Technology
Arts Sciences
TH Köln

Praxisprojekt WS17/18

Identifikation und Evaluation von Maßnahmen zur effizienten Performanceoptimierung eines web-basierten Buchungsprozesses

am Beispiel des Special Interest Reiseanbieters DIVE.IS

von
Sven Schiffer - 11088960

an der Technology, Arts and Sciences TH Köln
Campus Gummersbach
im Studiengang Medieninformatik (B.Sc.)

Betreuer: Prof. Dipl.-Des. Christian Noss
Technology, Arts and Sciences TH Köln

Technology, Arts and Sciences TH Köln, 4. Dezember 2018

Abstract

Der Special Interest Reiseanbieter DIVE.IS ist laut Tripadvisor auf Platz 1 der Outdoor-Aktivitäten in Reykjavik. Aus dem geplanten Relaunch der Website soll ein performanter, mobil optimierter Internetauftritt entstehen. Diese Arbeit listet Metriken zur Performanceoptimierung auf, welche Kosten und Nutzen gegeneinander abwägt, um die Zielsetzung zu erreichen. Zusätzlich wurden die fünf effizientesten Maßnahmen zur Performanceoptimierung einer Website prototypisch implementiert und ausgewertet. Die Evaluation der Prototypen unterstreicht die vermuteten Effizienzen der Maßnahmenauflistung.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Unternehmensvorstellung

DIVE.IS ist eine isländische Sporttauchschule mit Sitz in Reykjavík. Sie wurde 1997 von Tómas J. Knútsson gegründet. Tobias Klose leitet das Unternehmen seit 2007. Anfang 2018 beschäftigt das Unternehmen über 30 Mitarbeiter. DIVE.IS hat sich auf Schnorchel- und Tauchtouren spezialisiert und hält in dieser Kategorie Platz 1 auf Tripadvisor (Tripadvisor, 2017), einer Plattform auf der Nutzer ihre Erfahrungsberichte im Bereich Reisen bewerten können.

Die Tauchschule ist mit der weltweit führenden Tauchausbildungsorganisation PADI zertifiziert. Das System der PADI-Tauchausbildung basiert auf einer fortschrittlichen Ausbildung, die den Tauchschülern in verschiedenen Etappen Fertigkeiten, sicherheitsrelevante Informationen und lokales Umweltwissen vermittelt.

Die Betreuung der Web-Services wird von der deutschen Firma Klickmeister GmbH durchgeführt.

1.2 Problemfeld

Das Thema der Arbeit fokussiert sich auf die Optimierung der Ladegeschwindigkeit der Website `https://www.dive.is`. Der Trend, mobil im Internet zu surfen, nimmt gemäß eine Studie über die Nutzungszahlen von mobilen Devices gegenüber Desktop Computern immer mehr zu (Enge, 2018).

Bereits seit 2015 erhält Google laut dem online Magazin T3N (Hüfner, 2017) mehr mobile Suchanfragen als von Desktops. Dies bestätigen u.a. Design-Konzepte wie ”mobile-first”, bei denen zuerst eine für die mobile Ansicht optimierte Version konzipiert wird, bevor schrittweise größere Varianten für höhere Auflösungen umgesetzt werden.

Google veröffentlicht 2016 eine Studie deren Auswertung ergeben hat, dass viele Reisende nach der Ankunft spontan sein wollen (Google, 2016). 85 % aller Urlauber entscheiden vor Ort, welche Aktivitäten sie buchen. Die Hälfte nutzt dabei das Smartphone.

Viel Recherche der potentiellen Kunden wird in Netzwerken durchgeführt, die nicht immer die optimale Bandbreite liefern können. Das kann zum einen das teilweise langsame mobile Internet sein, zum anderen sind selbst Hotspots in Hotels oft überlastet wie Techquickie in ihrem Video (Techquickie, 2017) “Why Is Hotel Wi-Fi So Slow?” erklären.

Umso wichtiger ist es, dass die User Experience des Nutzers während des Buchungsprozesses nicht durch lange Ladezeiten negativ beeinflusst wird.

Dies belegt auch eine Umfrage (Akami, 2009), die bereits im Jahre 2009 von Akami durchgeführt wurde: “[...] 40 percent of shoppers will wait no more than three seconds before abandoning a retail or travel site.“

Hinzu kommt, dass immer mehr Touristen nach Island reisen. Im Jahre 2017 gab es im Vergleich zum Vorjahr einen Zuwachs von 24.2 %. Dies geht aus den veröffentlichten Einreisezahlen des isländischen Fremdenverkehrsamt Ferðamálastofa (ferdamalastofa, 2018) hervor.

In der Fallstudie “Removing Friction In UX: Last-Minute Travel Planning And Activity Booking (A Case Study)“ (Þór Gústafsson, 2017) hat sich das isländische Reisebüro Get local (`https://www.getlocal.is/`) der angesprochenen Herausforderung angenommen. Sie zeigt, welche Maßnahmen getätigt wurden, um ein optimales mobiles Nutzungserlebnis zu schaffen.

DIVE.IS ist laut Tripadvisor (Tripadvisor, 2017) auf Platz 1 der Outdoor-Aktivitäten in Reykjavik. Dieses Ranking soll sich auch in der Performance der Website widerspiegeln, um dieser Platzierung auch im digitalen Umfeld gerecht zu werden.

1.3 Zielsetzung

Ziel ist es eine Auflistung von nachvollziehbaren Metriken zur Performanceoptimierung zu erarbeiten, welche Kosten und Nutzen gegeneinander abwägt. Diese Auflistung soll dazu dienen

einen Eindruck über verschiedenste Performanceoptimierungsmöglichkeiten zu bekommen, um diese in einen Ressourcenplan eines zukünftigen Projektes konkret abwägen zu können.

1.4 Vorgehensweise

Vorab wird auf verschiedene gängige Performanceoptimierungsmöglichkeiten im Bereich Web Development eingegangen und diese erläutert. Im Anschluss werden mögliche Defizite der aktuellen Website identifiziert, nach dessen Einfluss auf die Performance aufgelistet und entsprechende Maßnahmen genannt. Hier steht der wirtschaftliche Kosten- und Nutzenfaktor im Vordergrund.

Optimierungen sollten sich auf Grund der hohen Marktanteile von Google's Suchmaschine stark auf Richtlinien der Firma Google beziehen (SEO Summary, 2018).

Für die effizientesten Optimierungsmaßnahmen sollen Prototypen entwickelt werden, mit denen der Geschwindigkeitsgewinn gemessern werden kann.

Kapitel 2

IST-Zustand

2.1 Infrastruktur

Seit Anfang 2017 hat die Planung einer neuen Website für DIVE.IS begonnen. Diese soll die mittlerweile überladene WordPress-Instanz ablösen und mittels Kirby realisiert werden. WordPress ist ein Open-Source Content-Management-System, welches in PHP geschrieben ist und die Daten in einer Datenbank speichert (WordPress, 2018a). Es wurden bereits über 50.000 WordPress-Plugins veröffentlicht, mit denen sich das System sehr schnell erweitern lässt (WordPress, 2018b). Im Gegensatz zu WordPress kommt das dateibasierte CMS Kirby ohne Datenbank aus.

Die aktuelle Website liegt bei einem isländischen Hosting-Anbieter. Dieser hat jedoch in der Vergangenheit zu mangelhaften Verfügbarkeiten geführt. Immer wieder kam es zu zurückgesetzen SSH-Zugängen und vollgelaufenen Festplatten, die ein Entwickeln erschwerten oder unmöglich machten. Auf Grund dieser Ausfälle hat die Firma Klickmeister beschlossen den Hoster zu wechseln. Das Hosting für die neue Website wurde auf einen bereits aus anderen Projekten bekannten Hoster verlagert.

Der neue Hosting-Vertrag sieht Solid-State-Drives für schnelle Antwortzeiten vor und stellt nahezu uneingeschränkten Shell Zugriff bereit. Diese Art des flexiblen Hostings eröffnet den Entwicklern der neuen Website eine Menge an Möglichkeiten zur Optimierung der Deploymentprozesse und Serververwaltung sowie Bereitstellung von statischen Ressourcen.

DIVE.IS setzt beim elektronischen Verkauf von Touren auf die Tourismus Verkaufsplattform Bókun. Bókun wurde 2012 in Reykjavík gegründet. Laut ihrer Website erwirtschafteten sie bereits Mitte 2016 1 % des isländischen Bruttoinlandsproduktes (Bókun, 2017). Das Unternehmen

stellt seinen Kunden eine Verkaufsplattform zur Verfügung, mit der sie unter anderem ohne technisches Wissen Produkte auf einer Website integrieren und online Zahlungen entgegennehmen können. Die Kommunikation zu Bókun wird über eingebundene Widgets mittels IFrames realisiert. Diese sind zwar einfach einzubinden, bringen aber auch viele negative Aspekte mit sich. Zum einen werden viele, nicht benötigte Daten zum Nutzer übertragen und nicht immer ist der bereitgestellte Code und dessen Struktur optimiert, wie sich aus diversen Analysetools wie Google's Pagespeed Insights ablesen lässt.

2.2 Codebase

Die Firma Klickmeister setzt in der jüngsten Vergangenheit auf das flexible, dateibasierte Content-Management-System Kirby.

