Web performance optimization

- How is an optimized front-end achieved?

LUNDS
UNIVERSITET
Lunds Tekniska Högskola

LTH School of Engineering at Campus Helsingborg

Department of Electrical and Information Technology

Bachelor thesis:
Robin Török
Sebastian Johansson

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
E-huset
Biblioteksdirektionen
Lunds universitet
Lund 2014

Abstract

The objective of this thesis was to gather information about how important web performance is and what to do to optimize web pages. This information will then be delivered as course materials to Edument AB. The course materials will be presentation materials that are going to be a part of their upcoming course about this topic.

The presentation materials were divided into eight parts in order to achieve a natural distribution between the areas in the topic. Our ambition was to finally provide well-elaborated presentation materials, which will not only contain theoretic but also practical examples. This was done by splitting up the time into four working phases for each part of the presentation materials.

During the first working phase a non-optimized example was tested in order for the bottleneck to be found and analysed.

During the second working phase the example was optimized, focusing on the bottleneck and to finally eliminate it.

The third was to test the optimized example and compare for further analysis.

The last phase was to document the conclusion of the tests made.

Keywords: HTML, CSS, JavaScript, Caching, Http, Web performance

Sammanfattning

Examensarbetet gick ut på att få information om varför det är viktigt med webbprestanda samt hur man optimerar hemsidor för bästa prestanda. Denna information skulle levereras som kursmaterial till Edument AB. Kursmaterialet kommer utgöra presentationsmaterial för deras kommande kurs angående ämnet.

Presentationsmaterialet delades upp i åtta delar för att få en naturlig fördelning mellan de olika områdena inom ämnet. Ambitionen var att slutligen få fram ett väl genomarbetat presentationsmaterial, som förutom teori även skulle innehålla praktiska exempel. Detta gjordes genom att dela upp tiden i fyra arbetsfaser för varje del av presentationsmaterialet.

Första fasen gick ut på att testa ett icke optimerat exempel där flaskhalsen skulle hittas och analyseras.

Under andra fasen optimerades exemplet med avseende på flaskhalsen och slutligen eliminera den.

Den tredje var att testa det optimerade exemplet och jämföra för vidare analys.

Den sista fasen var att dokumentera slutsatsen av de gjorda testerna.

Nyckelord: HTML, CSS, JavaScript, Caching, Http, Webbprestanda

Foreword

We would like to thank Edument AB for the opportunity to do this thesis. A special thanks to Tore Nestenius, co-founder of Edument AB, for the guidance and support throughout the thesis. We would also like to give a special thanks to Christian Nyberg, our examiner at LTH School of Engineering.

List of contents

1 Introduction

1.1 Background

This thesis is built on Edument AB's idea about creating a course about web performance. Edument was founded in 2010 by Acke Selem and Tore Nestenius. Edument is a software development and mentoring company specializing in the most challenging parts of software development . The employed consultants acts as mentors in other companies to make their development more effective. As well as offering mentoring, Edument also holds lectures about different subjects related to IT.

The work presented in this thesis will be a part of an upcoming course about web performance. The course will include seven hours of lecturing and demonstrations as well as seven hours of performance related exercises. The purpose of the course is to teach the basics about web performance such as finding the bottleneck and which aspects and resources that matters when building a high performance website. The thesis includes the presentation materials that will be used in the lectures.

1.2 Purpose

While a decade ago, it was acceptable for a web site to take a while to load, it today has become more important than ever to have a fast web site. Especially if the web site's intention is to generate revenue. According to recent studies, twenty percent of the visitors will abandon a webpage slower than 3 seconds, which directly affects the revenues. (Strangeloop Networks, 2012)

The main purpose of this thesis is to give information of how optimization of different aspects can be done, as well as present how the different optimization methods can save the overall time spent on downloading a web site. These areas will be represented in the presentation materials delivered to Edument.

1.3 Problem

The problems discussed and answered in this thesis are listed in section 1.3.1 and 1.3.2. They are divided into three main problems. These questions are broad and not easy to answer. Therefore they are divided into six smaller more concrete subquestions. A summarize of answers of these subquestions is presented in chapter 6.

1.3.1 Main problems

- How can the browser's performance be optimized, from the user's perspective, focusing on HTML, CSS, JavaScript and web services?

- How can web performance optimization and load testing be done and how is it analysed?
- How can the problem of web performance be handled?

1.3.2 Subproblems

- How do HTTP, HTML, CSS, JavaScript and images affect performance?
- What tools can be used to properly measure the web performance and how is the result interpreted?
- Can web optimization be automated?
- How should a company proceed to optimize it's already exsisting web site?
- How can caching be implemented properly and what are the pitfalls?
- Can a checklist be developed for guidence on how to optimize a web site?

1.4 Limitations

Because of the limited time, there had to be some limitations. One limitation was not to focus on code optimization. Although an effective code is important, it was not as important as the topics included in this thesis. The theoretical background of how it is possible to make changes to the code is mentioned, but this is not tested and thereby not presented in the results.

Another limitation of this thesis was the mobile platform. Most of the optimization methods in the thesis can also be applied on smartphones and tablets, but the focus was set on the desktop.

Optimization of TCP/IP and networks is not included in this thesis.

2 Technical Background

The aspects being handled in terms of web performance optimization are divided into two parts. The first part is how the web including browsers, the http protocol and different techniques affect the performance of certain web sites. The second is how the performance is affected by the resources used to build a web site.

2.1 Web

One of the ways the web perfomance can be affected is by the interaction between servers. The web, in our case refers to how the communication is handled, in order to achieve as minimal communication times as possible.

2.1.1 Web browsers

The web browser is a software application that is the interface between a user and the web. The web browser's function is to handle the data being sent and received trough the Http protocol. The way the browser handles the information about a web site is by the HTML document. The document is transformed into a DOM (Document Object Model) tree in order for the web browser to be able to read and render the web sites more efficiently.

Alongside with the DOM the browser's second function is to download all the resources into the internal memory, this will serve the rendering and functionalities needed by the webpage that is being shown.

The web browser is often a bottleneck in terms of optimization. It was never meant for it to be used for advanced applications such as Facebook and similar, which handle big amount of data.

When a browser downloads a web page it downloads different resources in parallel. This parallelism requires more CPU capacity and bandwidth. To be able to get the most out of it, and minimize the handling time for a better user experience, it is important to know how many parallel downloads is optimal for a certain web site. Optimizing the browser handling is being aggravated for developers because of the wide assortment of different browsers being used today. (Souders, S. 2007; Souders, S. 2009)

2.1.2 Http

Http is a protocol used by servers and computers to communicate with each other. When there is a connection established after DNS Lookup, information is shared. This information could, for instance, be a web site. Due to the fact that a website often contains multiple resources, some of which contains a relatively big amount of data, the content is divided into smaller Http packages.

When a client needs data from a server a request, containing information of what data is needed, is made. The respondent server then sends back a response with a payload containing the data being requested earlier, if this is permitted.

Since web sites nowadays have been growing tremendously in size, there is need for reduction of requests as well as the size of the payload sent back. The first point is especially important, the time spent on sending data back and forth often constitute the biggest part of the web site loading time. (Simonstl 2009)

2.1.3 Caching

When communicating across the web it is in everyone's interest to reduce the number of requests and bytes sent/received. When a user visits a web site that consists of multiple pages, the habit is often to visit the succeeding pages. The web site often reuses some content throughout the pages, which makes the number of requested resources fewer. To be able to reuse resources the web browser contains a local cache memory. The cache gives the ability for a client to load the reused content located in the cache and thereby reduce the number of Http requests. Due to the request not being sent, the time for retrieving needed data is reduced. Caching data is an important part of optimization when it comes to static content such as images, style sheets and scripts. (GTmetrix n.d; Souders, S. 2007)

2.1.4 CDN (Content Delivery Network)

A content delivery network is a network of servers spread across a region to serve people who own web sites or simply are storing data on a server. A CDN is providing a service that is minimizing the geographical distance between clients and hosts. The goal is to reduce the propagation time for Http requests/responses. Instead of one host storing all the data needed for the web site on a single entity, a good practice is to spread this data across a CDN network. Clients will then be able to retrieve data from the web site faster, no matter where the client is located in the world. (Souders, S. 2007; White, J. 2012)

For example a web site with users across the entire world can greatly benefit from using a CDN, while local news web site perhaps cannot. The CDN is mainly used to store static content. This is a service hosted by larger companies and can be both free and commercial. (Souders, S. 2007; White, J. 2012)

2.2 Resources

When visiting a web site, multiple resources are downloaded. These resources serve different purposes but works together to build a web site. The following section will introduce the most common web site resources.

