMS bereitgestellt werden, vom Nutzer zwischengespeichert werden kann. Das können beispielsweise die in Joomla! erstellten Module sein oder ähnliches.

Um das Caching zu aktivieren, muss man über den Reiter *System* zum Untermenü *Konfiguration* navigieren. In der Konfiguration navigiert man über den Reiter *System* und aktiviert dort das *Erweiterte Caching*. Abschließend muss man die Einstellung abspeichern.

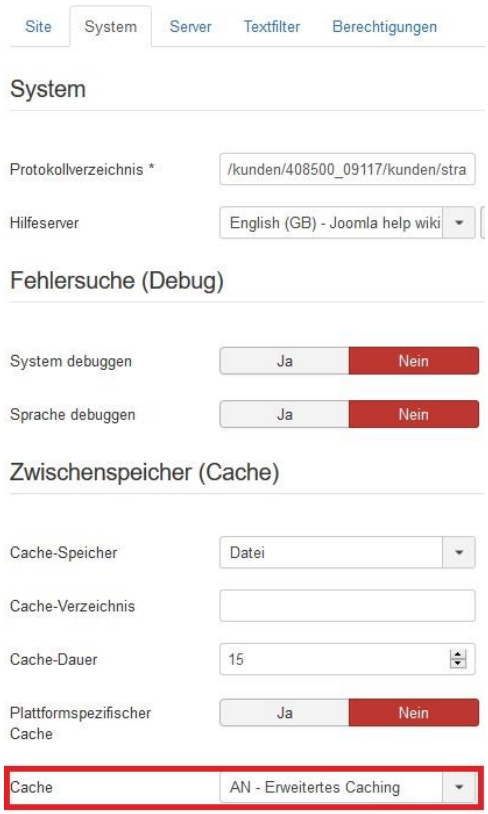

Abbildung 20: Joomla-Cache aktivieren

Weiter gibt es die Möglichkeit den Browser-Cache zu aktivieren. Das ermöglicht, dass die Webseite den vorliegenden Cache überprüft und gegebenenfalls eine Datei davon verwendet. Um Browser-Caching zu ermöglichen, muss man das von Joomla! bereitgestellte Plugin aktivieren. Dafür muss man über den Reiter *Erweiterungen* zum Menüpunkt *Plugins* navigieren. Dort sind verschiedene Plugins aufgelistet. Darunter auch das *Plugin System – Systemcache*. Sobald man auf das Plugin klickt, kann man den Status auf *aktiviert* setzen und dazu

noch bei dem Punkt *Browser-Cache benutzen* die Schaltfläche *Ja* aktivieren. Abschließend muss man die Einstellungen speichern.

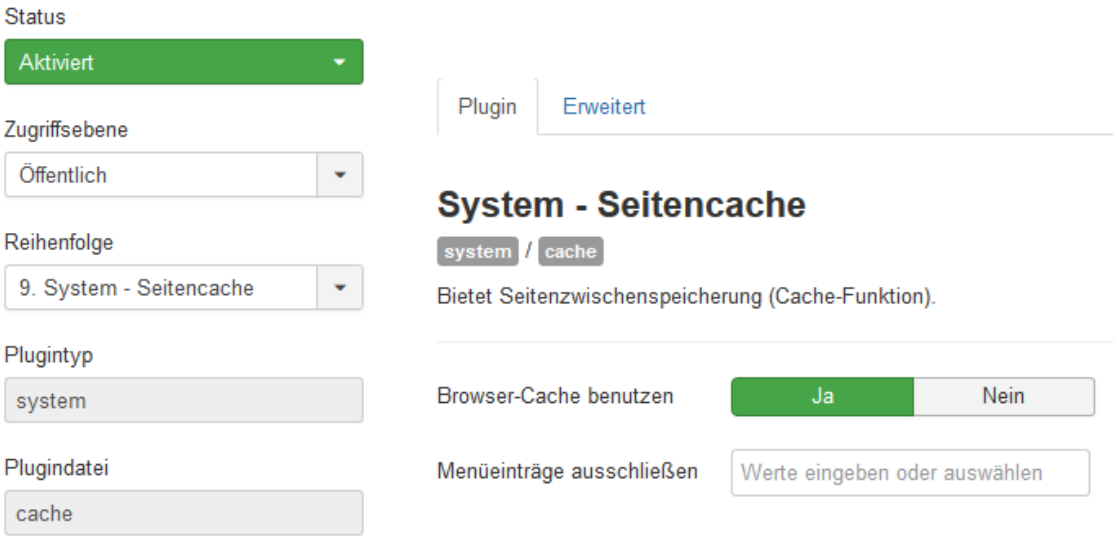

Abbildung 21: Joomla! Browser-Cache aktivieren

3.5.2 Gzip aktivieren

Bei einem Webserver, der Gzip aktiviert hat, werden nur die Daten komprimiert, die sich auch tatsächlich auf dem Webserver befinden. Die Daten, die vom CMS bereitgestellt werden wie beispielsweise Module oder Galerien, werden nicht von ihm komprimiert. Jedoch kann man in Joomla! einstellen, dass die Daten bevor sie versendet werden ebenfalls komprimiert werden. Die Aktivierung tätigt man ebenfalls im Untermenü *Konfiguration* über den Reiter *System*. Innerhalb des Konfigurationsmenü navigiert man zum Reiter *Server*, wo man die Gzip-Konfiguration aktivieren kann. Letztendlich muss man die neuen Einstellungen speichern.

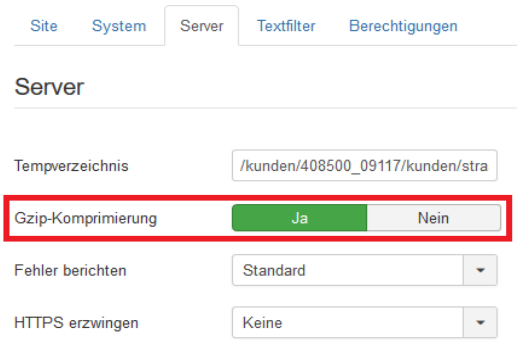

Abbildung 22: Joomla! Gzip-Aktivierung

3.6 HTTPS

HTTPS ist wie HTTP ein Kommunikationsprotokoll. Mithilfe dieses Protokolls werden sensible Informationen wie beispielsweise Benutzername/Passwort, Bankdaten oder persönliche Daten verschlüsselt empfangen und gesendet.