Dieses in PHP programmierte CMS soll auch die serverseitige Grundlage der neuen Website darstellen. Es ist sehr schlank gehalten, lässt sich jedoch auf Grund der Model-View-Controller (MVC) Artchitektur gut erweitern ohne die eigentliche Übersichtlichkeit zu verlieren.

Um den Programmiercode zu strukturieren wird über ein Kirby-Plugin auf das Atomic-Design-Pattern gesetzt. Frost (Frost, 2018) beschreibt die Struktur einer Website mit der Materienstruktur des Universums, welche auf wesentliche Kernelemente heruntergebrochen werden kann. Diese Kernelemente bilden zusammengesetzt größere Strukturen, die sich wiederum ebenfalls zu noch größeren Strukturen zusammensetzen. Die von Frost entworfene Design Methodik unterteilt den Code in drei verschiedene Kategorien: Atome, Moleküle und Organismen. Kleine Objekte/Codestrukturen werden in Atomen untergebracht, die in Molekülen zu größeren Codestrukturen zusammengefasst werden. Mehrere Moleküle können dann zu noch größeren und komplexeren Organismen gebündelt werden. Diese doch recht abstrakte Erläuterung lässt sich anhand eines Formulars gut beschreiben. Das Formular sei ein Organismus und enthält mehrere Formularfelder. Diese Formularfelder werden als Moleküle eingeordnet und enthalten Atome, wie z.B. ein Eingabefeld, ein Label und einen Statustext für das übergeordnete Formularfeld.

Die aktuelle Entwicklungsumgebung der neuen Dive-Webseite beinhaltet einen Build-Prozess. Dieser ist mittels Gulp realisiert. Gulp ist eine Sammlung von Werkzeugen zur Automatisierung von Aufgaben in einem Entwicklungsablauf (Gulp, 2018). Dieser Gulp-Prozess beobachtet alle dynamischen JavaScript- und Stylesheet-Dateien. Ändert sich eine dieser Dateien werden alle JavaScript-Dateien zu einer Datei zusammengefügt und minimiert. Beim Minimieren werden alle Steuerzeichen, wie Zeilenumbrüche und nicht benötige Leerzeichen, entfernt. Zusätzlich

werden Variablen- und Funktionsnamen verkürzt, um die Dateigröße zu minimieren. Ein ähnlicher Prozess wird auch auf die Stylesheets angewendet.

Um responsive Bilder bereitstellen zu können werden diese über CSS-Breakpoints gesteuert. Responsive Bilder haben den Vorteil, dass sie sich dem aktuellen Viewport des Browsers anpassen können. Für jede Seite im Backend (Kirby-Panel) wird eine SCSS-Datei angelegt. Ändert sich eine Datei im Panel oder wird eine neue Datei hinzugefügt, wird ein Kirby-Hook ausgelöst. Mit Hilfe eines Hooks lässt sich der Programmcode an bestimmte Ereignisse binden. Dieser Hook iteriert über alle Bilder der Seite und erzeugt für jedes Bild einen CSS-Selektor pro Breakpoint. Nachdem die SCSS-Dateien erzeugt worden sind, wird im Anschluss der Gulp-Prozess ausgeführt und eine neue CSS-Datei angelegt.

2.3 Render-Prozess

Im Folgenden wird der Render-Prozess des Kirby-CMS erläutert. Es wird von den anfänglich gesetzten Einstellungen des Kirby-Starter-Kits ausgegangen, um die Komplexität der Erläuterung so gering wie möglich zu halten. Die Basis der anzuzeigenden Seite bietet in erster Linie die URL und dessen direkter Dateipfad innerhalb des von Kirby bereitgestellten Content-Ordners. So wird die URL “domain.tld/pfad/unterpfad“ in den Dateipfad relativ zur .htaccess (insofern nicht anders konfiguriert) “./content/pfad/unterpfad“ aufgelöst. In dem genannten Content-Ordner wird nach einer Textdatei gesucht, welche Daten für die aufgerufene Seite bereithält. Das ist zum einen das zu rendernde Template, welches das Markup enthält, und zum anderen die Daten zum Befüllen des Templates. Der Dateiname definiert also in erster Linie wie die Seite hinter der URL aussehen soll.

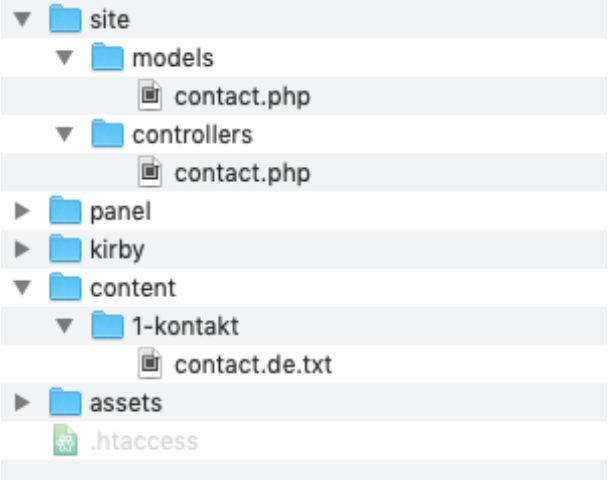

ABBILDUNG 2.1: Kirby CMS Dateistruktur

Zusätzlich können Codestücke in Controller und Models ausgelagert werden, die in den entsprechenden Unterordnern im Kirby-Site-Ordner hinterlegt werden. Mehr zu dem Thema Kirby Models und Controller kann unter `https://getkirby.com/docs/developer-guide/advanced` nachgeschlagen werden. Im Template steht dem Entwickler eine Variable zur Verfügung, die den Inhalt der Seite enthält. Sie ist mittels `$page` adressierbar.

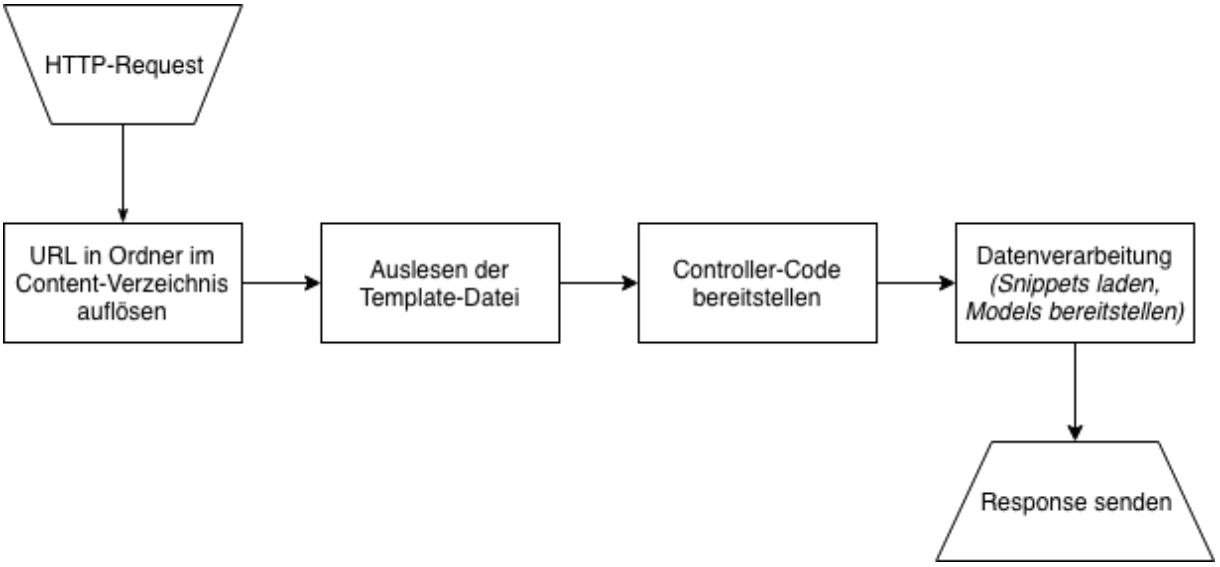

ABBILDUNG 2.2: Kirby Render Pipeline

Kapitel 3

Performance

3.1 Definition des Begriffes “Performance“ im Kontext

In dieser Arbeit wird der Begriff Performance sehr allgemein gehalten, um nicht nur die Ladegeschwindigkeit, sondern auch andere Metriken einer Website, die sich auf die User-Experience auswirken, im Allgemeinen zu beschreiben. Die Ladegeschwindigkeit einer Website lässt sich in Server-Performance und Front-End-Performance unterteilen.

Im Gabler Wirtschaftslexikon (Dr. Cordula Heldt, 2013) wird der Begriff Performance in der EDV definiert als “Verhalten eines Softwareprodukts bei der Ausführung; v.a. beurteilt anhand der Laufzeiteffizienz (Effizienz) und der Antwortzeiten.“

Die serverseitige Performance umfasst unter anderem die Verarbeitungsgeschwindigkeit der Website bis das HTML-Gerüst ausgeliefert werden kann. Clientseitige Performance beinhaltet Aspekte wie das DOM-Parsing und Reaktionsgeschwindigkeiten auf Benutzereingaben, sowie die Anzahl der Einzelbilder die pro Sekunde auf dem Display vom Endgerät gerendert werden können.