2.2.1 HTML (HyperText Markup Language)

HTML is the heart of web sites and is the standard language used when constructing a web page. The HTML document consists of tags enclosed in angle brackets. Most of the elements in HTML have a starting tag, such as `<div>` with an associated closing tag, `</div>`. Figure 2.1 describes an example of a simple HTML document. The document is built like a tree where a tag can have multiple child tags which lies within their parent.

```
<html>
    <head>
        <title>My title</title>
    </head>
    <body>
        <h1>My header</h1>
        <p>My paragraph</p>
    </body>
</html>
```

Figure 2.1: A very simple HTML document.

A tag can have attributes that provide it with properties. An image has a source attribute which leads to the path of the image, a div can have an ID to make it possible find it using style sheets and scripts. (W3Schools n.d.a)

There are different versions of HTML. While HTML4, or XHTML, has been the most common in the last decade, HTML5 is becoming more popular. The advantages of HTML5 are, among others, a simplified syntax, more powerful elements like `<header>`, `<footer>`, and `<menu>` and the multimedia tags `<audio>` and `<video>`. While these advantages can be attractive for a web developer, they should be used with caution. Not all browsers support HTML5, the older ones in particular, which can lead to compatibility issues for some of the viewers. (Minnick, C. 2013)

When a web browser downloads a HTML document it parses its data, render a DOM tree and looks for further resources to be downloaded. These resources can be style sheets, scripts and images, which either style the HTML or give it dynamic functionality that can enhance the user experience. (W3Schools n.d.a)

2.2.2 CSS (Cascading Style Sheet)

The CSS is a style sheet that describes the design of a web page and is used widely by web developers. Figure 2.2 shows an example on how a style sheet can look. It uses selectors to find elements in the HTML document to change their visual properties. In Table 2.1 you can see some selectors and examples on how they can be used. (W3Schools n.d.b)

```
body {
    background-color: orange;
}

#my_id {
    color: green;
}

h1 {
    font-size: 16px;
}
```

Figure 2.2: A simple style sheet.

Selector	**Example**
ID	`#my_id {background-color: white;}`
Class	`.my_class {color: black;}`
Type	`A {text-decoration: none;}`
Adjacent sibling	`H3 + #my_id {padding:10px;}`
Child	`#my_id > LI {font-size: 12px;}`
Descendant	`#my_id  A {color: #333;}`
Attribute	`[href="link.html"]{font-weight:bold;}`
Pseudo classes/elements	`A:hover {text-decoration: underline;}`

Table 2.1: CSS selectors

Some of these selectors are bad for performance. Using for example a descendant selector such as `#my_id a {color: #333;}` makes the browser search for all elements with the type `a` and then look for the id `my_id` through traversing the HTML tree until it has been found. Simply by

using a better ID or classes you can still do exactly the same thing without loosing time on finding the element. (Souders, S. 2009; W3Schools n.d.b)

A web designer may keep their CSS well commented and divided in multiple files for best maintainability and overview. However, comments and unnecessary spaces quickly add up to the file size which leads to a longer download time when a browser requests it. By removing spaces and comments in the CSS, you get a smaller file and a faster download. (Souders, S. 2009)

The more requests you make, the more time you spend on waiting for the server to prepare your response. By combining the style sheets you get fewer request on the same information.

If you want your design to change dynamically you can use something called CSS expressions. By using these you can implement scripts in your style sheet to change the design depending on events and similar. The problem with expressions is that the page is evaluated every time you move your mouse, scrolls the page, resize the window and so on. This can lead to a sluggish site, which lowers the user experience. (Souders, S. 2009)

While it is possible to write both internal and inline CSS inside the HTML document, it cannot be cached. Making it external makes the CSS more likely to being cached. (Souders, S. 2009)

2.2.3 JavaScript

JavaScript is a dynamic programming language used to enable the browser to execute scripts on client-side to interact with the HTML and CSS and dynamically change these. It can also be used to communicate asynchronously with servers without reloading the page.

JavaScript uses functions such as `document.getElementById(…)` to retrieve and alter elements from HTML. When the script interacts with the browser in this way it needs to access the DOM. The interaction between JavaScript and the DOM is a bottleneck in terms of web performance. Every time a script calls a function that interacts with the DOM, the DOM tree is traversed until the script finds the element. With many elements the tree becomes large and complex, which means traversing it becomes time consuming. Reducing the number of DOM elements by removing unnecessary tags in HTML both reduces the file size of the HTML document as well as making traversing the tree faster.

Changing the layout and design of a page using JavaScript makes the browser repaint the page. Doing this often leads to a sluggish page. The number of repaints can be reduced by, instead of changing the styling attributes, change the class of an element and thereby only repainting once.

JavaScript uses events such as onLoad and onClick. The HTML code `<button onClick="doSomething()"> Button </button>` will lead to that the JavaScript function `doSomething()` is executed whenever the button is clicked. Having a lot of these kinds of events on your site may lead to a poor user experience because of the added complexity. It also leads to a larger file, which directly affect the download time. Instead of this approach, event delegation is an alternative way to go. Event delegation means that you instead of, as in Figure 2.3, using the onClick event on every `td` item you have the onClick on the `table` element instead.

```
<table>
<tr>
  <td id="item1" onClick="x('item1 ')">item1</td>
  <td id="item2" onClick="x('item2 ')">item2</td>
</tr>
<tr>
  <td id="item3" onClick="x('item3 ')">item3</td>
  <td id="item4" onClick="x('item4 ')">item4</td>
</tr>
</table>
```

Figure 2.3: Table with the onClick attribute on every table cell.

The loading of JavaScript is perhaps one of the biggest issues. Placing a script in the wrong place can slow down your site radically. Always try to place it as low on the page as possible. Doing so it will not block any other downloading in older browsers and will not execute before the page is rendered. Sometimes a script is used to create the page, which makes this impossible.

As well as with CSS, JavaScripts can be compressed by removing unnecessary code and thereby reduce the file size. Combining JavaScripts reduces the number of Http requests, which more often than not leads to a faster page download. (Souders, S. 2007; Souders, S. 2009; W3Schools n.d.c)

2.2.3.1 Ajax (Asynchronous JavaScript and XML)

Ajax uses techniques in JavaScript to make asynchronous http requests, which makes it possible to fetch information in the background by using a so called `XMLHttpRequestObject`. Because it is done in the background there is no need for the webpage to be reloaded every the time. This also makes it possible to reduce the size of the requests. (W3Schools n.d.d)

Ajax can be used to download other resources in order to speed up succeeding pages. In Figure 2.4 there is an example on how this code might look. This is called preemtive loading and places the downloaded resources in the cache for faster fetching on succeeding pages. This technique can also be used to delay

heavier scripts which are not important enough to load in the begining. (Souders, S. 2007; Souders, S. 2009)

```
window.onload = doOnLoad;

function doOnLoad() {
   setTimeout("downloadResources()", 1000);
}

function downloadResources() {
   downloadJS("scripts/myScript.js");
   downloadCSS("style/myStyleSheet.css");
}

function downloadJS(url) {
   var element = document.createElement("script");
   element.src = url;
   document.body.appendChild(element);
}

function downloadCSS(url) {
   var element = document.createElement("link");
   element.rel = "stylesheet";
   element.type = "text/css";
   element.href = url;
   document.body.appendChild(element);
}
```

Figure 2.4: JavaScript code that downloads other script and style sheet when page is loaded.

2.2.4 Images

Images is one of the largest resources sent back and forth in networks. Beacause of this, one of the most important aspect when optimizing the images on a web site is to shrink their size.

In order to choose the right image format, images are split into two categories: graphics and photos. To place the image in the right category the features of the image have to be considered. Here is a list of what signifies each type:

Graphics

- High contrast and sharp color-transitions.
- Icons, Logos, Graphs.
- Relatively small amount of colors.

Photos

- Smooth color transition and gradients.
- Often millions of colors.

Below in Table 2.2 is a summary of the most important features of the most used image formats used on the web today.

	GIF	JPEG	PNG8	PNG24	PNG32
Transparency	Binary	No	Binary	No	Alpha
Animation	Yes	No	No	No	No
Nonlossy	Yes	No	Yes	Yes	Yes
Interlacing	Yes	Yes	Yes	Yes	Yes
Number of colors	256	2^24	256	2^24	2^24

Table 2.2: Properties of different image formats.