3.6.1 Unterschied zu HTTP

HTTPS besitzt folgende Unterschiede zu HTTP:

Tabelle 4: Unterschied zwischen HTTP und HTTPS[36]

HTTP	HTTPS
▪ Verwendet den Port 80	▪ Verwendet den Port 443
▪ Arbeitet auf der Anwendungsschicht	▪ Verwendet den TLS[37]
▪ Benötigt kein Zertifikat	▪ Benötigt ein SSL-Zertifikat, das von einer Zertifizierungsstelle validiert wurde
▪ Keine Verschlüsselung	▪ Verschlüsselung vor dem Senden

[36] Vgl. Media-Company, HTTPS und SSL Verschlüsselung – Was ist das und warum ist das für Webseiten unverzichtbar? - https://www.media-company.eu/blog/allgemein/https-und-ssl-verschluesselung/ Aufruf: 02.06.2019

[37] Siehe auch SECURITRY-INSIDER, Was ist TLS? - https://www.security-insider.de/was-ist-tls-transport-layer-security-a-673066/ Aufruf: 02.06.2019

3.6.2 Vor- und Nachteile von HTTPS

HTTPS hat folgende Vor- und Nachteile:

Vorteile:

- Schutz vor Hacker-Angriffen
- Erhöhung des Benutzervertrauens
- Verbesserte Grundlage beim Ranking von Google
- Erhöhung der Sicherheit bei der Nutzung von mobilen Endgeräten

Nachteile:

- Erwerb eines SSL-Zertifikats ist mit Kosten verbunden
- Mögliche Probleme beim Umstellungsprozess von HTTP zu HTTPS
- Kompatibilitätsprobleme aufgrund der noch jungen Technologie
- Erhöhung der Ausführungszeit des Servers

3.6.3 HTTPS nicht erzwingen

Um HTTPS nicht zu erzwingen, muss man zwei Einstellungen ändern. Die erste Einstellung befindet sich in Joomla! im Konfigurationsmenü unter *System* und dem Untermenü *Konfiguration*. Dort finden man unter dem Reiter *Server* den Einstellungspunkt *HTTPS erzwingen*. Diesen setzt man auf *Keine.*

Die zweite Einstellung befindet sich auf dem Server in der Datei configuration.php. In dieser Datei verändert man den Wert von *public $force_ssl* auf 0.

4 Rahmenbedingungen der Untersuchungen

Im Nachfolgenden werden die Strategien zur Verringerung der Ladezeiten anhand einer Test-Webseite angewandt und untersucht, um festzustellen, welchen Nutzen sie im Vergleich haben.

Zunächst wird die Test-Webseite im Hinblick auf die technischen Eigenschaften vorgestellt. Anschließend wird die Vorgehensweise erläutert.

4.1 Test-Webseite

Die Test-Webseite ist die Webseite vom Höhenservice Straß GmbH (*https://strass-hoehenservice-test.kserver-2.de*) und wurde über das CMS Joomla! erstellt.

Sie besteht aus den folgenden Datenquellen:

Tabelle 5: Alle Datenquellen der Test-Webseite

Bilder	▪ glasdach.jpg (292 KB) ▪ header.jpg (167 KB) ▪ laden.jpg (79 KB) ▪ schulung.jpg (68 KB) ▪ siegel.png (26 KB) ▪ viadukt.jpg (180 KB) ▪ arbeitsschutz.jpg (68 KB) ▪ dienstleistungen.jpg (66 KB) ▪ konzepte.jpg (40 KB) ▪ projekte.jpg (50 KB) ▪ pruefung.jpg (67 KB) ▪ qualifikationen.jpg (67 KB)
CSS-Dateien	▪ styles.css (12 KB)

	▪ wk-base.min.css (25 KB)
JS-Dateien	▪ drei jQuery-Dateien (jeweils <10 KB) ▪ caption.js (1 KB) ▪ scripts.js (2 KB) ▪ wk-base.js (5 KB)
Schriftarten	▪ https://fonts.googleapis.com/css?family=Montserrat:300,400,500,700

Anzumerken ist, dass die Test-Webseite bereits eine kurze Ladezeit besitzt, da an vielen Stellen mit Best Practice Methoden gearbeitet wurde. Damit ist die Test-Webseite gut geeignet, um die Ergebnisse der Untersuchung stellvertretend für moderne, professionelle Seiten im Internet zu interpretieren.

4.2 Ablauf der Untersuchungen

Zu aller erst wird eine Ausgangstest getätigt, um zu ermitteln, welche Ladezeiten die einzelnen Abschnitte (sh. Kapitel 2.3) für einen kompletten Seitenaufruf der Test-Webseite benötigen.

Anschließend werden die einzelnen Strategien zur Verringerung der Ladezeiten (sh. Kapitel 3) nach einer bestimmten Reihenfolge (sh. unten) angewandt. Nach jeder einzelnen Anwendung werden die Untersuchungsergebnisse mit den Untersuchungsergebnissen verglichen, die die kürzesten Ladezeiten aufweisen. Das heißt ebenfalls, dass die Anwendungen, die nicht wie erwartet positive Auswirkungen auf die Ladezeiten haben, wieder entfernt werden.

Die Reihenfolge der Test ist:

1. Einstellungen im Joomla!
2. Bilder
3. Kritischer Rendering-Pfad
4. Datenkomprimierung
5. Der Zwischenspeicher im Browser – Cache
6. HTTPS

Alle Tests werden acht Mal ausgeführt. Das ist notwendig, um ein genaueres Ergebnis zu erzielen, da aufgrund von Latenzen, Auslastung der Server oder anderen Ursachen Verfälschungen entstehen könnten. Nachdem alle Testdurchläufe ausgeführt wurden, werden sie zusammengefasst und grafisch dargestellt.

Abschließend zu den Untersuchungen wird ein Schlussfazit erstellt, um die Strategien gemeinschaftlich zu bewerten.

5 Evaluierung der Strategien

In diesem Kapitel werden die Strategien zur Verringerung der Ladezeiten an der Test-Webseite angewandt.

