Über den Zusammenhang zwischen *Ladegeschwindigkeiten von Websiten* und *Absprungraten*

Wissenschaftliches Arbeiten, Übung 4

Julian Krieger

5. Juli 2019

Inhaltsverzeichnis

1 Einleitung

Folgende Ausarbeitung erläutert die Zusammenhänge zwischen der *Ladegeschwindigkeit* von Webseiten im Internet und der *Absprungrate* ihrer Nutzer. Der Begriff Ladegeschwindigkeit (Englisch: *Pagespeed*) bezeichnet den Zeitrahmen, den eine Internetseite benötigt um ihre Inhalte zu laden. Sie steht im umgekehrt proportionalen Verhältnis zur Absprungrate, auch *Bounce Rate* genannt. Diese beschreibt den Anteil der Nutzer, die nur einen einzigen Seitenaufruf erzeugen (Ryte Magazine, 2016).
Achtet man in der Verwaltung einer Webpräsenz nicht auf die Einhaltung von empfohlenen Richtlinien, so lassen sich daraus unmittelbar entstehende Nachteile feststellen. *Amazon* zeigt beispielsweise, dass ihre Verkaufszahlen für jede Erhöhung der Ladezeit um 100 Millisekunden um etwa einen Prozentpunkt sinken (Kohavi & Longbotham, 2007). *Walmart* und *Aberdeen Group* weisen (mit einer Steigerung von zwei Prozent Wechselkurs pro Sekunde Ladezeitsenkung (Wallace, 2016) und sieben Prozent entgangenem Gewinn bei einer einsekündigen Erhöhung (Aberdeen Group, 2008)) ähnliche Werte auf.
Anschließend stellt der Bericht eine konkrete Variante vor, um die Größe einer Website zu minimieren um somit die Ladezeit beziehungsweise das benötigte Datenvolumen zu reduzieren. Im realen Anwendungsfall könnte man so die Absprungrate der Nutzer verringern. Der Fokus liegt dabei in der Verwendung von *Webpack* und ergänzenden Plugins, die über moderne Datenkompressions-Algorithmen den Seiteninhalt konzentrieren. Diese Verfahren minimieren die Anzahl der Informationspakete, die beim Aufrufen einer *URL* zwischen Server und Computer hin- und hergeschickt werden müssen.

2 Hintergrund

2.1 Technische Grundlagen

Der Leser sollte in der Lage sein, die Syntax von modernem *Javascript* im Ansatz zu verstehen. Zusätzlich sollte er mit den Begriffen *CSS* (Cascading Style Sheet), *HTML* (Hypertext Markup Language) und deren Funktionsweise vertraut sein. Vorwissen über die Funktionsweise der *NodeJS* Umgebung und dessen Paketverwaltungswerkzeugen ist hilfreich, aber nicht unbedingt notwendig.
Ich verwende zum Erzeugen der unten aufgeführten Testdaten *Webpack*. Webpack ist ein "Opensource-Javascript-Module-Packer", dessen Aufgabe darin besteht, alle Dateien eines Javascript Projektes zu einer Ressource zu bündeln und dessen Ressourcen zu transformieren (Koppers et al., 2012). Zur Installation und dem Aufrufen von Webpack verwende ich *yarn*, einen Paketmanager und Tool zum Ausführen von Javascript Modulen und eine Alternative zum bekannteren Programm **npm**.

2.2 Problemstellung

Je höher der Zeitpunkt vom Auswählen einer Webseite zur Benutzbarkeit, desto größer ist die Versuchung für den durchschnittlichen Nutzer, den Ladevorgang vorzeitig abzubrechen.
So beträgt die durchschnittliche Ladezeit einer Website 2018 auf Desktop Computern etwa 8.6 Sekunden. Diese Zahl erklärt sich zum Teil durch hohe Seitengewichte. Im Schnitt beträgt die Größe einer Website zum Messungszeitpunkt 1.8 Megabyte. Zum Vergleich ordnet das von Google entwickelte Werkzeug zum Messen von Seitenleistungen, *PageSpeed Insights*, eine Regelladezeit von unter 2 Sekunden als "Durchschnittlich" ein (Google, 2018).
Im Bereich des mobilen Internets fallen durch verschiedene Studien gemessene Kennzahlen noch höher aus. Beispielsweise zeigt eine Messung von Google & DoubleClick AdExchange (2016), dass eine durchschnittliche Website beim Aufruf über das *3G* Netzwerk 19 Sekunden braucht, bevor sie bedient werden kann. Zudem deutet sie darauf hin, dass über die Hälfte der Nutzer den Ladevorgang frühzeitig beendet, wenn dieser nicht nach 3 Sekunden abgeschlossen ist. Die Erwartungen von Endverbrauchern scheinen zudem zu steigen. Während eine Erhebung von Akamai & JupiterResearch (2006) 2006 noch 4 Sekunden als Maximalen Wert festlegt, den ein Online-Shopper vor dem frühzeitigem Verlassen der Seite wartet, aktualisierten die beiden amerikanischen Unternehmen diese Kennzahl in einer Wiederholungsstudie 2009 auf 2 Sekunden (Akamai & JupiterResearch, 2009). Diese Messwerte stehen im Gegensatz zur Entwicklung des durchschnittlichen Gewichtes einer Internetseite über die Jahre.