Since images are large resources on the web, it is important to take these into consideration. When optimizing images there are two steps to go through. The first is lossy compression wich means simply lowering the quality of the image in the extent that it is not visible to the eye or at least high enough quality. The second step is a lossless compression, wich deletes the unnessecary metadata of the image and reduces the color palette of the image, in certain extent that it only contain the number of pallettes that is actually used. The tools used for lossless optimization is shown in Table 2.3. (Smith, P. 2013; Souders, S. 2009)

A common mistake is to scale down large images in the browser. When this is done a larger image is downloaded, to later be reduced to, for example, a thumbnail. An example of this is when a full picture of 1280 x 1162 pixels is scaled down to 100 x 90 pixels. When downloading the bigger picture, that will not be used, it takes more time than it would on a smaller version of the same image. Therefore, it is important to save the image in the size that will be used, even though it will be used in several different sizes. (Smith, P. 2013; Souders, S. 2009)

Tool	Description
JpegTran (Recommended)	- Removes the metadata. - Use Huffman coding to compress the Jpeg.
PNGCrush (Recommended)	- Removes all chunks except the one for alpha transperancy. - Reduce the number of colours in the palette. - Tries 100 different methods for optimizing.
PNGOUT	- Performs bit depth, color and palette reduction.
optiPNG	- An extension on PNGCruch
GIFsicle	- Removes the duplicate information about a pixel in an animation. Used when a pixel does not change between frames during an animation.

Table 2.3: Optimizing tools for images.

2.3 Tools

In order to optimize a web site, it has to be analysed. By measuring with different performance tools, the bottlnecks can be found. These sections describes the different analysing tools used throughout this thesis.

2.3.1 Webpagetest.org

Webpagetest.org is an open source web performance tool developed by Google, which is used for testing the loading time of a web site. The testing tool can both be used online and downloaded to run locally on a computer. There are multiple ways to test a targeted web site with this tool. For instance you can get a waterfall chart presenting both the first as well as the second view of a web page. Another great functionality that may come in handy is generating videos showing how a page is loaded. This can even be done in a comparison between multiple pages.

Webpagetest.org does also contain an analysing tool, which runs during the tests, and present this trough grades and also some data of whether different optimizing techniques are applied or not.

More about this tool can be learned at the webpage *webpagetest.org/about*.

2.3.2 YSlow

YSlow is a free web performance analysing tool developed by Yahoo!. It runs tests on a targeted web site and returns grades of how optimized the web site is

regarding different aspects. In addition to rating the web site, it also provides the user with feedback on how to better optimize the targeted web site. The grades are based on 23 rules developed too best show where the targeted web site can be optimized.

More about this tool can be learned at the webpage *developer.yahoo.com/yslow*.

2.3.3 PageSpeed

PageSpeed is Google's web performance analysing tool, it analyses different well-known aspects of how to properly optimize a web site. The PageSpeed presents the result in form of grades and also gives advices on what is not properly optimized. PageSpeed and YSlow are similar tools with the same main focus, but the minor rules differ.

More about this tool can be learned at the webpage *developers.google.com/speed/pagespeed*.

2.4 Waterfall chart

The waterfall chart is the most common chart for presenting load-testing data of a web site. As seen in Figure 2.5 the waterfall chart shows in each line an Http request/response. It also shows how long it takes for the Http request/response to go through each state.

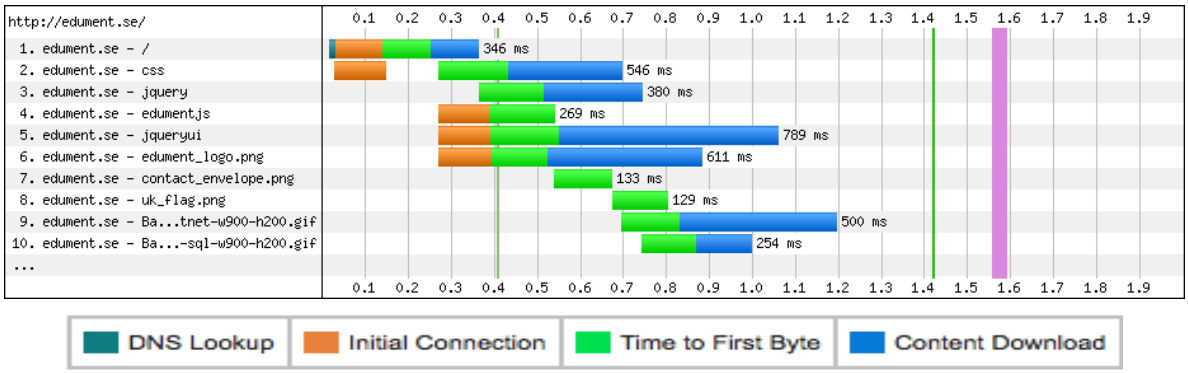


Figure 2.5: Part of Eduments web site. Colours by state.

The explanation of the different states is:

- DNS Lookup - The time it takes for the right server to be found.
- Initial Connection - The time it takes to set up the connection.
- Time To First Byte - The time between the Http requests sent to the first byte of the response returned.
- Start Render - First time the webpage is rendered.
- DOM Content Loaded - When the DOM has finished loading.

- Document Complete - Browser consider the page loaded.
- On Load - JavaScript event triggered when Document is Complete.

Seen in the Figure 2.6 what types resources are being loaded and also the name of each resource for easy access.

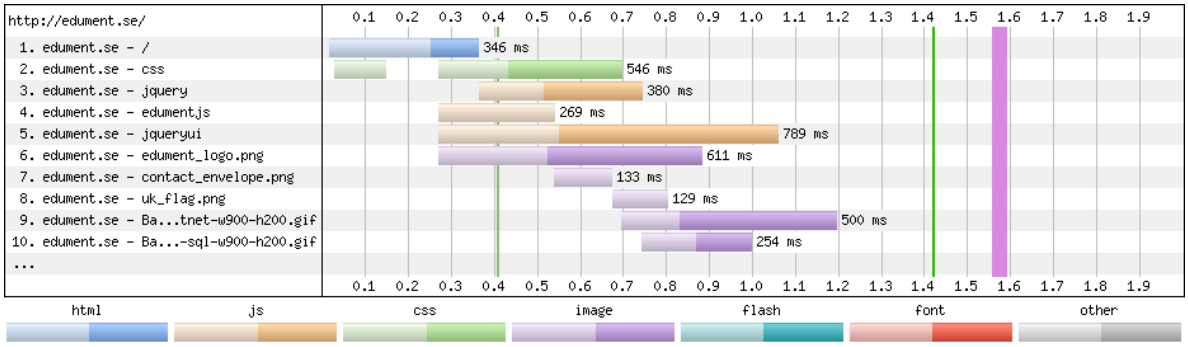


Figure 2.6: Part of Eduments web site. Colours by type.

3 Methodology

The purpose of the following chapter is to give an understanding of the methods used and how the problems and different situations were approached. It also describes how the information was gathered as well as how the tests were performed.

3.1 Information gathering

As the main assignment was to create course materials about web performance for Edument AB the first step was to gather information. Edument provided a number of books on the subject as well as notes and email conversations with links to articles about web performance. The challenge was to put together all the infomation found about the topic and evaluate it in order to deliver well elaborated course materials.

At the begining of the thesis the main source of information was books handed out by Edument AB. These books where from different publishing houses which contributed to a needed variation of setup. After a time of reading, a good overview of how the whole part of optimization was working had been obtained. This led to the testing phase.

The different testing tools used throughout the thesis was found both on the web and in the provided books. It was importat to test different tools to be able to find the tool that was best fitted for analysing web pages in terms of performance. The testing phase became an extended learning phase where many new techniques for optimizing was found throughout the tests. To be able to run the tests and being in control of what data was tested, a web site was set up as a testing environment.

3.2 Testing Environment

The reason for setting up a web site as a testing environment was that the tests had to be consistent. When owning the domain, all server preferences could be set to make test on for instance gzip and caching which may not have been possible otherwise. As the search of a webhost proceeded, Hostgator was found. Together with Hostgator's cheapest plan the domain lthexjobb.com was bought. This domain was the targeted web site to run tests on when using the testing tools webpagetest.org, YSlow and PageSpeed.

The web site was meant to be dynamic and was constantly modified by using different resources and different configuration settings on the server. The web site was modified to get as reliable test results as possible in order to accomplish as good examples as possible for the course materials.

3.3 Optimizing

The optimization phase proceeded throughout the entire thesis. As different aspects of web performance optimization was read upon, it was followed by testing and optimizing before heading towards the next topic. The testing and optimization phases for each aspect were running in multiple rounds before they were done.