5.1 Ausgangstest

Die Test-Webseite hat folgende Untersuchungsergebnisse vor dem Einsatz der Strategien zur Verringerung der Ladezeiten:

Tabelle 6: Ausgangstest

Ausgangstest	Loading	Scripting	Rendering	Painting
1	9,9	41	32,8	3,4
2	10,7	39,4	31,3	3,3
3	11,4	42,3	33,6	3,6
4	9,9	37	31,3	3,2
5	11,8	43,6	31,6	3
6	11,3	40,8	40,3	3,2
7	11	42,5	29,8	3,1
8	9,7	39,3	31,5	3,4
Durchschnitt	**10,7125**	**40,7375**	**32,775**	**3,275**

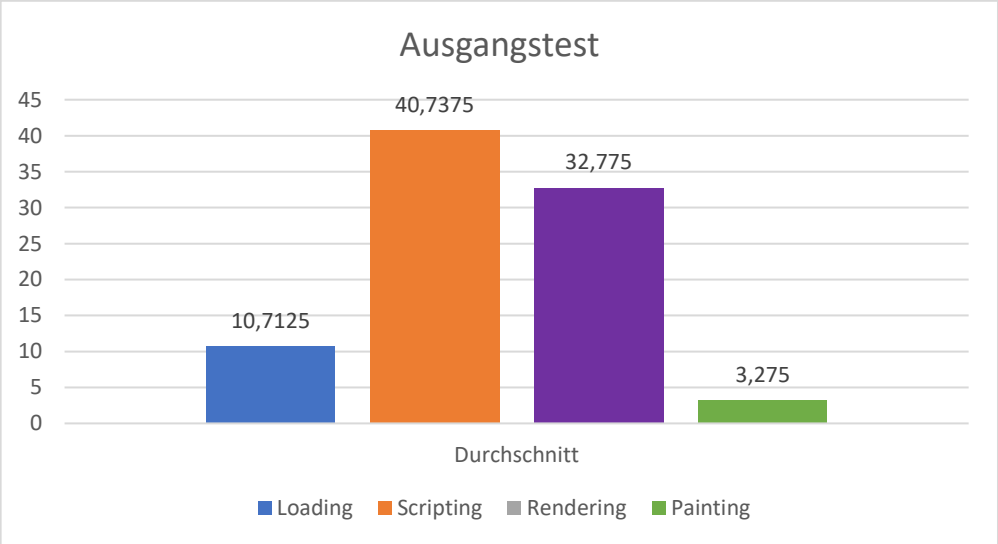

Abbildung 23: Ausgangstest

Gesamtladezeit: 87,5ms

5.2 Anwendung: Einstellungen in Joomla!

Nach den erfolgten Einstellungen, die im Kapitel 3.5 erläutert wurden, ergaben sich folgende Untersuchungsergebnisse:

Tabelle 7: Anwendung: Einstellungen in Joomla!

Einstellungen in Joomla!	Loading	Scripting	Rendering	Painting
1	11	39,3	35,7	3,6
2	11	39,4	31,3	3,2
3	11,4	42,5	33,6	3,1
4	10,6	39,4	31,9	3,1
5	10,3	39,1	32,6	3,6
6	10,5	40	31,1	3
7	10,6	43,2	32,8	3,2
8	10,1	39,1	29,5	3,3
Durchschnitt	**10,6875**	**40,25**	**32,3125**	**3,2625**

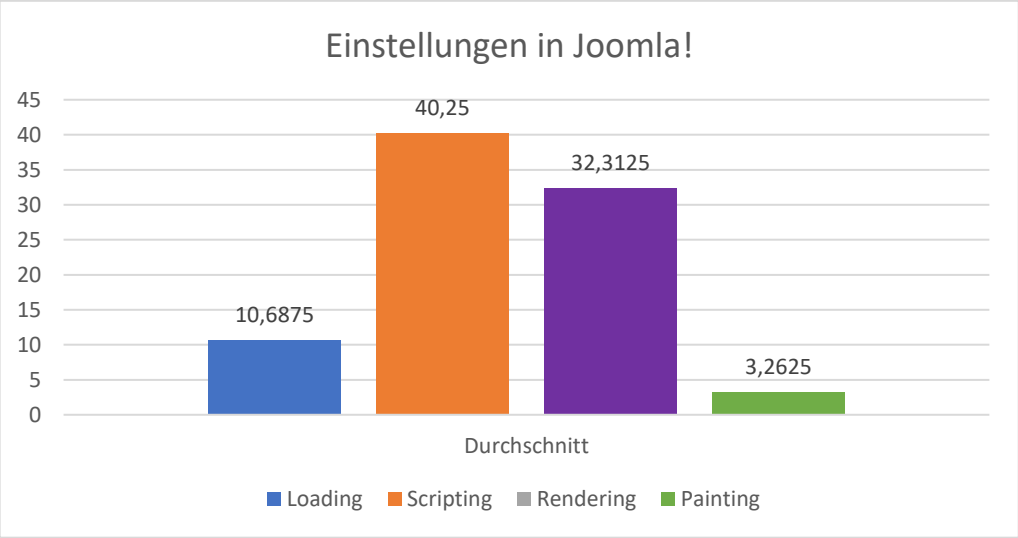

Abbildung 24: Anwendung: Einstellungen in Joomla!

Gesamtladezeit: 86,5125ms

Die Ladezeitverkürzung beträgt 0,9875ms. Damit wurde eine Ladezeitverkürzung von rund 1,13% erreicht. Speziell in der Kategorie Scripting und Rendering wurde die Zeit ca. um 0,5ms gesenkt.

Angesicht des wenigen Aufwandes, den man unternehmen muss, kann man festhalten, dass diese Strategie in jedem Fall verwendet werden sollte. Gerade bei

größeren Seiten, wo mehr Informationen komprimiert oder zwischengespeichert werden können, können diese Einstellungen weitaus bessere Ergebnisse liefern.

5.3 Anwendung: Bilder

Beim Einsatz von WebP-Bildern im Zusammenhang mit dem verzögerten Laden kann die Interpretation der Ergebnisse aufgrund der verschiedenen Ansatzpunkte nur schwer erfolgen. Aus diesem Grund werden die zwei Methoden jeweils getrennt und gemeinsam angewandt.