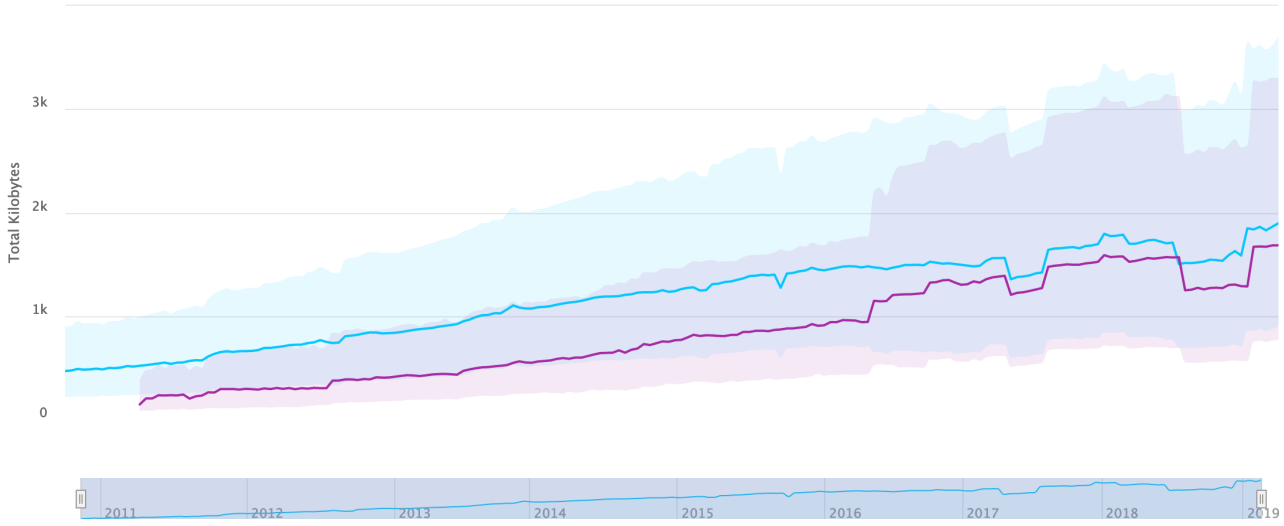

Figure 1: httparchive.org zeigt das Wachstum der Seitengröße von 2012 bis 2019

Ziel ist also die Vorstellung einer Lösungsvariante, um den Umfang der Daten einer Website wesentlich zu minimieren.

2.3 Bestehende Lösungsansätze zur Senkung von Ladezeiten

Ein Ansatz ist das Verwenden eines sogenannten *CDN* (Content Delivery Network). Dieses beschreibt eine Gruppe von international verteilten Servern, die einen schnellen Transfer von (Assets) garantieren. Der dem Endnutzer geografisch am nächsten stehende Server ist in der Lage, dem Endnutzer eine Kopie der hinterlegten Daten auf ökonomische Art und Weise zukommen zu lassen (Cloudflare, 2017). Eine weitere Möglichkeit besteht im Einsatz von *Caching* Methoden, welche Seiteninhalte zwischenpeichern und so die Ladezeit beim erneuten Anfordern von Daten reduzieren.

3 Methode

3.1 Lösungsansatz

Der vorgestellte Lösungsansatz orientiert sich an der Verwendung von *Webpack* im Zusammenspiel mit den Komprimierungsalgorithmen *mozjpg* für JPGs, *optipng* und *pngquant* für PNG Dateien und *gifsicle* für GIFs. Zudem werden die Webpack Plugins *html-loader*, *css-loader*, *OptimizeCSSAssetsPlugin* und *TerserJSPlugin* verwendet, um HTML, CSS und Javascript Code zu minifizieren. Der Vorgang der Minifizierung beschreibt den Vorgang des Löschens von Leerzeichen, Tabs und doppelt vorhandenen Elementen. Unterstützt werden die oben genannten Plugins durch die *file-loader*-Erweiterung im Zusammenspiel mit *image-webpack-loader*. Zusätzlich verwende ich das Zusatzpaket *Bundle-Analyzer-Plugin*, um das von Webpack generierte Bündel zu analysieren. Letzendlich ergänze ich den Vorgang mit *Babel*, welches den Programmcode so übersetzt, dass er auch von alten Browsern genutzt werden kann.