3.3.1 Http

Optimizing Http was done by first minimizing the number of requests. The testing was done by combining resources and checking the reduction of the overall download time using webpagetest.org.

Another test that was done was using gzip to compress the payload to reduce the size of the resources being downloaded.

3.3.2 Caching

When testing how caching affects web performance the Http header was modified in the servers .htaccess file. The rules were set for the Http by implementing a Far Future Header. The header was set in a way so the expiration date of the resource was 1 month. The code below in Figure 3.1 was used to accomplish this.

```
<IfModule mod_expires.c>
ExpiresActive On
# Images
ExpiresByType image/gif "access plus 1 month"
ExpiresByType image/ico "access plus 1 month"
ExpiresByType image/png "access plus 1 month"
ExpiresByType image/jpg "access plus 1 month"
ExpiresByType image/jpeg "access plus 1 month"
# CSS
ExpiresByType text/css "access 1 month"
# Javascript
ExpiresByType application/javascript "access plus 1
hour"
</IfModule>
```

Figure 3.1: Code for activating caching of resources.

3.3.3 CDN

In order to test how a content delivery network could boost the performance of web sites, an account at CloudFlare was created. CloudFlare is one of the leading CDNs as of today and provides a stable service. The company have

both free and commercial plans where the commercial ones provides for instance better security and speed.

For the tests regarding CDN, a sub-domain was added to the test domain. The sub-domain was connected to CloudFlares servers and acted like a CDN.

3.3.4 CSS

Since there are many ways to optimize style sheets, the main focus were on the area giving the most performance gain. This was the downloading area.

To minimize the time taken for style sheets to be downloaded, a number of measures can be taken. First off was minimizing the number of Http requests because of CSS.

Tests regarding compression of style sheets were also done to show how the size of a resource affects the download time.

3.3.5 JavaScript

The performed tests regarding JavaScript included tests on different placements of the scripts in the HTML document, combined scripts and compressed scripts.

3.3.6 Images

When testing how images affects performance and how they could be optimized, a number of steps were made. The first step was reduction in quality to reduce the size of an image, also known as lossy compression. The second step included lossless compression, where image quality were not affected. The final step was to give the ability for larger images to be rendered progressively on web sites.

In addition to these three steps, tests on how combined images affected the performance were also done.

3.4 Testing

The testing phase was started right after a thorough overview on how the different elements of a web site were setup. When the server was up and running, the needed resources were uploaded in order to start testing.

The testing was divided into multiple testing phases, one testing phase for each topic of the thesis. The testing was done with three different testing and analysing tools. These tools were helpful only to a certain extent. The rest had to be analysed with help of the knowledge gathered during the researching phases.

The tools used for testing was:

- Webpagetest.org
- YSlow
- PageSpeed

The analysing part of the tools provided grades of how optimized the web site was. After this, the data given by the analysing tools was interpreted and considerations were made on how to improve the web site, to get the best results as possible.

During the testing it was very important to generate consistent data under a consistent condition. The test always ran in two parts, firstly the non-optimized example and then the optimized one. Both parts of testing were done with the same browser, bandwidth, CPU power and server.

The tests were executed five times and the median was selected for the results to be as correct as possible. The tests provided each a before and after waterfall chart. The testing results were then used in the course materials as real life examples.

3.5 Course materials

As mentioned earlier, the product to be delivered to Edument AB is course materials intended to be a part of a future course on web performance.

The course will consist of lectures, demonstrations and educational assignments spread over two days, seven hours a day. The purpose of this thesis is to deliver presentation materials in form of power points which will be used as a teaching base when lecturing.

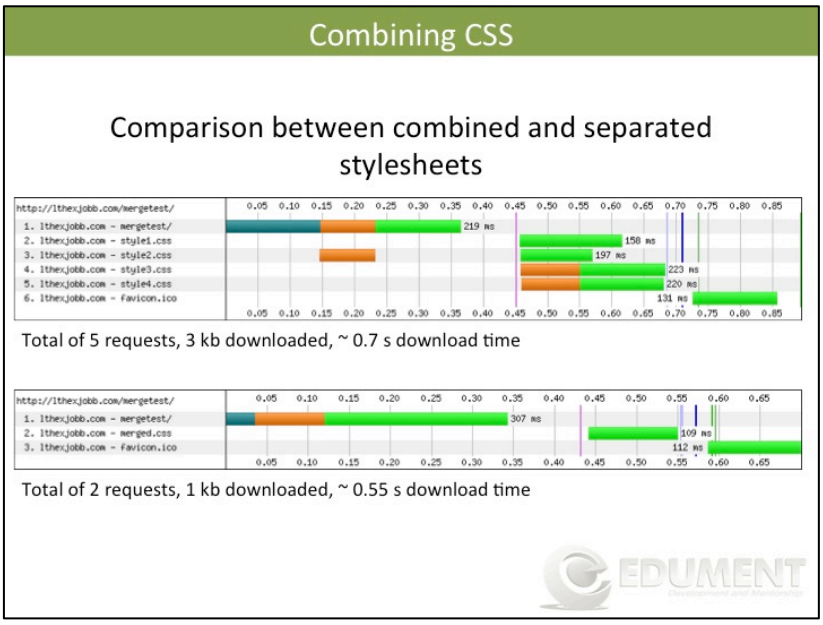


Figure 3.2: A slide from the course materials.

The presentations consists of six modules describing performance related problems and solutions in different situations. The subjects included are optimization of Http requests, HTML and CSS, JavaScript and images as well as how to analyse a waterfall chart and finding the bottlenecks.

The goal was to create enough, relevant and thrustworthy information as well as case examples that can be used when educating. In Figure 3.2 a slide of a presentation is shown to give a picture of what is delivered.

3.6 Sources of error

The main sources of errors in this thesis occurred during the testing phases.

- Wide range of web browsers:
  Each browser is handling web sites in different ways. This leads to some inconsistency when trying to optimize a cross browser web site.
- Number of times a test is run:
  The tests during this thesis were always done 5 times. Therefore the tests were presented as a possible outcome, which were also compared with the theoretical values.
- CDN:
  Only one free CDN was tested. Tests on CDN providers were not done which can also lead to a possible source of error.
- Only one type of server:
  During the whole thesis all the tests were running on an apache 2 server.
- Internet:
  Internet can be an unreliable medium, where performance can be affected by many different external sources.

3.7 Source Criticism

Most of the sources came from companies and authors that have worked with web performance for many years. Even though some books were half a decade old, much of the information gathered is valid even today. Almost everthing was tested to see if the methods and theory still was applicable.

The web sites from Google and Yahoo! are web sites related to the testing tools used and can therefore be trusted. The blogposts and articles from the lesser known sites were of course investigated thoroughly and tested so that the information was up to date and reliable.

Most of the information was gathered from Steve Souders books about web performance. Souders is a well known expert in web performance and is a Cheif of Performance at the company Fastly. The information retrieved from his books is highly reliable.

4 Analysis

This chapter will discuss and analyse the results found in chapter 5.

4.1 Http

While optimizing Http it was discovered that there were mainly two ways to manage it properly.

1. Reducing the number of Http requests.
2. Reduce the size of the Http payload.

The first and the most general way of optimizing with focus on the Http protocol is by reducing the number of requests. There was no single way or technique to do this, it was more like a line of argument throughout the whole thesis when optimizing the web.

In order to reduce the size of the Http payload, the resources were compressed. One way to do this was by configuring the server to enable compression by gzip. The server then compressed the payload with gzip before a response was sent and the client decompressed it when it was received. Gzipping was only possible when it is enabled on both the sending and receiving side.

Some versions of servers do not provide a support for gzip. In this case a possible solution is to use another compression method called deflate. There is no major difference between using gzip or deflate. Both does a great job compressing the requested resources.

To get a better clearance of how the gzip actually makes a difference the Figure 5.1 and Figure 5.2 was compared. In the comparison, a clear trend was showing that the effect of gzipping was only noticeable on bigger resources. This is due to the fact that gzipping comes with an increase of CPU usage. The CPU usage only outweighed the time earned by compressing when the resources sizes were less than 1 or 2kb.

All the resources, except for images, were compressed using gzip. A rule of thumb is to never gzip images. Images should instead be compressed with its own compression techniques before stored on the hosting server.

4.2 Minifying Http Requests

To minify the number of http requests the number of resources being downloaded has to be reduced. In order to reduce the different resources they were combined.

In Figure 5.3 a waterfall chart of the original page, with 13 requests, is shown. In this chart none of the resources are combined. In Figure 5.4 through 5.6 the different resources are combined by type. In some cases there is a negligable,

if any, save. The explanation for this is that there is only a relatively small amount of requests. Imagine a bigger page with a lot of requests. A page like that will have more to gain on reducing the number of requests because of the number of parallel downloads in the browser.