3.2 Erläuterung der Messverfahren

In modernen Browsern sind Entwicklerwerkzeuge implementiert. Mit diesen lässt sich unter anderem der Quellcode untersuchen, das Netzwerkhistogramm analysieren oder Metadaten der Website abrufen. Die Google PageSpeed Insights eignen sich gut, um eine Website aus

Google's Sichtweise analysieren zu lassen. Google's Lighthouse ist ein automatisiertes Werkzeug zur Verbesserung der Qualität von Webseiten, bietet deutlich detailliertere Analysen und kann auch auf lokalen Webseiten getestet werden (Google, 2018e). Lighthouse stellt Tests bereit, welche die Leistung und Barrierefreiheit fortschrittlicher Webanwendungen widerspiegeln können und somit messbare Metriken zur Evaluation der Websiteperformance bereitstellen. Lighthouse drosselt die Netzwerkgeschwindigkeit auf 3G und reduziert die CPU Geschwindigkeit auf ein Viertel, um die Performance eines mobilen Devices zu emulieren (GoogleChrome, 2018).

Um die Ergebnisse zu verifizieren, sollte ein weiteres Werkzeug hinzugezogen werden. Der Pingdom Website Speed Test bietet nochmal eine andere Sichtweise auf die Performance einer Website.

Um die Effizienz der Verfahren bewerten zu können, werden vorerst verschiedene Kriterien gemessen, welche im Anschluss mit der implementierten Optimierungsmaßnahme verglichen werden. So entsteht für jede implementierte Optimierung ein quantitativer Vergleichswert der im Abschnitt 5 evaluiert wird.

3.3 Bewertungskriterien der Maßnahmen

Um die Performance einer Website bewerten zu können gibt es verschiedene Anhaltspunkte. Die Dateigröße einer Website allein reicht nicht aus, um die Ladegeschwindigkeit beurteilen zu können.

Die in den Tests verwendeten Metriken ergeben sich aus den Ergebnissen des Lighthouse Performance Audits (Google, 2018e) und lauten:

- Server Response Time
- Transferred Bytes
- First Meaningful Paint
- First Interactive
- Consistently Interactive

Die **Server Response Time** beschreibt die Zeit, die der Server benötigt, um die angefragte Ressource auszuliefern (Google, 2018d). Langsame Antwortzeiten können unter anderem

langsamer Anwendungslogik, lansamen Datenbankabfragen oder mangelnder CPU-Power zu Grunde liegen. Google sieht vor die Antwortzeit unter 200ms zu halten (Google, 2018d).

Die **Transferred Bytes** geben die übertragenen Bytes der Website wider und beinhalten alle Ressourcen inklusive der verzögert geladenen Daten, die erst übertragen werden, wenn z.B. durch scrollen, neue Elemente in den Viewport gelangen.

Die Zeit bis der Nutzer das Gefühl hat, dass der primäre Inhalt der Seite sichtbar ist, wird von Google als **First Meaningful Paint** bezeichnet (Google, 2018c).

Nach dem ersten bedeutsamen Anstrich der Website folgt die minimale Interagierbarkeit: **First Interactive** (Google, 2018b). Hier sind die meisten Benutzeroberflächenelemente verwendbar.

Die letzte Phase benennt Google mit **Consistently Interactive** (Google, 2018a). Hier kann der Nutzer mit allen Elementen in vollem Umfang interagieren. Zusätzlich gilt dieser Schritt erst als erreicht, sobald der Main-Thread der Seite alle 50ms die Kontrolle erhält und die Netzwerkübertragungen inaktiv sind (Google, 2018a).

Kapitel 4

Performanceoptimierungen

4.1 Gängige Optimierungsmöglichkeiten

In diesem Kapitel werden gängige Techniken zur Performanceoptimierung aufgelistet und erläutert. Es wird zwischen clientseitigen- und serverseitigen Optimierungen unterteilt.

4.1.1 Clientseitige Optimierungen

Minifying Textfiles
Bei der Minimierung von Textdateien handelt es sich um das Löschen redundanter Informationen aus der Datei, um die Dateigröße zu minimieren. Darunter zählen sowohl Steuerzeichen, die vom Browser nicht interpretiert werden, als auch Kommentare des Entwicklers (Google, 2018f). Um die Minifizierung umzusetzen, gibt es viele verschiedene Möglichkeiten, die auf die jeweilige Entwicklungsumgebung abgestimmt werden müssen.

Reducing Requests
Wie Gash in dem Google Leitfaden über Web-Grundlagen schreibt, ist es wichtig die Anzahl an HTTP-Anfragen zu minimieren (Gash, 2018). Darunter zählt nicht nur das Zusammenfügen von Textdateien wie JavaScript und Stylesheets, sondern auch das Zusammenfügen von grafischen Inhalten. Diese werden Sprite genannt. Verschiedene Bilder in einem Sprite haben folglich die gleiche URL. Das anzuzeigende Bild wird durch die Positionierung des Sprites als CSS-Hintergrund ausgewählt. Gash (Gash, 2018) erwähnt außerdem, dass Sprites in den meisten Fällen nur bei kleineren Bildern effektiv sind, weil hier die zusätzliche prozentuale Netzwerkbelastung am höchsten ist.

Removing unused CSS (CSS Purging)

Wie die GitHub Repositories ”purifycss” (purifycss, 2018) und ”purgecss” (FullHuman, 2018) zeigen, gibt es für diese Methode keinen eindeutigen Namen. Jedoch widmen sie sich dem gleichen Thema: Reduzieren der Dateigröße von CSS-Dateien mittels Erkennung und Löschung von nicht genutzten Selektoren. Hier wird sowohl das Markup auf nicht genutzte CSS-Selektoren untersucht, als auch das externe JavaScript, weil diese auch CSS-Eigenschaften enthalten können. Diese Art der Dateigrößenreduzierung ist gerade bei der Verwendung von großen CSS Frameworks wie Bootstrap nützlich, da nicht immer alle Komponenten genutzt werden.

Lazyloading

Wagner beschreibt Lazyloading als eine Technik, um nicht kritische Ressourcen verzögert zu laden (Wagner, 2018). In der Regel sind das alle Bilder und Videos, die sich nicht im initialen Viewport befinden. Dies betrifft nicht nur die Scrollhöhe, sondern auch Slideshows oder Akkordeons. Für dieses Verfahren gibt es bereits viele Bibliotheken. An dieser Stelle sollte abgewägt werden, welche Lazyloading-Methoden benötigt werden, um die Größe der Bibliothek klein zu halten.

Optimizing Images

Laut Ilya Grigorik machen Bilder in den meisten Fällen den größten Anteil an herunterzuladenden Bytes aus (Grigorik, 2018a). Er weist darauf hin, dass viele gestalterische Effekte auch mit CSS oder Schriftarten zu erzielen sind. Manche Bilder können als Vektorgrafik abgespeichert werden. Eine Vektordatei speichert geometrische Formen ab, während eine Rastergrafik jeden Pixel mit einem Farbwert füllt. Für Bilder mit geometrischen Formen wie Logos, Text und Symbole sollten im Hinblick auf die Dateigröße und Skalierbarkeit Vektorgrafiken herangezogen werden. Vektorgrafiken gehören zu den angesprochenen Textdateien. Auch diese lassen sich also wie in Abschnitt 4.1.1 minifizieren.

Rastergrafiken bestehen aus einem Raster von Pixeln, die Farb- und Transparenzinformationen enthalten. Hier sollte die Qualität verringert werden, um Bytes einzusparen. Dies ist oft auch ohne sichtbaren Verlust realisierbar. Auch in diesem Fall muss das Verfahren an die Entwicklungsumgebung im Projekt angepasst werden. Werden Bilder vom Kunden gepflegt, sollte über eine Komprimierungsautomatisierung nachgedacht werden. Erfolgt die Pflege der Bilder durch den Administrator, können diese je nach Anzahl und Frequenz der Aktualisierungen mit einem externen Programm komprimiert werden, bevor diese zur Website hinzugefügt werden.

Rastergrafiken sollten zusätzlich zur Komprimierung nur in der tatsächlich angezeigten Größe ausgeliefert werden. Das HTML ”srcset”-Attribut bietet die Möglichkeit, abhängig vom Viewport des Clients, verschiedene Bilder zu laden (w3schools, 2018).

Optimizing Critical Path CSS

Um den Inhalt einer Website rendern zu können, muss der Browser zuerst alle Stil- und Layoutinformationen verarbeiten (Google, 2018g). Dies bedeutet, dass das Rendern erst begonnen wird, wenn auch alle externen verlinkten Stylesheets heruntergeladen sind. Diese Verzögerung wird “Render-Blocking-CSS“ genannt. Um diese Blockierung zu vermeiden, wird kritisches CSS in den <head> - Bereich des HTML-Dokumentes geschrieben und das übrige CSS nachgeladen.

Optimizing Web Fonts

Grigorik schreibt in dem Artikel ”Web Font Optimization”, dass Typografie eine grundlegende Voraussetzung für gutes Design, Branding und Lesbarkeit ist (Grigorik, 2018b). Um eine Webfont anzeigen zu können, muss diese vorerst heruntergeladen werden, falls diese nicht bereits gecached ist oder lokal auf dem Client installiert ist. Das Herunterladen der Schriftart verzögert möglicherweise das Rendern. Daher sollte man die Schriftart verzögert laden und nur jene Schriftarten / Schriftvarianten einbinden, die auch genutzt werden.