5.3.1 Anwendung: WebP-Bilder

Hier die Ergebnisse der Tests mit dem Einsatz von WebP-Bildern:

Tabelle 8: Einsatz von WebP-Bildern

Bilder (WebP)	Loading	Scripting	Rendering	Painting
1	9,6	41,2	34,1	4,7
2	9,6	46,7	32,4	4,8
3	12	40,4	30,7	7,1
4	10,3	38,3	31,9	5,2
5	9,8	38,2	31,3	6,8
6	9,9	37,3	30,9	7
7	10,5	40	30,4	5,7
8	10,5	40	30,4	5,7
Durchschnitt	**10,275**	**40,2625**	**31,5125**	**5,875**

Gesamtladezeit: 87,925ms

Im Vergleich zum Einsatz von JPG-Bildern erhöht sich die Ladezeit um 1,4125ms. Auffällig ist, dass vor allem das Painting mehr Zeit benötigt. Positive Auswirkungen hatte das WebP-Format aber beim Loading und beim Rendering. Zurückführen kann man das auf den geringeren Speicherplatz, den das WebP-Format benötigt, um Bilder der gleichen Qualität zu sichern.

5.3.2 Anwendung: Verzögertes Laden

Hier die Ergebnisse zum verzögerten Laden:

Tabelle 9: Einsatz von verzögerten Laden

Bilder (Verzögertes Laden)	Loading	Scripting	Rendering	Painting
1	9,2	42,9	32,9	5,4
2	9,7	43,4	32,4	4,9
3	9,7	46,1	32,4	4,9
4	10,6	42,6	31,6	5,4
5	10,6	41,9	33,1	5,3
6	9,9	43,2	34,3	4,8
7	9,4	42,1	32,8	5,1
8	9,9	41,9	32,5	5,2
Durchschnitt	**9,875**	**43,0125**	**32,75**	**5,125**

Gesamtladezeit: 90,7625

Auch beim verzögerten Laden kann man keine positiven Auswirkungen erkennen. Vor allem das Scripting erhöht aufgrund der Ausführung von acht JS-Anweisungen um 2,7625ms. Das Painting benötigt auch mehr Zeit. Das kann man damit begründen, dass acht weitere Bilder gezeichnet werden müssen. Das Loading und das Rendering blieben wie erwartet ungefähr gleich.

5.3.3 Anwendung beider Strategien

Beim Einsatz von beiden Strategien wurde für den Platzhalter ebenfalls ein WebP-Format verwendet, um die einheitliche Verwendung der Bildformate zu gewährleisten. Das sieht wie folgt aus:

```
<img src="placeholder.jpg" data-src="images/viadukt.jpg"/>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<img src="placeholder.webp" data-src="images/viadukt.webp"/>
```

Abbildung 25: Ausführung beider Strategien

Die Untersuchungen ergaben folgende Ergebnisse bei der Anwendung von beiden Strategien:

Tabelle 10: Anwendung: Bilder

Bilder	Loading	Scripting	Rendering	Painting
1	9,8	43,3	31,1	6,2
2	11,2	42,5	31,9	5,1
3	10,2	44,3	32,4	5,4
4	10,5	39,1	33,6	4
5	10,2	41	34,2	5,1
6	10,6	44,1	33,1	5
7	10,6	41,5	31,4	4,6
8	9,4	39,1	34,2	4,4
Durchschnitt	**10,3125**	**41,8625**	**32,7375**	**4,975**

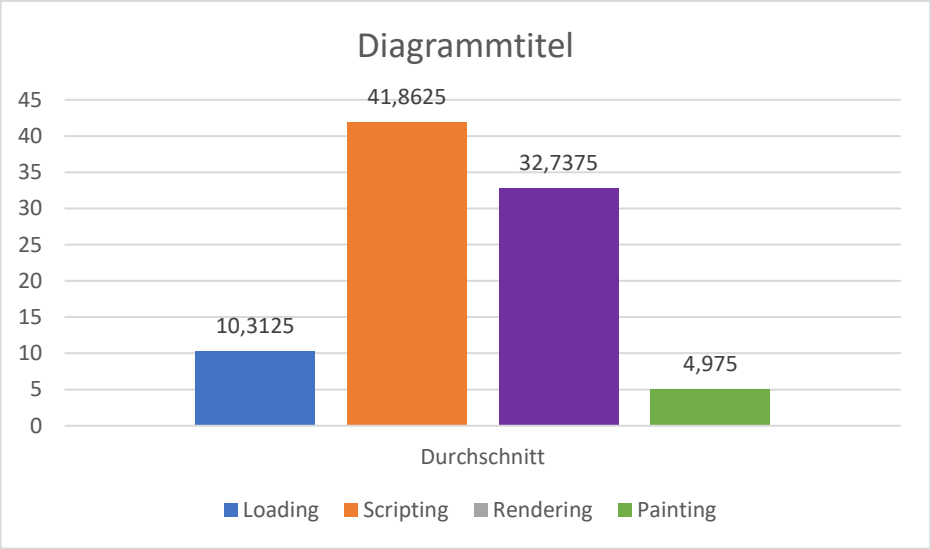

Abbildung 26: Anwendung: Bilder

Gesamtladezeit: 89,8875ms

In diesem Fall wurde die Ladezeit der Test-Webseite um 3,375ms erhöht. Eine Erhöhung um 3,9%. Vor allem beim Painting ist das Ergebnis aussagekräftig. Hier braucht der Browser 1,7125ms (52,5%) länger als beim Ergebnis zuvor. Das kann man erneut auf das WebP-Format zurückführen.

Am Ende sind die Ergebnisse erwartungsgemäß. Die Ladezeiten von beiden Strategien nähern sich an. Wobei die Schwächen beider erhalten bleiben.

Nichtdestotrotz könnten beide Strategien ihren Einsatz im modernen Web finden. Vorallem das verzögerte Laden bietet sich bei Webseiten mit sehr vielen Bildern an. Mit dieser Strategie kann man zwar keine besseren Ladezeiten erreichen, aber die Webseite stellt dem Nutzer schnell etwas bereit, um damit zu interagieren.

Das WebP-Format ist aufgrund seiner Leichtigkeit ideal für Webseiten mit eingeschränkten Speichervolumen. In anderen Kriterien ist das JPG-Format die bessere Wahl.