Even though a bigger web site has more to gain on minimizing the number of requests, Figure 5.7 shows that not every resource did have a shorter download time. The number of requests has however in this example been reduced to seven.

4.3 Caching

Figure 5.11 and Figure 5.10 is showing the second page view when the same webpage is cached respectivly not cached. In the waterfall charts there are 304 responses shown by the Http protocol. 304 response means that a check of the expiration date has been made and that it have not been modified and can therefore be used from the cache. This means that the resources does not have to be sent from the server. The request is however still made.

```
<IfModule mod_expires.c>

# Enable expirations
ExpiresActive On

# Images
ExpiresByType image/gif "access plus 1 month"
ExpiresByType image/ico "access plus 1 month"
ExpiresByType image/png "access plus 1 month"
ExpiresByType image/jpg "access plus 1 month"
ExpiresByType image/jpeg "access plus 1 month"

# CSS
ExpiresByType text/css "access 1 month"

# Javascript
ExpiresByType application/javascript "access
plus 1 hour"

</IfModule>
```

Figure 4.1: Code for activating caching.

When enabling caching of static content for a web site, the Http header has to be modified in order for the client, which is receiving the content, to know whether to cache it or not.

Figure 4.1 shows how the far future headers were implemented for all the resources except for the HTML document.

The reason for not caching resources for under 1 hour is that it is not likely that people revisits a home page in the time frame of one hour. The far future header is often the solution which is used to cache a resource for as long as possible and is often recommended for static content. If the content is dynamic it can lead to unwanted effects.

Caching should only be used on static resources wich are likely not to be changed in the far future. The pitfall of caching is when dynamic resources are cached and will therefore not be updated when needed. This can lead to outdated information being shown on the webpage.

A possible solution for this is to simply set a version number in the filename of the resource, which then is incremented along every update. The change in the resources name will lead to a new Http request and the information will always be up to date.

4.4 Minifying Code

By minifying style sheets and scripts the size is reduced which leads to a shorter download time. Figure 5.12 shows a chart over a webpage without any kind of code compression. The total download time is approximately 0.95 seconds. The time spent on downloading CSS is 156 milliseconds and downloading JavaScript 594 milliseconds. The size of the style sheet is 75kb and for the script 242kb. These files comes from a big news site with a lot of styling and functionality.

By minifying, or compressing, the code in the style sheet the size is reduced to 66kb. A reduction of 12%. The downloading time was reduced to 153 milliseconds, which was a negligable saving.

Compressing the JavaScript gave a much bigger save in both size and time. 242kb was compressed to 82kb and 594 milliseconds was reduced to 317. This means a reduction of 66% respectively 47%.

Clearly, the bigger the file is the more the savings there are by compressing. JavaScript offers a bigger save than CSS, this is because of the comments are more likely to add up in a script than in a style sheet. Long variable names can also be shortened in scripts which is impossible to automate in style sheets.

Minifying code is a great and simple way to reduce the download time on resources. Although this is good for performance, the maintainabilty is close to gone on minfied code. This problem is easily avoided by working in full size CSS and JavaScript and compressing them every time they are uploaded to the server. This gives the best of both worlds. Creating a script that takes care of this makes it possible to automate the minification.

4.5 CDN

To reduce the time spent on finding the correct server to download content from, the geographical distance between client and server can be reduced. This also reduce the time spent in cables just transporting data. One way to do this is to investigate the users of the site and place the server as close as possible to as many users as possible. This can be hard if the site is an international web site with users from the whole world. This is where a CDN come in handy.

In Figure 5.16 a chart showing the downloading of a page to a browser in Dulles, USA. Because the server is located in the US the download is relatively fast. US users would probably not complain about a slow web site. Consider loading the same site from Sydney, Australia while the server is still located in the US. The loading in this case is presented in Figure 5.17. In Sydney the loading takes almost three times as long. Here is obviously room for improvement.

With all static content such as style sheets, scripts and images placed at a host, in this case CloudFlare, the distance between client and server is reduced dramatically. Loading the same resources as before but with static content on a CDN, to a client in Dulles, does not seem to reduce the overall time more than by a few milliseconds. The big win is in Sydney where the total download time is reduced from 2.2 seconds to a little bit over 1.3. In terms of web performance this is considered to be a huge save.

While a CDN may seem to be a good choice for boosting the performance of a web site it has some drawbacks. One drawback is that you do not have 100% control over the servers that hosts the content. This includes everything from uptime to the security.

4.6 CSS

The style sheets were not optimized more than by combining multiple file into one and by compressing the code. This is already mentioned in section 4.2 and 4.4 and is therefore not described or analysed in this section.

4.7 JavaScript

Optimizing the affect that JavaScript has on web performance, the focus was on the biggest issue, the loading time. Since scripts put on top of the page can block other resources when it is being downloaded or executed, multiple loading tests were made. In these tests, scripts were put in different places in the document to compare the cases to one another. The tests were then made by using webpagetest.org to get a waterfall chart over the requests when first entering the pages. These charts are shown in Figure 5.20 trough 5.23. As shown in the tests, JavaScript is blocking the ability for the browser to

download multiple resources in parallel. In the first test the script blocked all images, which did not get downloaded until the script was fully loaded, parsed and run. In Figure 5.21 the script is placed between the style sheets and the images. It is still blocked all the images. As seen in the two subsequent Figures the script only blocks what comes after. In Figure 5.23 it did not block any resources but the favicon.

Putting JavaScript at the bottom is clearly the best choice. This may however not always be possible. Imagine a script that helps creating the web page, when put at the bottom the site could be rendered later than wished. This is a drawback when creating elements needen upon load. Combining scripts will minimize the number of requests. A split between functions that will render the page and functions that are needed later will however enable the important scripts to be loaded early while the rest is put at the bottom to increase the parallellism in the browser.

By using Ajax to download heavy scripts after the page is loaded the page rendered faster. One drawback of this is that the script cannot run until it is downloaded which makes it hard to put scripts that alter information on hold. In Figure 5.24 this technique is shown in a waterfall chart. Another way to use this technique is to download content needed later, for instance another script or style sheet needed on succeeding pages. By doing this, the downloaded content is already in the cache which speeds up the download for the next page.

4.8 Images

An important thing to think of when using lossy optimization is that the point of optimizing images is to give a better user-experience. That is why it is important to lower the quality as much as possible, without compromising with the visuals.

When using lossy optimization on images it is good to take into consideration that Jpegs is a lossy format and should therefore be stored as a PNG24/32 before publishing. The reason that the PNG is a better format for the web than GIF is that they serve the same purpose with the only difference that GIF can be used for animation. If GIF is not used as an animation, it is better to convert it into a PNG format due to the fact that it is a better compression format, which finally leads to a smaller image size.

Lossless image optimization in turn gives a smaller size reduction, but is complementary for the lossy part. Nevertheless it is a very important part for giving an image its finishing touch and finally having a fully optimized image. When photos are taken, information about the photos is stored alongside with it. This extra information is stored as metadata. An example of metadata is

where and when the image was taken. The metadata is therefore redundant and removed by lossless optimizing tools.

As Jpegs is used for photo types of images they can often get relatively large. When a Jpeg becomes larger than 10kb it is often a good practice to take progressive Jpegs into consideration. By saving it as progressive the browser renders the picture from poor to best quality instead of from top to bottom with full quality as they normally are rendered. This is shown in Figure 5.26.

In terms of logos and icons, when the two steps for optimization has been done a final check for optimization is to use sprites if applicable. When two logos has been optimized as PNGs, they should be merged together as a sprite as shown in Figure 5.28. This can reduces the overall size of the images due to the fact that only one colour palette is used. This is why it is a good practice to merge images with similar colours into one sprite if multiple are used. Another way to optimize a sprite is by aligning the logos horizontally. The compression for both GIF and PNG is working horizontally, and therefore a higher size reduction can be achieved. Finally a sprite should go through the optimization once again.

This seems like a lot of work on web sites with a lot of images. The tools listed in section 2.2.4 are partly automated. To fully automate the image compression, a script that runs the tool of choice can be created. However, when automating the optimization it is harder to get the quality of choice. Sometimes, the time spent on optimizing images is not worth the gain in performance.

Below, in Figure 4.2, there is a proposal of a decision tree when optimizing images. The tree is based on the decisions made in this thesis.