Serve static content from a cookieless domain

Der Pingdom Website Speed Test bemängelt am Beispiel von DIVE.IS, dass statische Ressourcen von einer Domain geladen werden, auf der Cookies gesetzt sind. (Pingdom, 2018) Dies hat zur Folge, dass jede Ressource in der Regel die nicht benötigten Cookiedaten ebenfalls herunterlädt. Auch wenn die Daten gewöhnlicherweise nur sehr klein sind, können sie sich trotzdem auf einen zweistelligen Kilobyte Betrag aufsummieren. Es gibt je nach Entwicklungsumgebung verschiedene Ansätze, wie man statische Ressourcen ohne Cookie ausliefern kann, ohne ein komplexes CDN einzurichten. Ein möglicher Lösungsansatz ist das Einrichten einer Subdomain, die nur eine Verknüpfung der eigenlichen Website ist, um so Requests identisch von einer anderen Domain zu beantworten.

Use proper Browser Caching

Um wiederholtes Herunterladen von unveränderten Dateien zu vermeiden und die Ladegeschwindigkeit der Website drastisch zu steigern, sollten diese Dateien gecached werden.

Preloading of contents

Es gibt einige Techniken, die es dem Browser ermöglichen Inhalte vorzuladen. Auf diese Techniken geht Torben Leuschner in dem Artikel ”Rasend schnelle Ladezeiten dank Prefetching, Preloading und Prerendering” ein (Leuschner, 2018). Diese Techniken sind laut Leuschner noch unzureichend verbreitet, haben aber keinerlei Nachteile in älteren Browsern. Leuschner listet insgesamt sechs verschiedene Techniken auf:

- DNS-Prefetch

- Preconnect
- Prefetch
- Subresource
- Preload
- Prerender

Der Wert “dns-prefetch“ ermittelt die DNS-Informationen einer Domain. “preconnect“ führt zusätzlich zur DNS-Auflösung noch ein TCP- und TLS-Handshake durch. Die Anweisung “prefetch“ weist den Browser darauf hin eine Ressource herunterzuladen. Die Ausführung dieser Anweisung hängt von verschiedenen Bedingungen ab und kann unter Umständen vom Browser ignoriert werden. “subressource“ referenziert ebenfalls eine Datei, jedoch mit höherer Priorität als “prefetch“. Mittels der “Preload“-Anweisung lässt sich zusätzlich zu “subressource“ noch der Ressourcentyp definieren. Somit kann der Browser entscheiden, welche Ressourcentypen priorisiert werden. Der Wert “prerender“ ermöglicht es eine gesamte URL inklusive verlinkter Ressourcen herunterzuladen.

Optimizing DOM-Nodes

Um Bytes bei der Übertragung zu sparen, sollte versucht werden das Markup so klein wie möglich zu halten. Zusätzlich benötigt die Render-Engine des Browsers mehr Zeit, um mehrere Elemente zu verarbeiten. Gemäß Google können generelle JavaScript Selektoren wie `document.querySelectorAll('li')` große Mengen des Arbeitsspeichers belegen (Google, 2018h).

Optimizing CSS Selectors

Die Optimierung von CSS-Selektoren entlastet die Render-Pipeline des Browsers Čurić (2018). Čurić zeigt auch, dass die Optimierung nach aktuellem Stand der Technik die Performance nur sehr geringfügig verbessert. Dies wird durch den, mittlerweile nicht mehr in den Google Pagespeed Insights vorhandenen, Test im Jahre 2013 unterstützt.

4.1.2 Serverseitige Optimierungen

Transfer Encoding

Es gibt die Möglichkeit, einen Transfer-Encoding Wert an den Client zu schicken. Dieser gibt an, in welcher Form die gesendeten Daten an den Client enkodiert werden (Mozilla, 2018).

Hier können verschiedene Algorithmen zum Einsatz kommen, welche die übertragenen Daten komprimieren und somit Bandbreite einsparen.

Serving static Markup
Das Bereitstellen von statischem Markup kann einen enormen Performanceschub geben. Dies hängt davon ab, wie lange der Server zur Verarbeitung der Anfrage benötigt. Das große Angebot an Static-Site-Generatoren auf `https://www.staticgen.com/` zeigt, dass das Vorgenerieren von Markup ein wichtiges Thema im Bereich Web-Performance ist.

4.2 Maßnahmenübersicht

Die in Abschnitt 4.1 beschriebenen Optimierungsmöglichkeiten werden im Folgenden nach ihrer Rentabilität in Bezug auf Nutzen und Aufwand aufgelistet:

Maßnahme	**Nutzen**	**Aufwand**	**Rentabilität**	**Abhängigkeiten**
Minifying Textfiles	Gering	Gering	Mittel	Build-Prozess
Reducing Requests	Gering	Gering	Mittel	Build-Prozess
Removing unused CSS	Mittel	Mittel	Mittel	Build-Prozess
Lazyloading	Sehr hoch	Gering	Sehr hoch	-
Optimizing Images	Sehr hoch	Mittel	Sehr hoch	-
Optimizing Critical Path CSS	Hoch	Hoch	Mittel	Build-Prozess
Optimizing Web Fonts	Hoch	Sehr gering	Sehr hoch	-
Serve static content from a cookieless domain	Sehr gering	Mittel	Gering	Zusätzliche Subdomains oder CDN
Use proper Browser Caching	Sehr hoch	Gering	Sehr hoch	
Preloading of contents	Sehr hoch	Hoch	Mittel	Kenntnisse über Nutzerverhalten
Optimizing DOM-Nodes	Mittel	Hoch	Mittel	
Optimizing CSS Selectors	Gering	Mittel	Gering	

Maßnahme	**Nutzen**	**Aufwand**	**Rentabilität**	**Abhängigkeiten**
Transfer Encoding	Sehr hoch	Sehr gering	Sehr hoch	
Serving static Markup	Sehr hoch	-	-	

TABELLE 4.1: Maßnahmenübersicht

Im Folgenden werden die Rentabiliätsentscheidungen aus Tabelle 4.1 aufgeführt:

- **Minifying Textfiles:** Die Einsparung von Sonderzeichen reduziert nur geringfügig die Dateigröße. Jedoch werden nicht minifizierte Dateien von Lighthouse bemängelt.
- **Reducing Requests:** Die Einsparung von Requests im Test hat nur zu einem geringfügigen Geschwindigkeitsvorteil geführt.
- **Removing unused CSS:** Dies ist sehr stark abhängig von den nicht genutzten CSS-Anweisungen. Gerade bei der vollständigen Einbindung von großen CSS-Frameworks ist dieses Verfahren sehr effektiv.
- **Lazyloading:** Es werden nicht nur Ressourcen eingespart, sondern auch das Document-Ready-Event wird erst ausgelöst, wenn alle Bilder geladen sind.
- **Optimizing Images:** Jedes eingesparte Kilobyte verkürzt die Ladezeit.
- **Optimizing Critical Path CSS:** Das Extrahieren eines kritischen Teils des CSS gestaltet sich je nach Komplexität des Projektes als schwierig, da jede Seite möglicherweise unterschiedliches CSS benötigt. Zusätzlich ist dieses Verfahren unter Verwendung der HTTP2.0 Push-Methode nicht mehr notwendig. Der Browser muss nicht mehr nach kritischen Ressourcen im DOM suchen, sondern bekommt diese über HTTP2.0 mitgeteilt. (Smashing Magazine, 2017)
- **Preloading of contents:** Die Techniken ”dns-prefetch” und ”preconnect” können mit nur sehr geringem Aufwand implementiert werden, haben aber auch den geringsten Nutzen. Das Vorladen mittels ”prerender” hat den größten Effekt,
- **Optimizing DOM-Nodes:** Das Optimieren der DOM-Nodes ist nur sinnvoll, wenn diese über den empfohlenen Grenzdaten von Google liegen.
- **Optimizing CSS Selectors:** Nur für High-End-Optimierungen lohnenswert, weil die Implementierung sehr aufwendig ist und der Nutzen nur sehr gering ausfällt.
- **Transfer Encoding:** Die Komprimierung kann große Datenmengen einsparen und ist schnell zu implementieren.

- **Serving static Markup:** Der Aufwand zur Generierung von statischem Markup kann sehr stark variieren. Im praktischen Anwendungsfall von Kirby beinhaltet dieses bereits einen Static-Site-Cache. Bei der Generierung von statischem Markup muss sich der Programmierer stehts über clientseitige- und serverseitige Anwendungslogik im Klaren sein.

4.3 Identifizieren und Auflisten von Defiziten

Nach einer Analyse des Entwicklungsstandes im Bezug auf die in Kapitel 4.1 aufgelisteten Optimierungsmaßnahmen ergaben sich folgende Defizite:

- Die übertragene CSS-Datei ist zu groß
- Zu ladende Bilder blockieren das Rendern
- Die im Header eingebundene CSS-Datei blockiert das Rendern der Seite
- Die übertragenen Bilder werden in zu hoher Auflösung ausgeliefert
- Statische Ressourcen werden nicht korrekt gecached
- Übertragene Dateien sind nicht komprimiert
- Es werden nicht genutzte Schriftarten geladen
- Die Antwortzeiten des Servers liegen bei über einer Sekunde

4.4 Maßnahmenevaluation

In diesem Abschnitt werden die aufgelisteten Performancedefizite in Kapitel 4.3 den Optimierungsmöglichkeiten aus Kapitel 4.1 gegenüberstellt.