5.4 Anwendung: Kritischer Rendering-Pfad

Nach dem Entfernen des nichtkritischen CSS sieht der kritische CSS wie folgt aus:

```
*{-webkit-box-sizing:inherit;box-sizing:inherit;background-repeat:no-repeat}
html{-webkit-box-sizing:border-box;box-sizing:border-box;scroll-behavior:smooth}
body{font-family:'Montserrat',sans-serif;color:rgb(46,46,46);background:#ffffff;text-rendering:optimizeLegibility;-webkit-font-smoothing:antialiased}
.breite{width:90%;max-width:1340px;margin:0 auto}
input{font-size:1em}input[type="submit"],.cookie input.accept[type="submit"]{background-color:rgb(197,0,11);color:#fff}
.screen-reader-text{display:none}a{color:rgb(197,0,11)}
#site{border-top:1px solid rgb(197,0,11)}
header{background:rgb(197,0,11);color:#ffffff}
header .breite{position:relative;padding-top:.3em}
.top-row{display:none;position:absolute;right:0;top:1em;font-size:.8em}
.top-row a{color:inherit;text-decoration:none}
.top-row .mail{margin-right:1.5em;padding-left:1.5em;background:url(../img/envelope.svg) left center/1em no-repeat}
.top-row .tel{padding-left:1.5em;background:url(../img/phone.svg) left center/1em no-repeat}
#logo{display:inline-block;margin-right:auto;width:120px}#logo img{display:block}
nav.wk-horizontal{-ms-flex-item-align:end;align-self:flex-end}
nav.wk-horizontal ul{background:transparent;margin:0 -1em;padding-bottom:.5em;font-size:1.3em;text-transform:uppercase;letter-spacing:.05em;font-weight:500}
nav.wk-horizontal ul li a{position:relative}
nav.wk-horizontal ul li a::after{content:'';position:absolute;left:1em;bottom:.4em;right:100%;height:2px;background:rgb(255,255,255)}
nav.wk-horizontal ul .item-108 a::before{display:block;content:'Unsere';font-size:.6em;text-transform:none;letter-spacing:0;font-weight:400}
nav.wk-horizontal ul .item-109 a::before{display:block;content:'Unser';font-size:.6em;text-transform:none;letter-spacing:0;font-weight:400}
nav.wk-horizontal ul .item-110 a::before{display:block;content:'Unsere';font-size:.6em;text-transform:none;letter-spacing:0;font-weight:400}
.hero{position:relative;overflow:hidden}.hero img{display:block;width:100%;max-width:none;min-width:480px}.hero .title{display:none;background:rgba(197,0,11,1);color:#ffffff;
main .breite{padding:2em 0 3em}
main{background:url(../img/haken.jpg) right center no-repeat}
.page-101 main .breite{position:relative;padding:3em 0 5em 3.5em}
.page-101 main .breite::before{content:'';display:block;position:absolute;top:0;left:0;background:rgb(197,0,11);width:8px;height:120px;border-bottom-left-radius:4px;border-bott
h1{margin-bottom:1em;color:rgb(94,94,94);font-weight:700}
h1 small{font-size:.6666em;font-weight:400}
footer a{color:inherit}#gotop{opacity:1;position:fixed;bottom:10px;right:10px;z-index:1000;width:40px;height:40px;border-radius:50%;background:rgb(197,0,11) url("../img/symbols
@media only screen and (min-width:768px){
    header .breite{display:-webkit-box;display:-ms-flexbox;display:flex;-ms-flex-wrap:wrap;flex-wrap:wrap}
    .hero .title{position:absolute;left:0;bottom:0;right:0;background:rgba(197,0,11,.65);font-size:1.4em}}@media only screen and (min-width:1024px){#logo{width:11em}
    .top-row{display:block}.hero .title{display:block}
}
@media only screen and (min-width:1280px){
    nav.wk-horizontal ul{font-size:1.5556em}
}
```

Abbildung 27: Kritische CSS der Test-Webseite

Diese neue CSS-Datei ist 4kB groß und ersetzt, wie im Kapitel 3.3.2 beschrieben, die Ausgangsdatei. Die Ausgangsdatei wird daraufhin am Ende des schließenden Body-Tags eingebunden.

Um die eingebunden JS-Dateien in Joomla! asynchron zu laden, muss man die Art der Einbindung beachten. Joomla! stellt eine Funktion bereit, wie man JS einbinden kann. Dafür wird folgende Funktion genutzt:

```
public function addScript($url, $type = "text/javascript", $defer = false, $async = false)
{
    $this->_scripts[$url]['mime'] = $type;
    $this->_scripts[$url]['defer'] = $defer;
    $this->_scripts[$url]['async'] = $async;

    return $this;
}
```

Abbildung 28:Joomla!-Funktion zur Einbindung von JS

Das bedeutet, dass im Falle der Test-Webseite die Angabe wie folgt aussieht:

```
$this->addScript( 'https://dev.webkommunikation24.de/wk-base/0.5.0/wk-base.js', 'text/javascript', true, true );
$this->addScript( '/templates/'.$this->template.'/js/scripts.js?ver=1', 'text/javascript', true, true );
```

Abbildung 29: JS-Einbindung am Beispiel der Test-Webseite

Das sind die Untersuchungsergebnisse nachdem der kritische Rendering-Pfad angepasst wurde:

Tabelle 11: Anwendung: Kritische Rendering-Pfad

Kritischer Rendering-Pfad	Loading	Scripting	Rendering	Painting
1	13,4	31,3	29,6	2,1
2	11,5	30,8	26	1,7
3	10,8	30,7	26,6	1,7
4	11,7	32,5	27,4	1,8
5	10,9	33,8	26,7	1,6
6	12,5	30,3	26,5	1,6
7	10,9	31,8	26,5	1,7
8	12,3	33,1	26,7	1,7
Durchschnitt	**11,75**	**31,7875**	**27**	**1,7375**