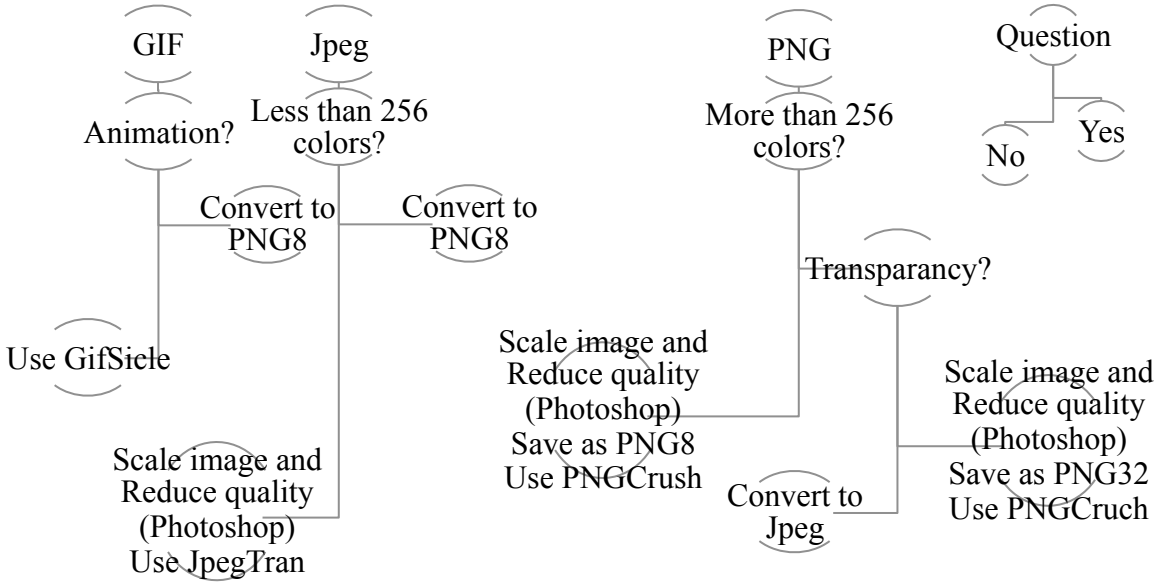


Figure 4.2: Decision tree

4.9 Tools

As discovered later on in this thesis Yslow and Pagespeed were generating great analysing results. The drawback was that they did not generate such visual results that were required in the course materials. Therefore Yslow and Pagespeed were only used in the beginning of the testing phases. They were used to get a better understanding of the fundamental parts of optimization and also analysing where the possible bottleneck might be. The main part of the testing phase was spent on the webpagetest.org's testing tool. It was used for testing the non-optimized target, finding the bottleneck, testing the optimized target and finally generating proper testing results that showed differences. These testing results where then used in the course materials as real life examples.

5 Results

Thanks to the testing tools used, results were generated across the entire project. These results are presented in this chapter. Alongside with the waterfall charts and calculations, there is also explanation for each test case.

5.1 Http

When using gzip compression on the Http requests, a reduction of size was observed. In the tests, gzip was enabled to compress every resource except for the images. One of the test is shown below, where Figure 5.1 demonstrates the test with gzip disabled and Figure 5.2 with gzip enabled.

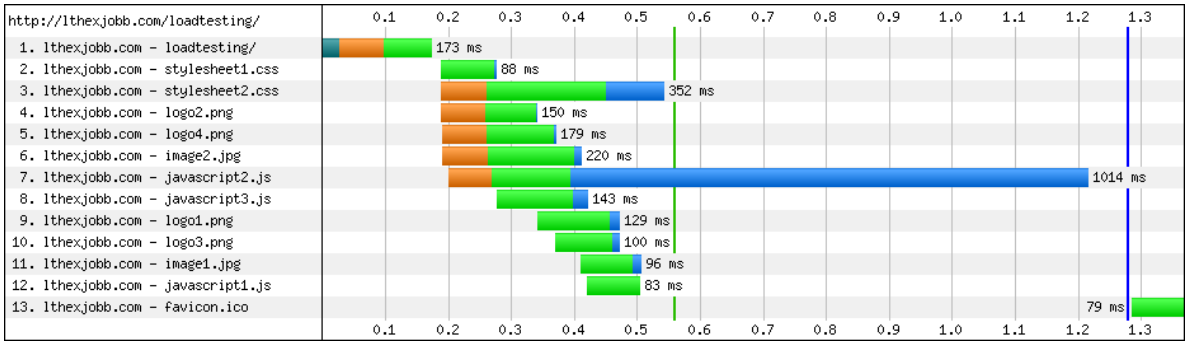


Figure 5.1: First view with gzip disabled.

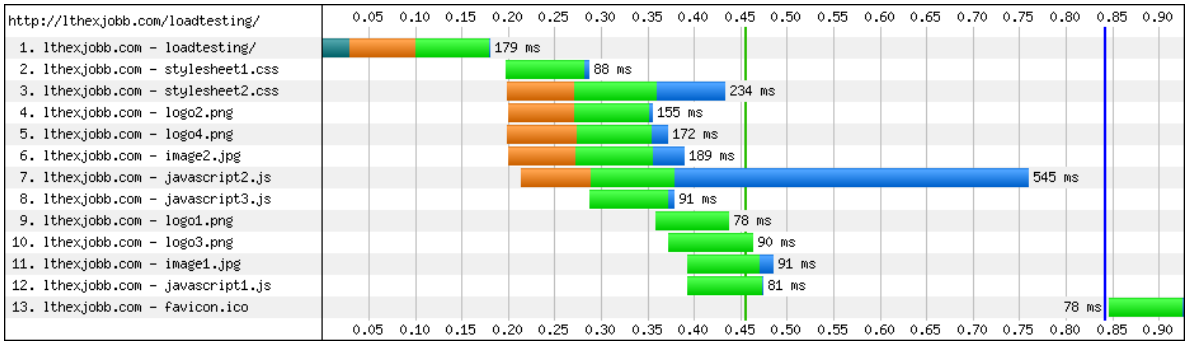


Figure 5.2: First view with gzip enabled.

Every request, that was compressed by gzip, had in the second figure a smaller file size. A trend is seen in this graph where the smaller resources have not resulted in a reduced loading time. The affect is the opposite on the bigger resource, for example javascript2.js. The logos and images have not been changed because gzip was not used on these resources, for this reason they are not taken into consideration.

5.2 Minimizing Http Requests

To minimize the number of requests sent from the browser there is one major action to take. That is combining files.

Combining style sheets and scripts and is basically adding one file to another. Combining images is done by adding them to a sprite. Below, combinings of the different resources is presented.

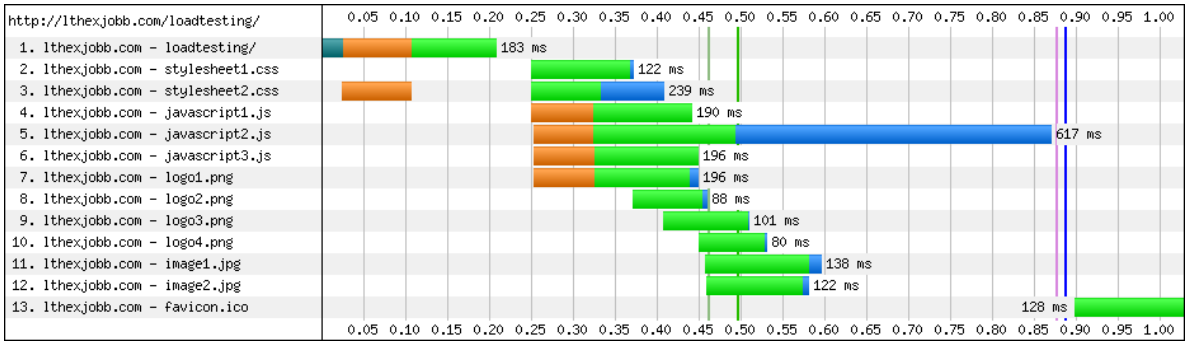


Figure 5.3: No combining done.

As seen above, when no combining is done the stylesheets takes 122 milliseconds respectively 239 millisecond to download. The scripts takes 190, 617 and 196 milliseconds. The second script, javascript2.js, is much bigger than the other two. The logos or icons are the same size and takes about 80 to 100 milliseconds to downlad, with exception of the first, logo1.png, that has an inital connection on approximately 15 milliseconds.