Defizit	**Maßnahme**
Die übertragene CSS-Datei ist zu groß	Removing unused CSS
Zu ladende Bilder blockieren das Rendern	Lazyloading
Die im Header eingebundene CSS-Datei blockiert das Rendern der Seite	Optimizing Critical Path CSS
Die übertragenen Bilder werden in zu hoher Auflösung ausgeliefert	Optimizing Images

Statische Ressourcen werden nicht korrekt gecached	Use proper Browser Caching
Übertragene Dateien sind nicht komprimiert	Transfer Encoding
Es werden nicht genutzte Schriftarten geladen	Optimizing Web Fonts
Die Antwortzeiten des Servers liegen bei über einer Sekunde	Serving static Markup

TABELLE 4.2: Evaluation der Defizite

Kapitel 5

Prototypische Implementierung der effizientesten Maßnahmen

Die aus der Evaluation in Kapitel 4.4 extrahierten effizientesten Optiomierungsmöglichkeiten sollen prototypisch implementiert werden. Der Code zu den Prototypen kann unter folgender URL abgerufen werden: `https://bitbucket.org/klickmeister/dive-relaunch2017/`.

5.1 Lazyloading

Um das verzögerte Laden von Bildern zu implementieren wurde auf eine Bibliothek zurückgegriffen, die bereits alle notwendigen Funktionen beinhaltet `https://github.com/verlok/lazyload`. Die Bibliothek wurde in vanilla JavaScript geschrieben, sodass keine weiteren Abhängigkeiten bestehen. Die Library wurde mit in den in Kapitel 2 angesprochen GULP-Build-Prozess aufgenommen, sodass diese zur Laufzeit verfügbar ist. Durch das verwendete Atomic-Design-Pattern konnte das Lazyloading auf der gesamten Seite global implementiert werden, ohne mehrere Codestellen abändern zu müssen. Das Image-Atom wurde um eine CSS-Klasse erweitert, die der Lazyloading-Bibliothek als Selektor für verzögert zu ladende Bilder dient. Zusätzlich wurden CSS-Klassen für das Erscheinen von fertig heruntergeladenen Bildern gesetzt.

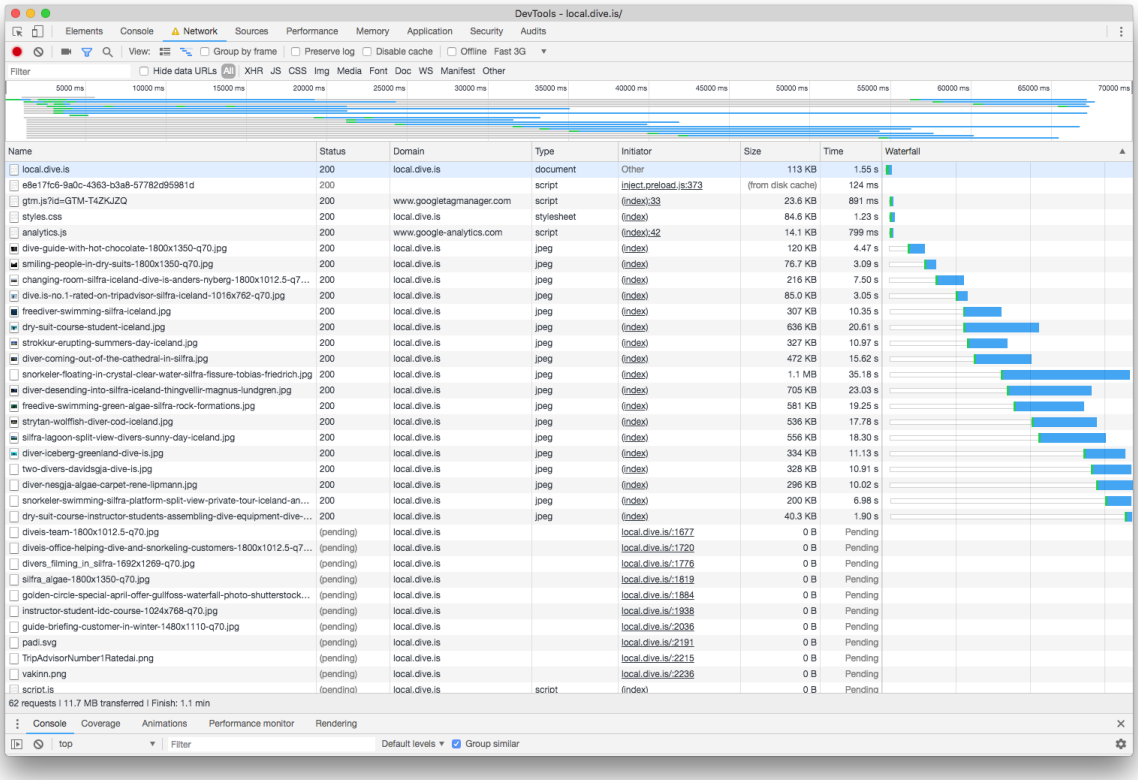

ABBILDUNG 5.1: Netzwerkdiagramm ohne Lazyloading

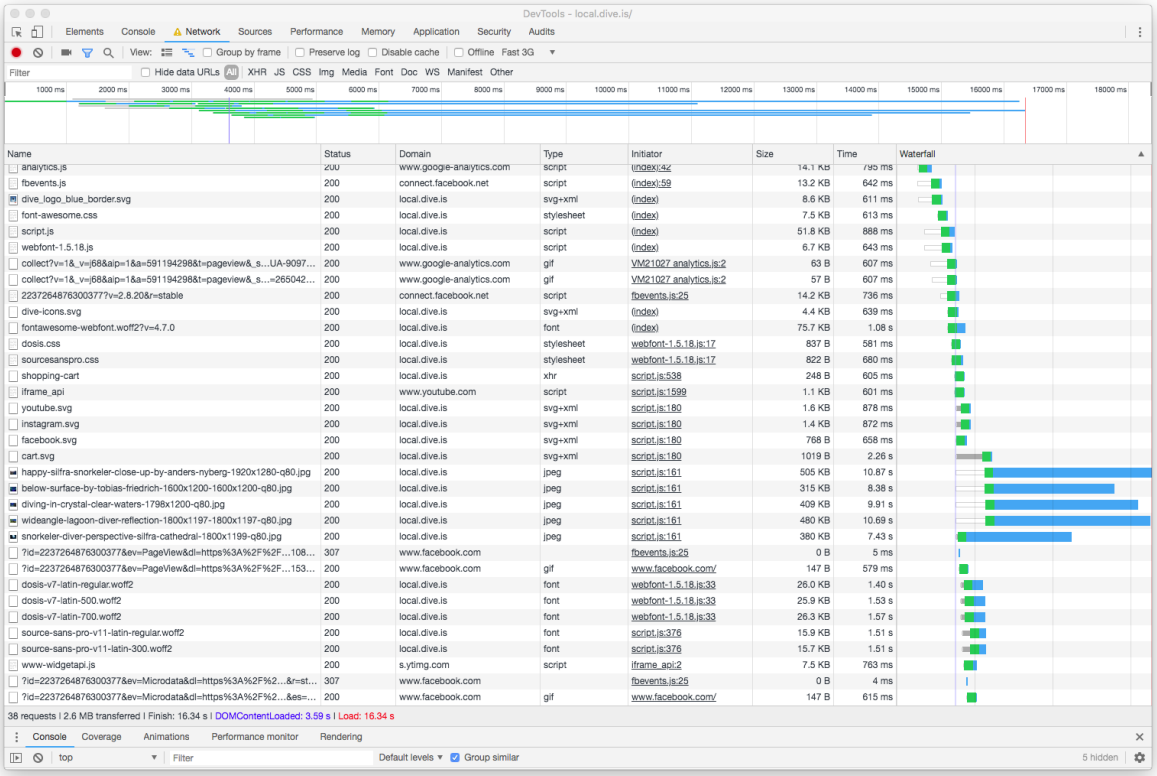

ABBILDUNG 5.2: Netzwerkdiagramm mit Lazyloading

Vergleicht man nun die beiden Netzwerkdiagramme, ist gut zu erkennen wie ohne das Lazyloading alle Bilder gleichzeitig angefragt werden. Dies blockiert nicht nur andere Ressourcen, sondern verzögert auch das ”Document-Ready-Event” und somit den gesamten Seitenaufbau enorm. Lazyloading bei der Verwendung von Bildern ist daher nahezu unerlässlich.

5.2 Optimizing Images

Um Bilder zu optimieren wurde auf die im CMS integrierte Thumbnail-Methode zurückgegriffen. Diese erlaubt es Bilder auf selbstdefinierte Größen zuzuschneiden und die Intensität der Komprimierung zu bestimmen. Zusätzlich wird das erzeugte Thumbnail im Dateisystem abgelegt und zwischengespeichert, sodass dieses nur einmalig bis zur Leerung des Caches generiert werden muss. Um den optimalen Qualitätsparameter zu bestimmen wurde hinsichtlich der absoluten Dateigröße und der relativ eingesparten Dateigröße, eine Tabelle erstellt. Als Datenquellen wurden drei JPEG-Bilder mit unterschiedlicher Farbvarianz in der Größe 1024x768

Pixel ausgewählt.