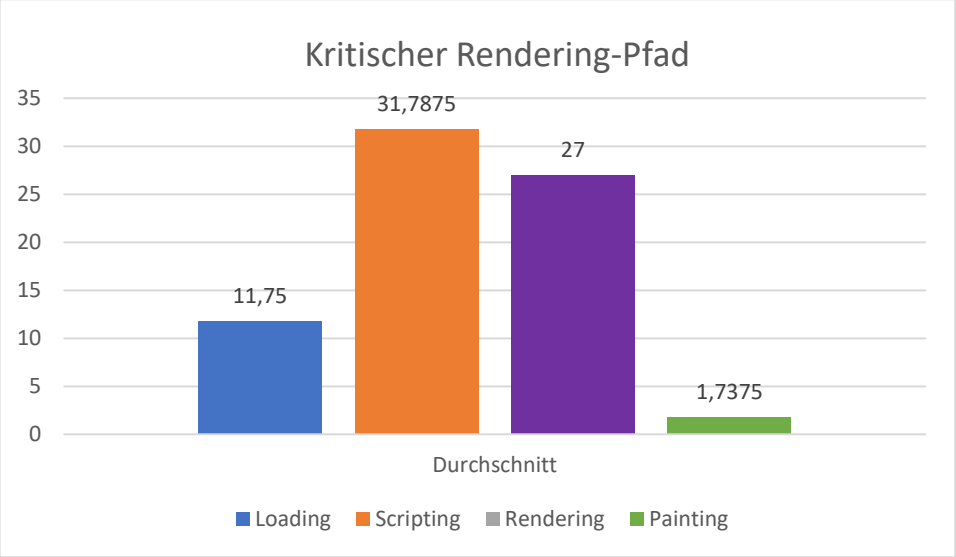

Abbildung 30: Anwendung: Kritischer Rendering-Pfad

Gesamtladezeit: 72,275ms

Der angepasste kritische Rendering-Pfad hat somit die Gesamtladezeit um 14,2375ms senken können. Das ist eine Verbesserung um 16,46%.

Das Scripting konnte um rund 10ms gesenkt werden. Damit kann man schluss-folgern, dass das Scripting schneller ausgeführt werden kann, wenn die Daten besser organisiert sind. Zudem scheint die asynchrone Ausführung ebenfalls eine positive Auswirkung auf das Scripting zu haben.

Erwartungsgemäß wurde das Rendering-Zeit verringert, weil weniger Elemente diese blockieren.

Erstaunlich sind die Ergebnisse beim Painting. Die Zeit ist 65,08% schneller als zuvor. Ein Grund dafür könnte ebenfalls die schnelle Bereitstellung der nötigen Ressourcen sein.

Die einzige negative Auswirkung, den der angepasste kritische Rendering-Pfad hat, liegt beim Loading. Da braucht der Browser 1,4375ms länger. Das kann an der zusätzlichen CSS-Datei liegen.

5.5 Datenkomprimierung

Laut *gidnetwork*[38] wurden 72,7% der Bytes der Test-Webseite (vorher: 25,162; nachher: 6,863) durch Gzip und Deflate komprimiert. Nach den erneuten Untersuchungen ergaben sich folgende Ergebnisse:

Tabelle 12: Anwendung: Datenkomprimierung

Datenkomprimierung	Loading	Scripting	Rendering	Painting
1	11,7	30,9	26,9	1,8
2	12	31,3	26,2	1,7
3	10,3	32,7	28,5	1,7
4	12	31,6	26,8	1,7
5	11,8	32	26,5	1,7
6	11,7	31,5	28,8	1,7
7	11,6	31,1	27	1,9
8	11,9	32	27,1	1,8
Durchschnitt	**11,625**	**31,6375**	**27,225**	**1,75**

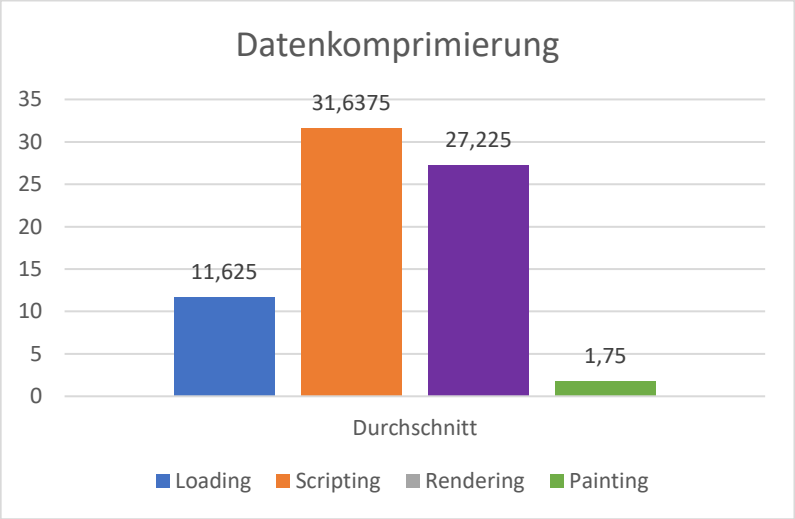

Abbildung 31: Anwendung: Datenkomprimierung

[38] Vgl. gidnetwork - http://www.gidnetwork.com/tools/gzip-test.php, Aufruf: 29.05.2019

Gesamtladezeit: 72,2375ms

Die Datenkomprimierung verkürzte die Gesamtladezeit um weitere 0,0375ms. Somit bleiben die Ergebnisse weitaus unverändert. Man kann davon ausgehen, dass die Ergebnisse deutlich besser wären, wenn man eine Webseite untersucht, die deutlich größer ist als die Test-Webseite. Denn desto mehr Daten komprimiert werden können, umso weniger Bytes muss der Browser herunterladen.

Da die Implementierung von Gzip und Deflate so unkompliziert ist, kann man diese Strategie zur Verringerung der Ladezeiten bei allen Webseiten verwenden.

5.6 Anwendung: Der Zwischenspeicher – Cache

Nach der Implementierung des Caches wie im Kapitel 3.4 beschrieben, ergaben sich folgende Untersuchungsergebnisse:

Tabelle 13: Anwendung: Der Zwischenspeicher - Cache

Cache	Loading	Scripting	Rendering	Painting
1	11,6	31	27,1	1,6
2	11,8	31,2	27	1,6
3	11,3	30,8	26,8	1,7
4	11,5	31,6	27,4	1,6
5	11,2	31,8	27,5	1,6
6	11,3	31,2	27,8	1,7
7	10,9	32	27,3	1,6
8	11,7	31,5	27,4	1,6
Durchschnitt	**11,4125**	**31,3875**	**27,2875**	**1,625**