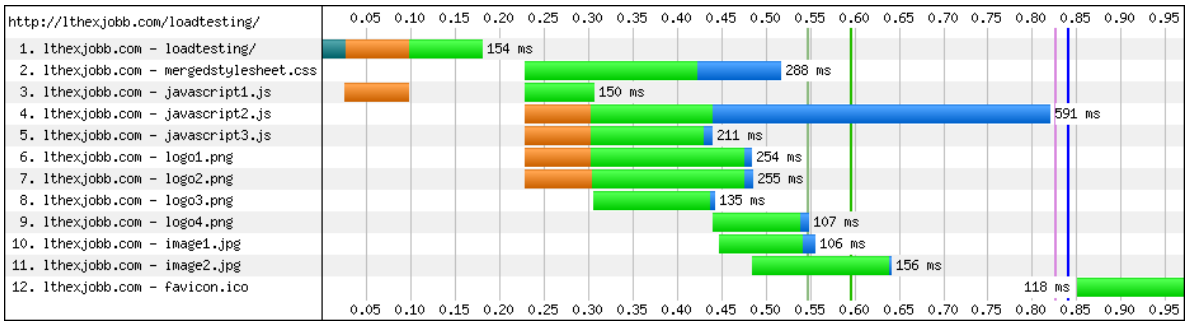


Figure 5.4: Style sheets combined.

Figure 5.4 shows how combining style sheets affect the download time. It now takes 288 milliseconds instead of 239, which was the longest before. The download time is in fact longer, but now the second icon (logo2.png) also can be downloaded in parallell.

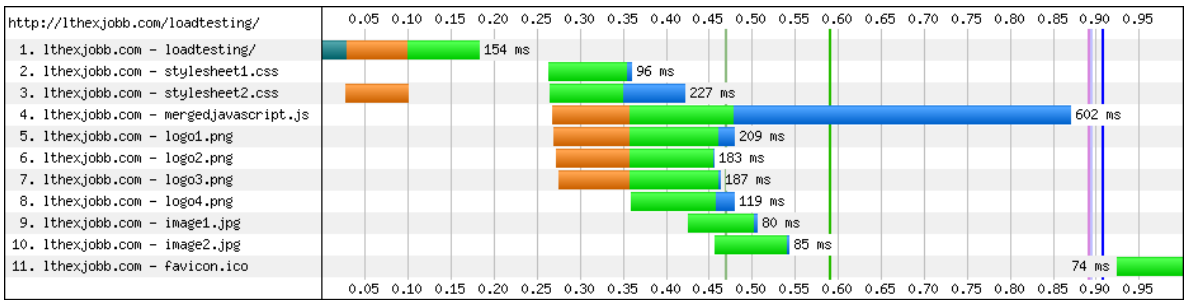


Figure 5.5: JavaScript combined.

Above in Figure 5.5 the second and third script is combined. The first script was so small, under 1kb, and was important to be executed first and was therefore inlined into the HTML document.

The total time taken to download the scripts was 602 milliseconds. Combining the scripts may not be the ultimate technique when the scripts are this big. The biggest gain here is however the reduction of the number of requests.

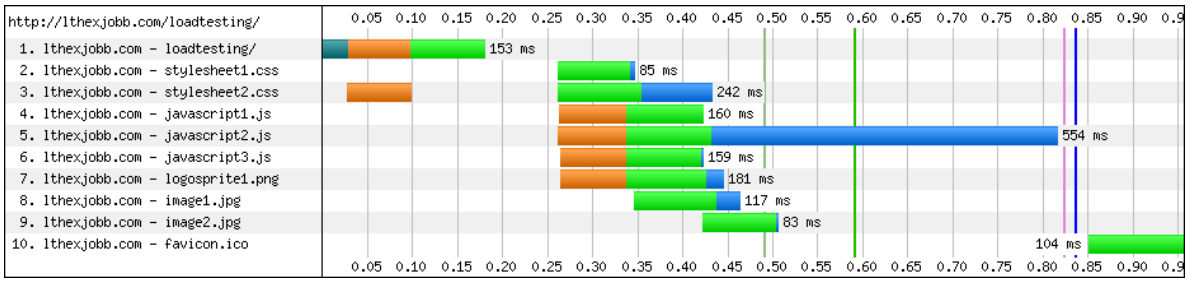


Figure 5.6: Logos combined.

By using CSS Sprites the logos and icons were combined. This is shown in Figure 5.6 above. Now all the logos are downloaded in parallell with the scripts and style sheets. It does not however affect the total download time more than by a few milliseconds.

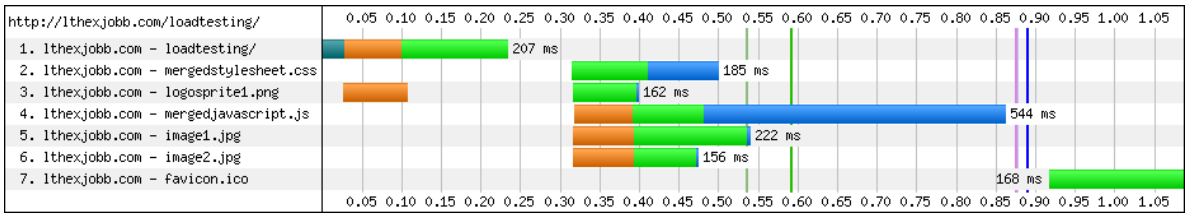


Figure 5.7: All combined.

In Figure 5.7 below all the combining techniques are used. Compared to Figure 5.3 were no combining was done the biggest different is the number of requests. From 13 down to 7. The total download time is however the same. This is because of the big script, javascript2.js. This shows that combining is not always the answer to better web performance. By instead splitting the JavaScripts and use the full amount of parallell downloads the web browsers can offer the total download time will be reduced. This differs between different web browsers and should therefore always be tested to reach the ultimate optimization level of a specific site.

5.3 Caching

The figures below shows the testresults generated during the testphase of caching.

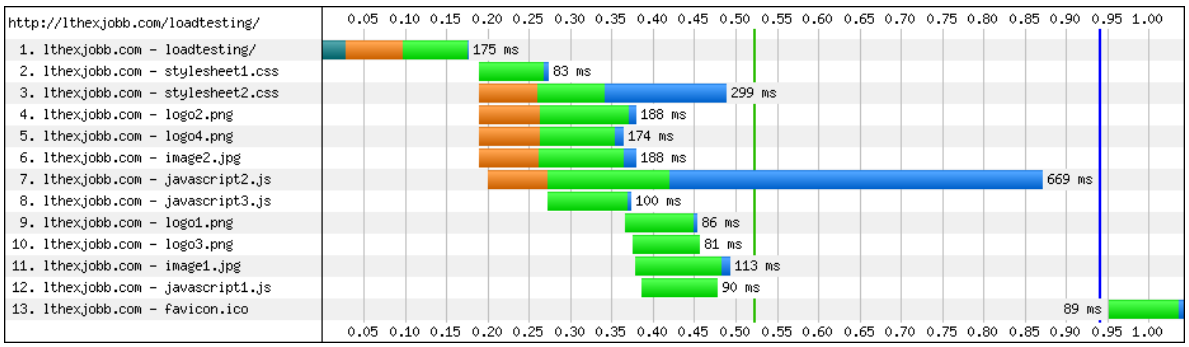


Figure 5.8: First view with caching disabled.

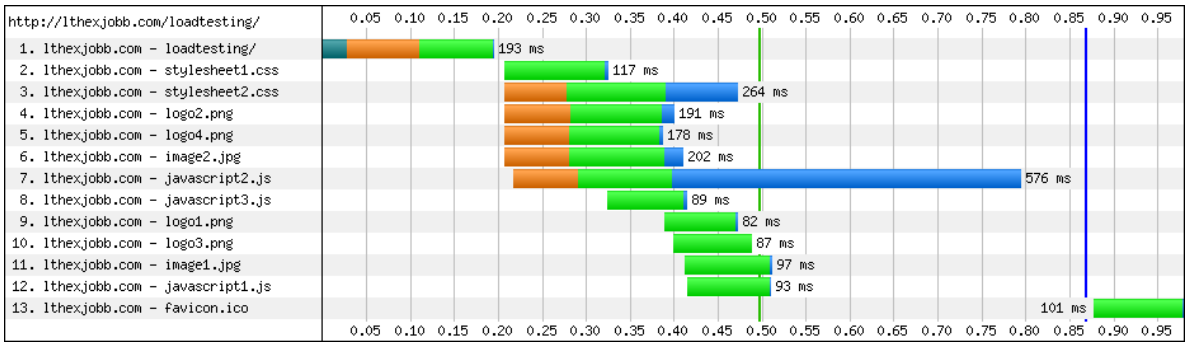


Figure 5.9: First view with caching enabled.