Anhand der folgenden Grafiken lässt sich sehr gut erkennen, dass die Komprimierung für JPEG-Bilder bei 90 % am effektivsten hinsichtlich Qualitätsverlust und Dateigröße ist. Betrachtungen haben jedoch gezeigt, dass der Qualitätsunterschied zwischen 90 % und 70 % für das menschliche Auge nur im direkten Vergleich erkennbar sind. Somit wurde die Qualität der Komprimierung auf 70 % gesetzt, um die Dateigröße weitestgehend zu reduzieren.

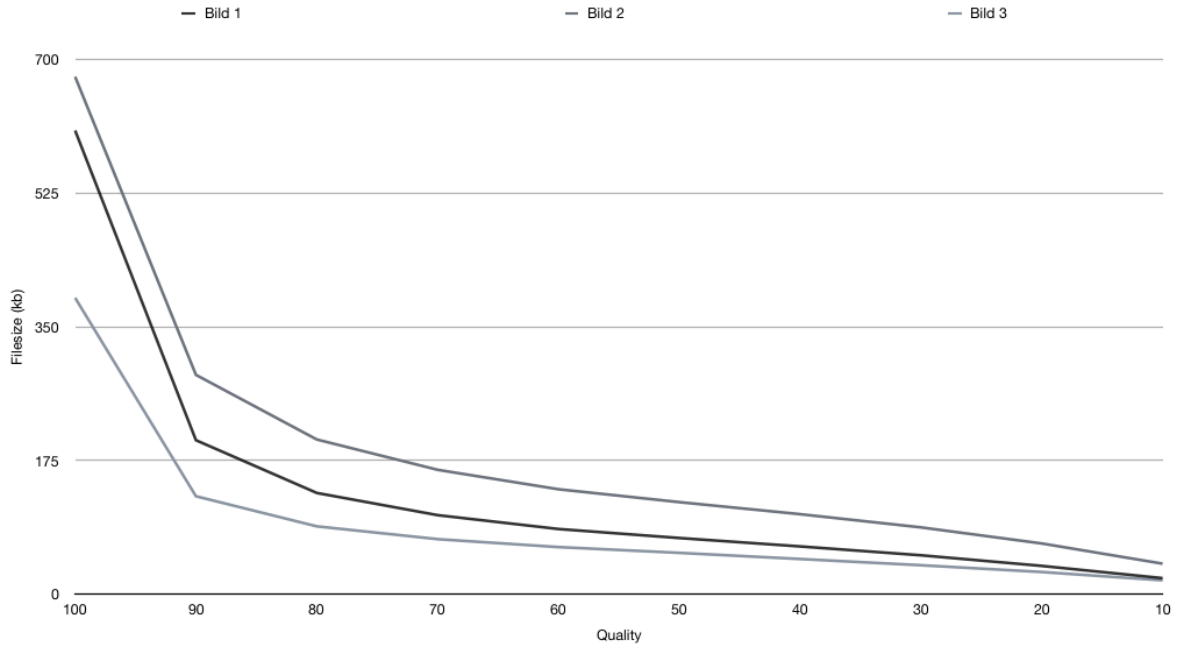

ABBILDUNG 5.3: Diagramm der absoluten Dateigrößen

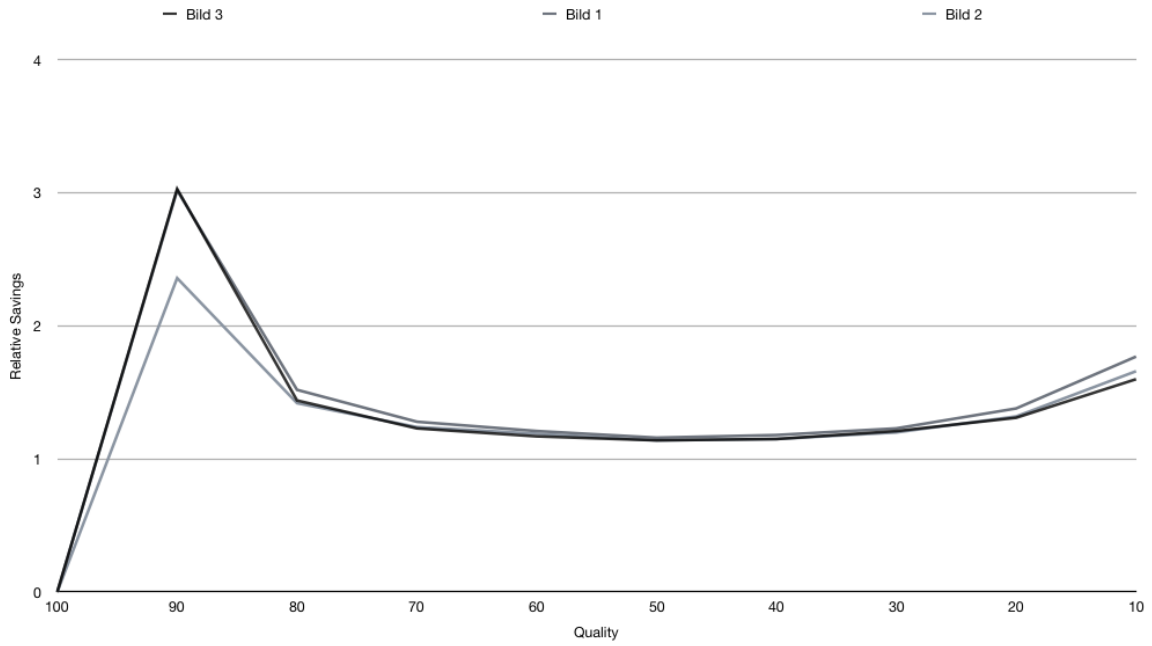

ABBILDUNG 5.4: Diagramm der relativen Einsparung im Vergleich zur vorherigen Qualitätsstufe

5.3 Use proper Browser Caching

Auf Grund der sehr einfachen und schnellen Realisierbarkeit dieser Optimierungsmaßnahme wurden Cache-Header für alle verwendeten Dateitypen in der ".htaccess"-Datei gesetzt. Diese Datei ermöglicht es Konfigurationsänderungen des Apache Webservers pro Verzeichnis vorzunehmen (Apache Software Foundation, 2018). Je nach verwendetem Webserver und Webserver-Konfiguration müssen die Cache-Header ggf. auf eine andere Weise gesetzt werden.

Vergleicht man nun die beiden Grafiken 5.5 und 5.6 wird deutlich, dass das Caching sehr effektiv ist. Der blaue Balken spiegelt die Downloadzeit wieder. Alle cachebaren Dateien die vom DIVE.IS Webserver ausgeliefert werden, haben einen viel kleineren blauen Balken im Wasserfall-Diagramm, weil diese nicht mehr über das Netzwerk geladen werden müssen.