3.2 Implementierung

Zu Testzwecken werden diese oben genannten Einstellungen in der Webpack Konfigurationsdatei **webpack.config.js** wie folgt initialisiert:

```
[...]
 module: {
   rules: [
     {
       test: /\.html$/, //Pack and minify html
       use: {
         loader: 'html-loader',
         options: {
           minimize: true, //Minimiere HTML
           interpolate: true, //Erlaube das Interpolieren von Bilder in HTML
           removeComments: true, //Entferne Kommentare
           collapseWhitespace: true, //Entferne Leerzeichen und Tabs, soweit m glich
```

```
        },
      },
    },
    {
      test: /\.(sa|sc|c)ss$/, //Packe und Minifiziere CSS
      use: [
        {
          loader: MiniCssExtractPlugin.loader, //Injiziere CSS in die Seite
        },
        {
          loader: 'css-loader', //L dt CSS als Javascript Module
          options: {
            import: true,
          },
        }
      ],
    },
    {
      test: /\.js$/, //Packe und Minifiziere Javascript
      exclude: /node_modules/,
      use: [
        {
          loader: 'babel-loader', //Erlaube das Transpilieren von Javascript f r   ltere
  Browser
          options: {
            rootMode: 'upward',
          },
        }
      ],
    },
    {
      test: /\.(png|svg|jpg|gif)$/, //Minifiziere png, svg, jpg, gif
      exclude: /repo_assets/,
      use: [
        {
          loader: 'file-loader',
          options: {
            name: '[hash].[ext]',
          },
        },
        {
          loader: 'image-webpack-loader', //Kompressionsalgorithmen
          options: {
            mozjpeg: {
              progressive: true,
              quality: 65
            },
            optipng: {
              enabled: false,
            },
            pngquant: {
              quality: '65-90',
              speed: 4
            },
            gifsicle: {
              interlaced: false,
            }
          }
        },
      ],
    },
  ],
},
plugins: [
  new HtmlWebPackPlugin({
    title: 'index',
    inject: true,
    template: path.resolve('./src/index.html'),
    filename: path.resolve('./dist/index.html'),
  }),
  new MiniCssExtractPlugin({
    filename: '[hash].css',
    chunkFilename: '[hash].css',
```

```
    }),
    new BundleAnalyzerPlugin()
  ],
  [...]
  optimization: {
    minimizer: [new TerserJSPlugin({}), new OptimizeCSSAssetsPlugin({})],
  }
```

In diesem Versuch nehmen wir einen Projektaufbau bestehend aus einer `index.html` Datei, welche drei `<img>` Bildelemente enthält und einer `index.js` Datei, die per `document.getElementById()` die `src` Attribute der Bildelemente auf den Pfad zu den jeweiligen Abbildungen setzt. Diese sind zur verbesserten Darstellung der verschiedenen Kompressionsalgorithmen verhältnismaßig groß gewählt.

Name	Größe
bigjpg.jpg	37MB
bigpng.png	9.4M
giphy.gif	1M

Table 1: Die verwendeten Bilder und ihre Größen

Ein Aufruf mit dem Kommando `yarn run build` lässt das Webpack Programm laufen. Dieses bündelt alle Bilder und Code Dateien zusammen, und erzeugt daraus optimierte und komprimierte Pakete. Das Bundle-Analyzer-Plugin erzeugt durch die Option `generateStatsFile: true` in **webpack.config.js** eine Datei mit Statistiken des erzeugten Bündels. Als nächstes erzeugen wir zum Vergleich drei verschiedene Pakete:

Index	Name	Code* Größe	Bilder Größe	Gesamt
1	Webpack mit Kompression und Minifizierung	8KB	5.8MB	5.9MB
2	Webpack ohne Kompression	8KB	47.4MB	47.4 MB
3	Statische Seite ohne Webpack	16KB	47.5B	47.4 MB

*Table 2: *index.html, index.js und page.css*

Webpack erstellt Paket 1 mit oben gezeigter Konfiguration. Das Entfernen der Kompressionsalgorithmen führt zu Paket 2. Paket 3 beschreibt die Quelldateien als statische Seite ohne Komprimierung und Minifizierung.