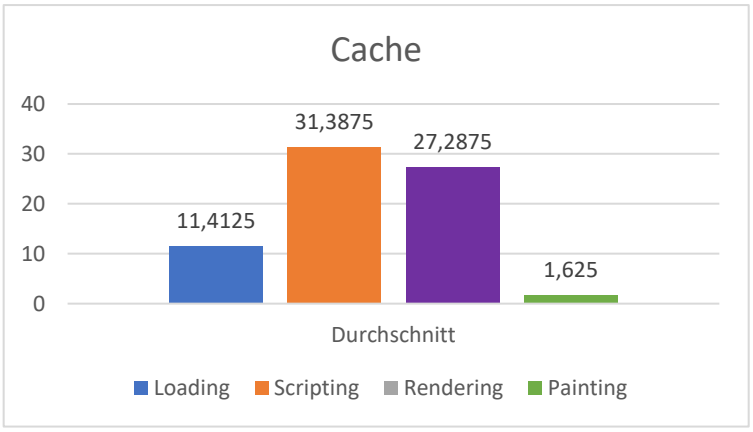

Abbildung 32: Anwendung: Der Zwischenspeicher - Cache

Gesamtladezeit: 71,7125ms

Mit dieser Strategie konnte die Gesamtladezeit um 0,525ms verringert werden. Das ist eine Verbesserung um 0,73%.

Wie man anhand der *Tabelle 11* erkennen kann, konnte man speziell die Ladezeit von Loading und Painting senken. Daraus könnte man schlussfolgern, dass die zwischengespeicherten Daten, die der Browser aus seinem Zwischenspeicher erhält, schneller geladen werden als vom Server.

Außerdem kann man vermuten, dass der Browser das Painting bei bekannten Daten schneller ausführen kann.

Abschließend kann man festhalten, dass der Zwischenspeicher von jeder Webseite genutzt werden sollte, da die meisten Browser eine voreingestellte Zwischenspeichergröße von über 200MB besitzen. Moderne Browser erkennen früh, welche Daten ladezeitkritisch sind und zwischengespeichert werden sollten.

5.7 Anwendung: HTTPS

Der folgenden Tests wurde ohne HTTPS ausgeführt:

Tabelle 14: Anwendung: HTTPS

Ohne HTTPS	Loading	Scripting	Rendering	Painting
1	9,3	25,2	27,3	1,7
2	9,4	25,7	27,4	1,7
3	9,6	25,4	27	1,6
4	9,2	25,4	27,8	1,7
5	9,3	26,2	27,5	1,7
6	9,5	25,7	26,9	1,6
7	9,1	25,2	27,8	1,7
8	9,5	25,6	27,3	1,7
Durchschnitt	**9,3625**	**25,55**	**27,375**	**1,675**

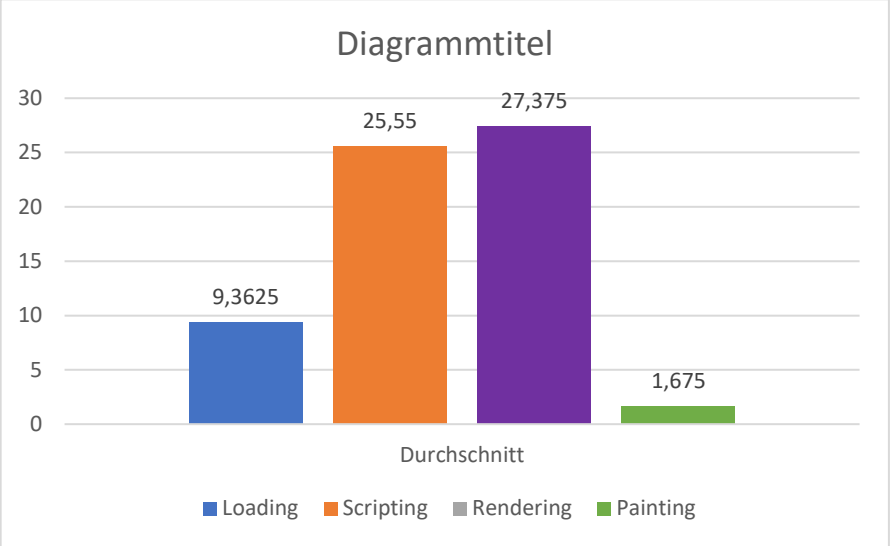

Abbildung 33: Anwendung: HTTPS

Gesamtladezeit: 63,9625

Damit wurde die Gesamtladezeit um weitere 7,75ms verkürzt. Das ist eine Verbesserung um 10,81%.

Man kann davon ausgehen, dass das Loading aufgrund der schnelleren Bereitstellung der Informationen verkürzt wird. Dazu müssen weniger Anfragen vom Server bearbeitet werden.

Dass das Rendering verkürzt wurde, kann man mit dem geringeren Aufwand des Browsers erklären. Der Browser muss die empfangenen Daten nicht erst übersetzen und kann direkt mit den nächsten Schritten beginnen.

Die Abweichungen von Rendering und Painting sind so gering, dass man davon ausgehen kann, dass HTTPS keinen Einfluss auf diese hat.

Dieses Ergebnis ist zugleich das Schlussresultat. Gegenüber den Ausgangstests konnte die Gesamtladezeit um 23,5275ms verkürzt werden. Das ist eine Verbesserung von 26,9%.

6 Fazit

Die Untersuchungen zeigen, dass die Strategien zur Verringerung der Ladezeiten zum größten Teil zu positiven Ergebnissen führten. Im Hinblick auf den Aufwand, den der Entwickler zusätzlich tätigen muss, ist der Nutzen als sehr hoch zu betrachten. Dabei muss man auch beachten, dass die Test-Webseite nicht viele Datenmengen verwendet und man bei größeren Webseiten deutlich bessere Ergebnisse erzielen könnte.

Zusammenfassend ist festzuhalten, dass fünf von sechs Anwendungen den erwartenden Erfolg erzielten. Moderne Bildtypen haben gute Ansätze und nehmen bereits weniger Speicherplatz ein als JPEG im Vergleich. Sicherlich ist es nur eine Frage der Zeit bis das altbewährte Bildformat ersetzt werden kann.