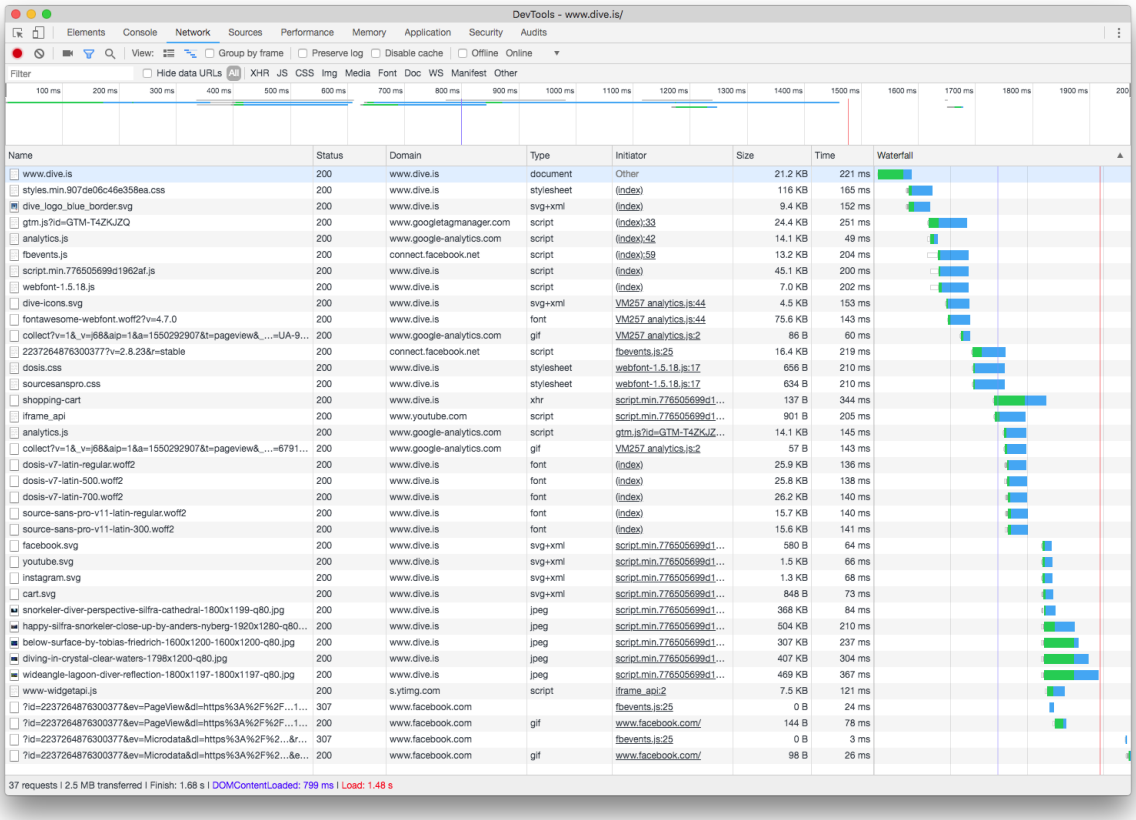

ABBILDUNG 5.5: Netzwerkübersicht ohne Caching

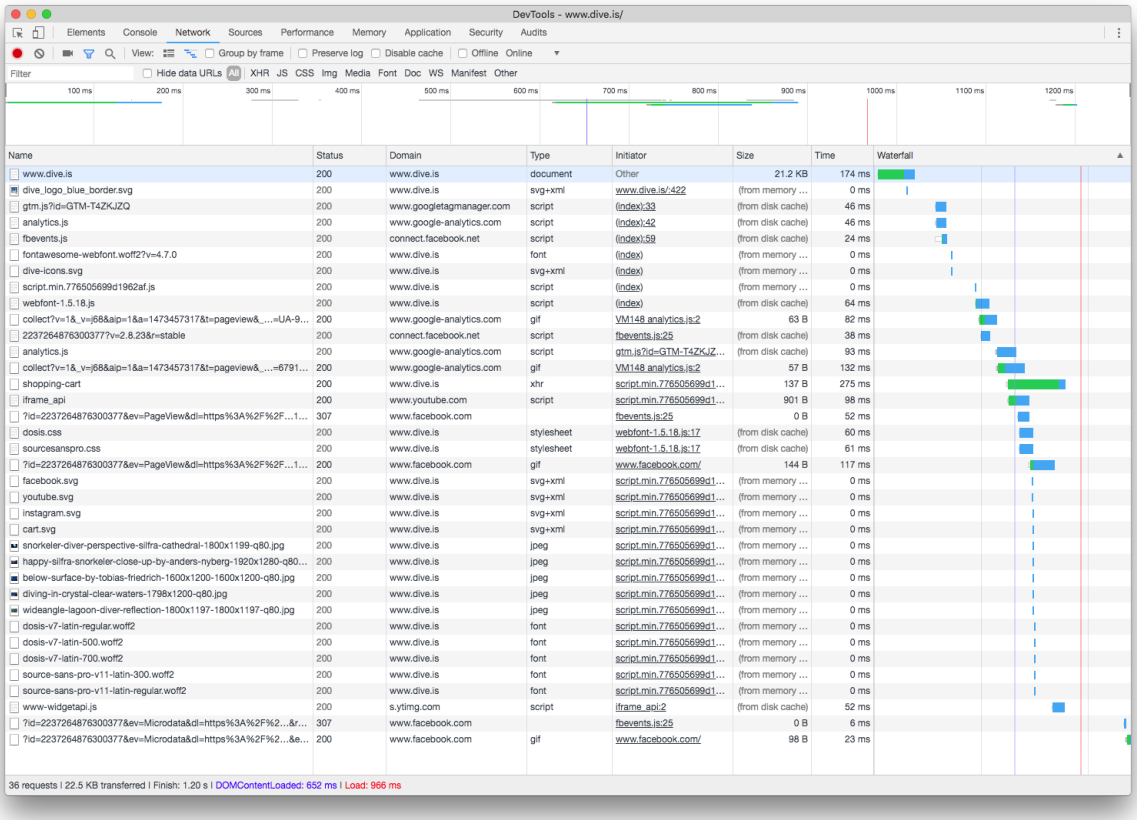

ABBILDUNG 5.6: Netzwerkübersicht mit Caching

5.4 Transfer Encoding

Ebenfalls schnell zu realisieren ist das Enkodieren der gesendeten Datenpakete. Hier wurde die GZIP-Komprimierung aktiviert. Diese wurde ebenfalls in die ”.htaccess”-Datei geschrieben. Der Code kann aus dem in Kapitel 5 genannten Link entnommen werden. Nachfolgende Grafiken 5.7 und 5.8 zeigen, wie effektiv die GZIP-Komprimierung ist. Beispielhaft kann die effektiv übertragene Dateigröße der Stylesheet Datei “styles.css“ betrachtet werden. Diese wurde von 1.0MB auf unter 100KB reduziert.