4 Evaluation

Die gezeigte Lösungsvariante funktioniert unter der Voraussetzung, das der Computer, auf dem sie eingesetzt wird, die Benutzung von Javascript ermöglicht, was zum Teil auf älteren Systemen oder bei hohen Sicherheitsvorgaben nicht immer der Fall ist.
Im Gegensatz zu einem in 2.3 vorgestellten CDN ist der Einsatz von Webpack und Komprimirungsstrategien weitreichend kostenlos und skaliert direkt mit der Anzahl und Größe der auf der Webseite eingesetzten Medien. Bei einer zusätzlichen Verwendung hochmoderner Minimierungsalgorithmen wie Googles *Zopfli* muss man jedoch darauf achten, dass die Kosten des prozessorintensiven Vorgangs (Vandevenne, 2013) der (De-) Komprimierung nicht den Nutzen übersteigen.

5 Fazit

Anhand der oben gezeigten Daten ist sichtbar, dass die Größe der Daten der im Fallbeispiel erzeugten, einfachen Internetseite wesentlich reduziert werden konnten. Der Inhalt der Code Dateien konnte um 8 Kilobyte reduziert werden. Während dieser Vorgang im Fallbeispiel relativ klein erscheint, ist er beim Aufbau einer realistischen Website mit großen CSS Dateien und mehreren Javascript Abhängigkeiten durchaus relevant. Zusätzlich entstand bei der Verwendung von geeigneten Kompressionsalgorithmen ein deutlich messbarer Effekt. Der Umfang der Bilddateien konnte um 42 Megabyte reduziert werden. Da Bilder und Grafiken heutzutage über 20 Prozent des Datenverkehrs ausmachen (Internet Archive, 2018), könnte man Bildkomprimierungsprozesse in der modernen Webentwicklung als unbedingt notwendig bezeichnen.

6 Ressourcen

Die in Abbildung 2 gezeigten Werte wurden durch drei verschiedene Pakete erzeugt. Eine vollständige Implementierung dieser befindet sich auf **https://github.com/juliankrieger/Wissenschaftliches-Arbeiten-4**. Die Messwerte im Index 1 können zusammen mit ihrer Umsetzung auf dem **master** Branch gefunden werden. Indizes 2 und 3 liegen jeweils auf den Branches **no-compress-wp** und **static-no-webpack**.

Referenzen

Aberdeen Group. (2008). *The performance of web applications: Customers are won or lost in one second.* Retrieved 2019-7-05, from https://web.archive.org/web/20180316082024/http://www.aberdeen.com/research/5136/ra-performance-web-application/content.aspx

Akamai, & JupiterResearch. (2006). *Akamai and JupiterResearch identify web page response times| akamai.* Retrieved 2019-07-04, from https://www.akamai.com/uk/en/about/news/press/2006-press/akamai-and-jupiterresearch-identify-4-seconds-as-the-new-threshold-of-acceptability-for-retail-web-page-response-times.jsp

Akamai, & JupiterResearch. (2009).

Cloudflare. (2017). *What is a CDN? how does a CDN work?* Retrieved 2019-07-04, from https://www.cloudflare.com/learning/cdn/what-is-a-cdn/

Google. (2018). *About pagespeed insights.* Retrieved 2019-07-05, from https://developers.google.com/speed/docs/insights/v5/about

Google, & DoubleClick AdExchange. (2016). *Mobile load time and user abandonment.* Retrieved 2019-07-05, from https://developer.akamai.com/blog/2016/09/14/mobile-load-time-user-abandonment

Internet Archive. (2018). *Page weight.* Retrieved 2019-07-05, from https://httparchive.org/reports/page-weight

Kohavi, R., & Longbotham, R. (2007). Online experiments: Lessons learned. , *40*(9), 103–105. Retrieved 2019-07-04, from http://ieeexplore.ieee.org/document/4302627/ doi: 10.1109/MC.2007.328

Koppers, T., Larkin, S., & Johannes, E. (2012). *Webpack.com.* Retrieved 2019-07-05, from https://webpack.js.org/

Ryte Magazine. (2016). *Was ist eine bounce rate? - ryte wiki.* Retrieved 2019-07-04, from https://de.ryte.com/wiki/Bounce_Rate

Vandevenne, L. (2013). *Compress data more densely with zopfli.* Retrieved from https://developers.googleblog.com/2013/02/compress-data-more-densely-with-zopfli.html

Wallace, D. (2016). *How page load time can impact conversions [infographic].* Retrieved 2019-07-04, from https://infographicjournal.com/how-page-load-time-can-impact-conversions/