Zusätzlich haben alle angewandten Strategien (ausschließlich Anwendung: HTTPS) einen positiven Effekt auf das Ranking von modernen Suchmaschinen. Grund dafür ist die verbesserte Performance und die Einhaltung von Programmierrichtlinien der Suchmaschinen wie beispielsweise die von Google.[39]

Das Verringern von Ladezeiten von Webseiten wird stets ein Thema bleiben. Jedoch muss beachtet werden, dass man durch das Einhalten aller Programmierrichtlinien und das Verkleinern von Ausgangsdaten sowie allen anderen Strategien, die die Ladezeit verringern können, kein Optimum für alle erreicht werden kann. Das liegt einzig und allein im Auge des Betrachters.

[39] Siehe auch Google, Startleitfaden zur Suchmaschinenoptimierung (SEO) - https://support.google.com/webmasters/answer/7451184?hl=de Aufruf: 04.06.2019

Literaturverzeichnis

[Q1] Think with Google, Find Out How You Stack Up to New Industry Benchmarks for Mobile Page Speed - https://think.storage.googleapis.com/docs/mobile-page-speed-new-industry-benchmarks.pdf Stand: Februar 2017

[Q2] Guillaume Thibaux, ETAM's page speed load time reduced by 700 ms - https://blog.quanta.io/etam/ Aufruf: 05.06.2019

[Q3] Greg Linden, Marissa Mayer at Web 2.0 - http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html Aufruf: 05.06.2019

[Q4 Abb.1] Inquirer.net, Tim Barnes-Lee - https://technology.inquirer.net/60845/inventor-world-wide-web-wins-computings-nobel-prize

[Q5] BERNERS-LEE, Request for Comments 1945 - https://tools.ietf.org/html/rfc1945 Stand: Mai 1996

[Q6] FIELDING, Request for Comments 2616 - https://tools.ietf.org/html/rfc2616 Stand: Juni 1999

[Q7] BELSHE, Request for Comments 7540 - https://tools.ietf.org/html/rfc7540 Stand: Mai 2015

[Q8] PEON & RUELLAN, Request for Comments 7541 - https://tools.ietf.org/html/rfc7541 Stand: Mai 2015

[Q9] MEIER, GREGOR, Pagespeed Optimierung, Schritt für Schritt zur schnelleren Webseite, München, 2016 S. 13-17

[Q10 Abb.3] Inflow - https://inflow.de/dynamisch-statisch Aufruf: 04.05.2019

[Q11] RODRIGUEZ HERRERA, The JPEG standar - https://w3.ual.es/~vruiz/Docencia/Apuntes/Coding/Image/03-JPEG/index.html Aufruf: 27.05.2019

[Q12] BOUTELL, Request for Comments - https://tools.ietf.org/html/rfc2083 Stand: März 1997

[Q13] ONLINE-UMWANDELN, Informationen zum WEBP Dateiformat - https://online-umwandeln.de/dateiformat/webp/ Stand: 21.04.2019

[Q14] Can I use? - https://caniuse.com/#feat=webp Stand: 23.05.2019

[Q15] MEIER, Pagespeed Optimierung, S. 36-37

[Q16] MEIER, Pagespeed Optimierung, S. 44-45

[Q17] BILLO, Was ist Datenkompression? - https://www.storage-insider.de/was-ist-datenkompression-a-764167/ Aufruf: 23.05.2019

[Q18] MÜHLINGHAU, STEFAN, Der LZ77 Algorithmus - http://www.gm.fh-koeln.de/~hk/lehre/ala/ws0506/Praktikum/Projekt/D_rot/Ausarbeitung.pdf Aufruf: 25.05.2019

[Q19] DEUTSCH, Request for Comments 1952 - https://tools.ietf.org/html/rfc1952 Stand: Mai 1996

[Q20] DEUTSCH – Request for Comments 1951 - https://tools.ietf.org/html/rfc1951 Stand: Mai 1996

[Q21] HUFFMAN, DAVID, A Method for the Construction of Minimum-Redundancy Codes - http://compression.ru/download/articles/huff/huffman_1952_minimum-redundancy-codes.pdf Stand: 1952

[Q22] MEIER, PageSpeed Optimierung, S. 93

[Q23] Nicole Martin, Has “Above the Fold” Gone the Way of Phone Books? - https://www.paceco.com/insights/design-development/fold-importance-web-design/ Stand: 28. August 2017

[Q24 Abb. 16] Google, https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-tree-construction?hl=de Stand 27.05.2019

[Q25] W3C Recommendation, https://www.w3.org/TR/html5/ Stand: 14. Dezember 2017

[Q26] Google, Objektmodell erstellen, https://developers.google.com/web/fundamentals/performance/critical-rendering-path/constructing-the-object-model?hl=de Aufruf: 01.05.2019

[Q27] Google, Erstellung der Rendering-Baumstruktur, Layout, und Paint - https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-tree-construction?hl=de Stand: 27.04.2019

[Q28 Abb.17] Jonas Sebastian Ohlsson - https://jonassebastianohlsson.com/criticalpathcssgenerator/ Stand: 24.04.2019

[Q29] W3schools, HTML Layouts - https://www.w3schools.com/html/html_layout.asp Stand: 29.04.2019

[Q30] OHLSSON, JONAS SEBASTIAN, Critical Path CSS Generator, https://jonassebastianohlsson.com/criticalpathcssgenerator/#what-is Stand: 24.04.2019

[Q31] Mozilla, JavaScript-Grundlagen - https://developer.mozilla.org/de/docs/Learn/Getting_started_with_the_web/JavaScript_basis Aufruf 02.05.2019

[Q32] Google, Interaktivität dank JavaScript - https://developers.google.com/web/fundamentals/performance/critical-rendering-path/adding-interactivity-with-javascript?hl=de Aufruf: 02.05.2019

[Q33] Media-Company, HTTPS und SSL Verschlüsselung – Was ist das und warum ist das für Webseiten unverzichtbar? - https://www.media-company.eu/blog/allgemein/https-und-ssl-verschluesselung/ Aufruf: 02.06.2019

[Q34] SECURITRY-INSIDER, Was ist TLS? - https://www.security-insider.de/was-ist-tls-transport-layer-security-a-673066/ Aufruf: 02.06.2019

[Q35] gidnetwork - http://www.gidnetwork.com/tools/gzip-test.php, Aufruf: 29.05.2019

[Q36] Google, Startleitfaden zur Suchmaschinenoptimierung (SEO) - https://support.google.com/webmasters/answer/7451184?hl=de Aufruf: 04.06.2019

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Ort, Datum Vorname Nachname