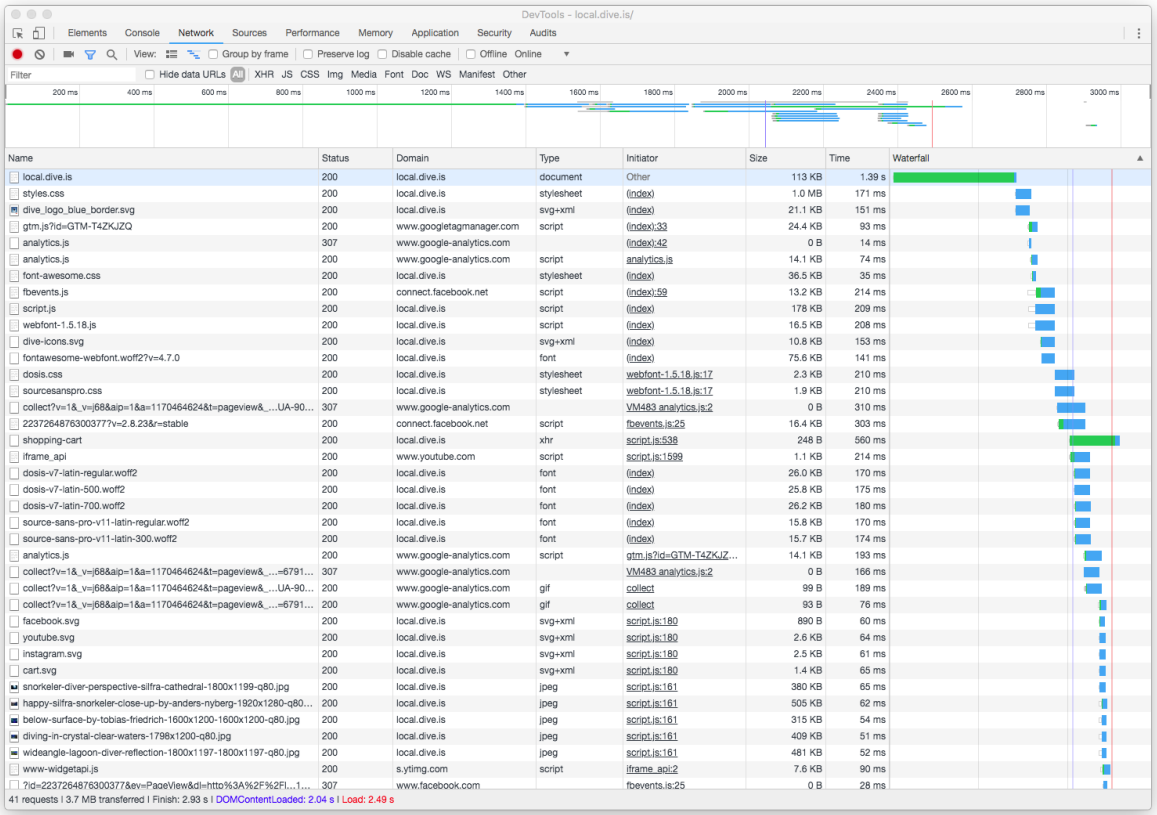

ABBILDUNG 5.7: Netzwerkübersicht ohne GZIP Kodierung

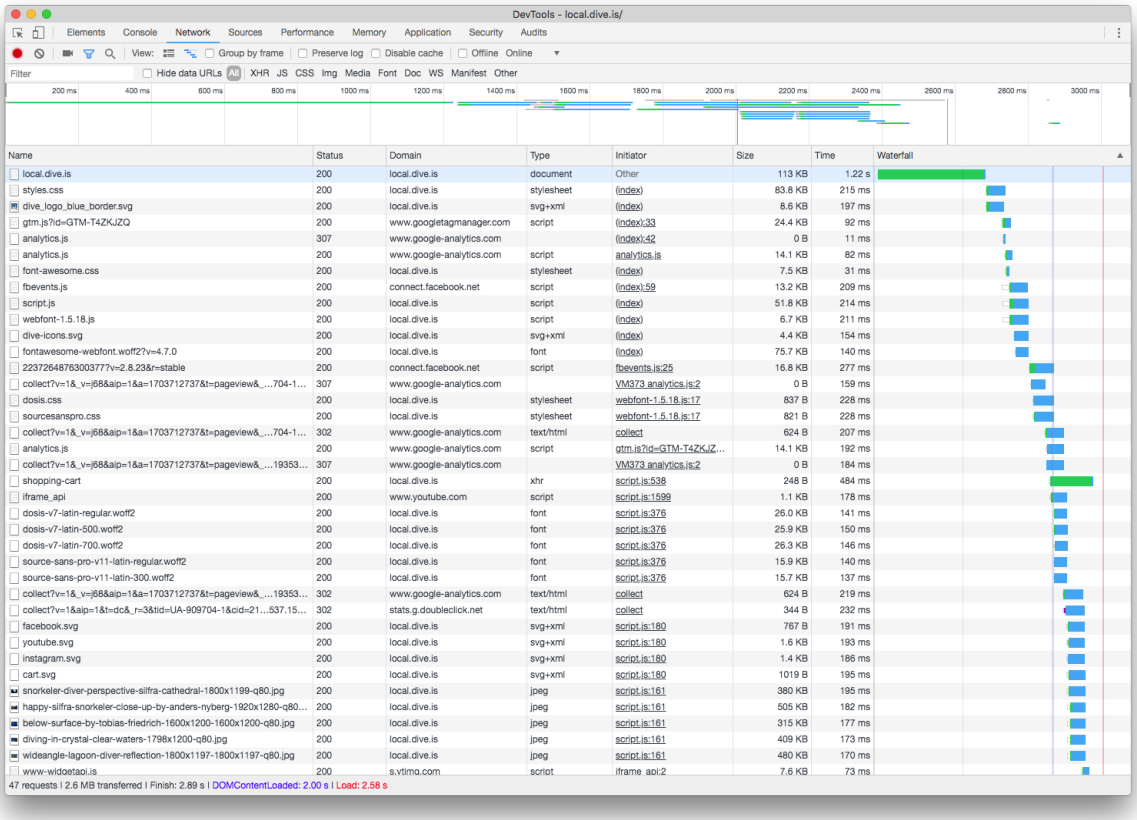

ABBILDUNG 5.8: Netzwerkübersicht mit GZIP Kodierung

5.5 Optimizing Web Fonts

Diese Optimierungsmaßnahme bedarf keiner direkten Programmierung, sondern lediglich einem Überblick über das Projekt. In diesem Projekt wurde nur die Schriftart ”Dosis” in drei verschiedenen Schriftstärken (300, 500 und 700) eingebunden, sowie ”Source Sans Pro” in 300 und 400. Durch das Laden von ausschließlich genutzten Schriftstärken konnten die benötigten Schriftarten mit fünf Requests und einer Gesamtgröße von ~110kb heruntergeladen werden. Zusätzlich werden die Webfonts verzögert via JavaScript geladen, um wichtigere Request nicht zu blockieren.

Kapitel 6

Fazit

In dieser Arbeit ging es darum eine Übersicht von Performanceoptimierungen zu erlangen, die für zukünftige Projekte als Grundlage für Budgetierungsfragen im Bezug auf die Performance einer Website genutzt werden kann.

Um dies zu erreichen wurden gängige Performanceoptimierungen nach ihrer Effizienz aufgelistet und prototypisch in eine bestehende Website implementiert.

Die Messungen der Prototypen haben gezeigt, dass das verzögerte Laden und Optimieren von Bildern unerlässlich für eine performante Website ist. Es wurde deutlich, dass selbst mit schnell zu implementierenden Maßnahmen ein großer Teil der Ladezeit eingespart werden kann. Ebenfalls zeigt diese Arbeit, dass Performanceoptimierung ein iterativer Prozess ist. Eine ideal optimierte Website ist auf Grund ihres asymptotischen Aufwandes in der Praxis kaum zu realisieren.

Eine Benutzeranalyse in Zusammenhang mit den in Kapitel 4.1.1 angesprochenen Techniken des Preloadings könnte Thema zukünftiger Arbeiten/Forschungen sein.

Literaturverzeichnis

Akami (2009). Akamai reveals 2 seconds as the new threshold of acceptability for ecommerce web page response times. http://www.webcitation.org/71ZNOuBKd. Aufgerufen am: 2018-05-10.

Apache Software Foundation (2018). Apache http server tutorial: .htaccess files. https://httpd.apache.org/docs/2.4/howto/htaccess.html. Aufgerufen am: 2018-07-27.

Bókun (2017). About us - changing the course of tourism through innovation. http://bokun.io/about/. Aufgerufen am: 2017-11-06.

Dr. Cordula Heldt, Prof. Dr. Richard Lackes, D. M. S. P. D. W. B. P. D. C. B. (2013). Performance. https://wirtschaftslexikon.gabler.de/definition/performance-46460/version-175614. Aufgerufen am: 2018-05-25.

Enge, E. (2018). Mobile vs desktop usage in 2018: Mobile takes the lead. https://www.stonetemple.com/mobile-vs-desktop-usage-study/.

ferdamalastofa (2018). 2,2 million foreign passengers 2017. https://www.ferdamalastofa.is/en/moya/news/22-million-foreign-passengers-2017. Aufgerufen am: 2018-05-10.

Frost, B. (2018). Atomic design methodology. http://atomicdesign.bradfrost.com/chapter-2/.

FullHuman (2018). Fullhuman/purgecss: Remove unused css. https://github.com/FullHuman/purgecss. Aufgerufen am: 2018-06-25.

Gash, D. (2018). Http request. https://developers.google.com/web/fundamentals/performance/get-started/httprequests-5. Aufgerufen am: 2018-06-25.

Google (2016). Travel trends 2016: Data reveals hot spots and new consumer insights. https://www.thinkwithgoogle.com/advertising-channels/mobile/travel-trends-2016-data-consumer-insights/. Aufgerufen am: 2017-11-28.

Google (2018a). Consistently interactive. https://developers.google.com/web/tools/lighthouse/audits/consistently-interactive. Aufgerufen am: 2018-06-03.

Google (2018b). First interactive. https://developers.google.com/web/tools/lighthouse/audits/first-interactive. Aufgerufen am: 2018-06-03.

Google (2018c). First meaningful paint. https://developers.google.com/web/tools/lighthouse/audits/first-meaningful-paint. Aufgerufen am: 2018-06-03.

Google (2018d). Improve server response time. https://developers.google.com/speed/docs/insights/Server. Aufgerufen am: 2018-06-03.

Google (2018e). Lighthouse. https://developers.google.com/web/tools/lighthouse/. Aufgerufen am: 2018-05-25.

Google (2018f). Minify resources (html, css, and javascript). https://developers.google.com/speed/docs/insights/MinifyResources. Aufgerufen am: 2018-06-25.

Google (2018g). Optimize css delivery. https://developers.google.com/speed/docs/insights/OptimizeCSSDelivery. Aufgerufen am: 2018-06-25.

Google (2018h). Uses an excessive dom size. https://developers.google.com/web/tools/lighthouse/audits/dom-size. Aufgerufen am: 2018-07-11.

GoogleChrome (2018). Googlechrome/lighthouse: Auditing, performance metrics, and best practices for progressive web apps. https://github.com/GoogleChrome/lighthouse. Aufgerufen am: 2018-06-03.

Grigorik, I. (2018a). Bildoptimierung. https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/image-optimization. Aufgerufen am: 2018-06-25.

Grigorik, I. (2018b). Web font optimization. https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/webfont-optimization. Aufgerufen am: 2018-06-25.

Gulp (2018). gulp.js - the streaming build system. https://gulpjs.com/.

Hüfner, D. (2017). Es ist offiziell: Google erhält mobil mehr suchanfragen als von desktops. http://t3n.de/news/google-mobile-suche-anfragen-608634/. Aufgerufen am: 2017-11-28.

Leuschner, T. (2018). Rasend schnelle ladezeiten dank prefetching, preloading und prerendering. https://de.ryte.com/magazine/rasend-schnelle-ladezeiten-dank-prefetching-preloading-und-prerendering. Aufgerufen am: 2018-07-03.

Mozilla (2018). Transfer-encoding - http. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Transfer-Encoding. Aufgerufen am: 2018-07-11.

Pingdom (2018). Website speed test. https://tools.pingdom.com/#!/b0GxMs/dive.is. Aufgerufen am: 2018-07-03.

purifycss (2018). purifycss/purifycss: Remove unused css. also works with single-page apps. https://github.com/purifycss/purifycss. Aufgerufen am: 2018-06-25.

SEO Summary (2018). Suchmaschinen marktanteile in deutschland 2001 - 2018. https://seo-summary.de/suchmaschinen//.

Smashing Magazine (2017). A comprehensive guide to http/2 server push. https://www.smashingmagazine.com/2017/04/guide-http2-server-push/.

Techquickie (2017). Why is hotel wi-fi so slow? https://www.youtube.com/watch?v=84spE6J1G8U. Aufgerufen am: 2017-11-04.

Tripadvisor (2017). Bewertung auf tripadvisor für dive.is. https://www.tripadvisor.de/Attraction_Review-g189970-d1625653-Reviews-DIVE_IS-Reykjavik_Capital_Region.html. Aufgerufen am: 2017-11-06.

w3schools (2018). Html srcset attribute. https://www.w3schools.com/tags/att_source_srcset.asp. Aufgerufen am: 2018-06-25.

Wagner, J. (2018). Lazy loading images and video. https://developers.google.com/web/fundamentals/performance/lazy-loading-guidance/images-and-video/. Aufgerufen am: 2018-06-25.

WordPress (2018a). Requirements | wordpress.org. https://wordpress.org/about/requirements/.

WordPress (2018b). Wordpress plugins. https://wordpress.org/plugins/.

Þór Gústafsson, E. (2017). Removing friction in ux: Last-minute travel planning and activity booking (a case study). https://www.smashingmagazine.com/2017/08/removing-friction-ux-last-minute-travel-planning-activity-booking/. Aufgerufen am: 2017-11-28.

Čurić, I. (2018). Optimizing css: Id selectors and other myths. https://www.sitepoint.com/optimizing-css-id-selectors-and-other-myths/. Aufgerufen am: 2018-07-11.