The images above are representing the Http request in the first view of the webpage. The first waterfall chart, Figure 5.8, is showing when caching is enabled and the second, Figure 5.9, when it is disabled. The second or repeat view is different. This is shown in Figure 5.10 repectively Figure 5.11.

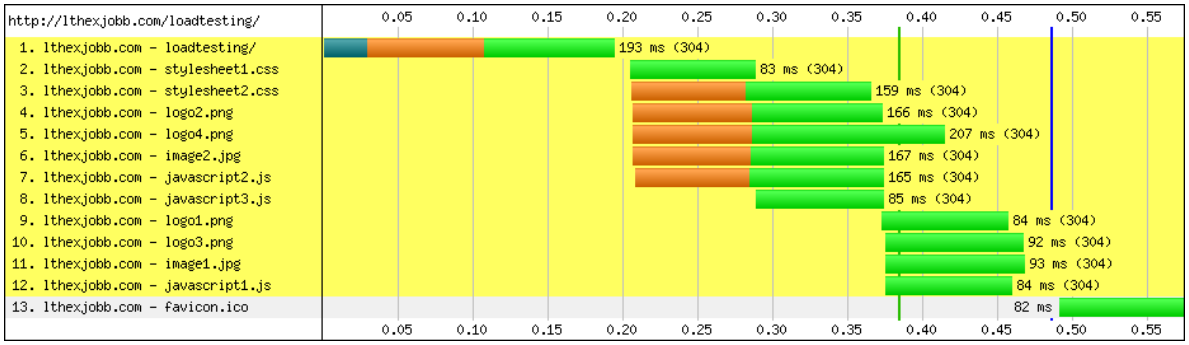


Figure 5.10: Second view with caching disabled.

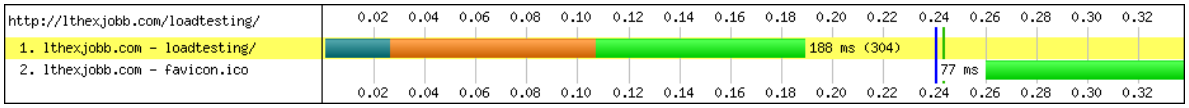


Figure 5.11: Second view with caching enabled.

The yellow background in Figure 5.10 and Figure 5.11 is indicating a 304 redirect. Redirect means that a conditional GET have been made and returned the response 304. 304 means that no modification has been made to the resource since the resource was requested last.

5.4 Minifying Code

Below, in Figure 5.12, is a waterfall chart over a web page download with no code minified. The focus here is on the uncompressed CSS and JavaScript. The style sheet is 75kb and the script is 242kb big.

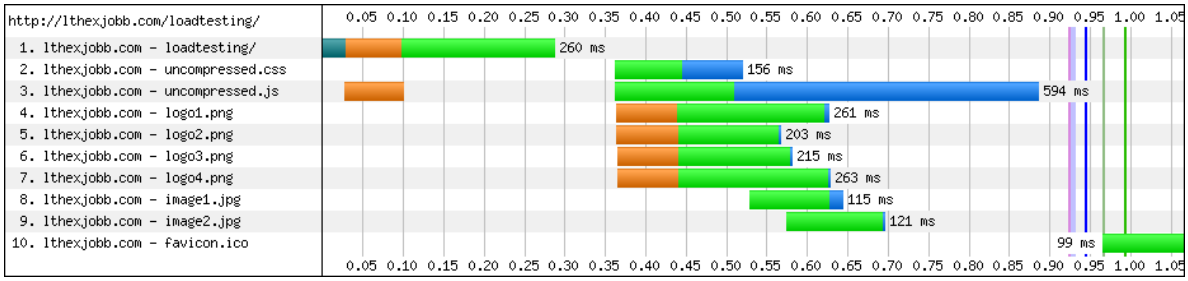


Figure 5.12: Nothing minified. CSS - 75kb, JS - 242kb.

The following two charts presents the results when CSS and JavaScript is compressed.

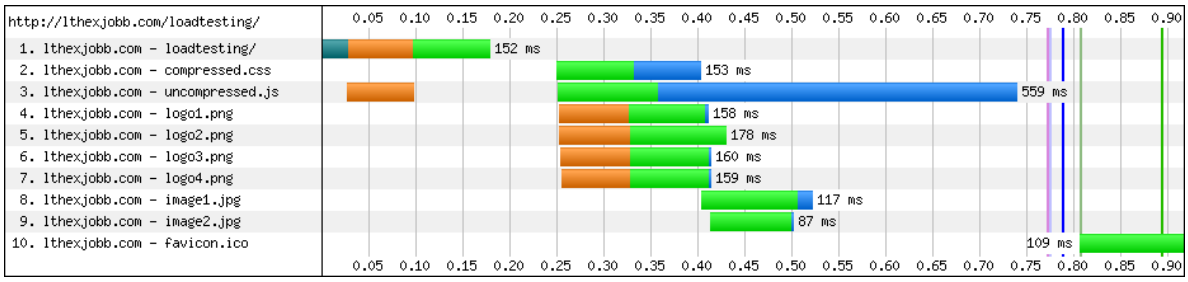


Figure 5.13: Compressed CSS - 66kb.

In Figure 5.13 the style sheet is compress to 66kb, which is a reduction of 9kb. It does not affect the download time of the style sheet significally.

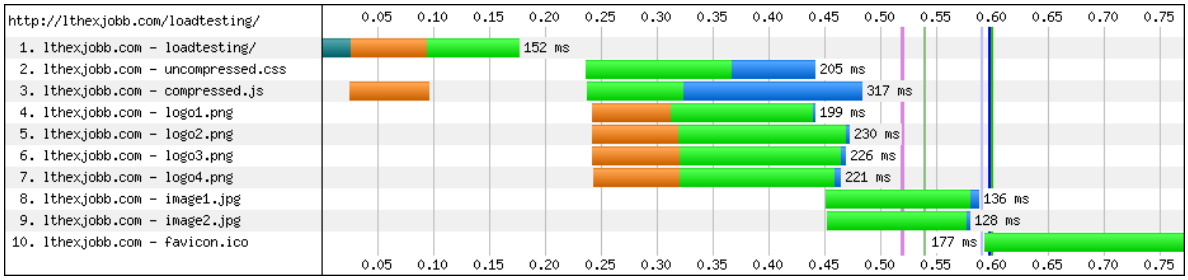


Figure 5.14: Compressed JavaScript - 82kb.

Above in Figure 5.14 only the script is compressed. It is reduced to 82kb, which also gives a reduction of the download time.

Figure 5.15: Both CSS and JavaScript compressed.

In this final chart, Figure 5.15, both the CSS and JavaScript are compressed. Compared to Figure 5.12 the total download time is reduced by approximately 40%.

5.5 CDN

As mentioned earlier, in section 2.1.4, a content delivery network serves to locate web contents as close to as many users as possible. Since the web host used in this thesis has their servers in the US Figure 5.16 shows the time taken to download a web page directly from their server to a client in Dulles, US.

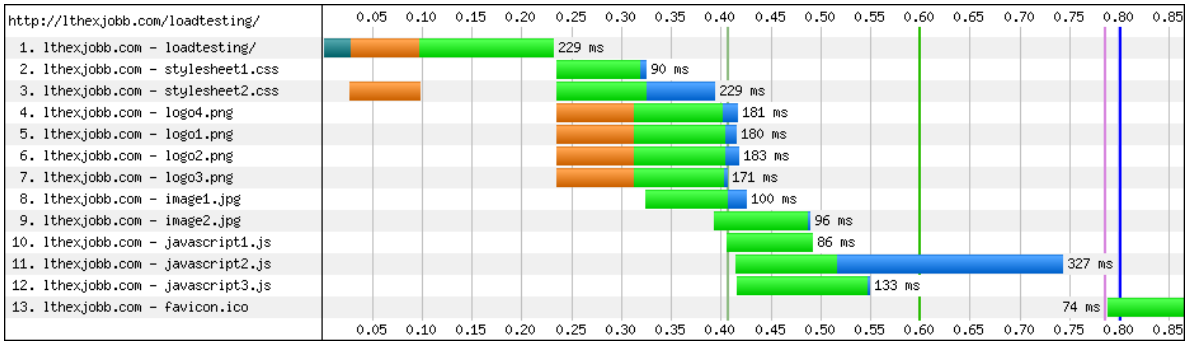


Figure 5.16: First view without CDN. Dulles, USA

The content in Figure 5.16 is downloaded in approximately 0.8 seconds. Accessing the same server to download the same content from Australia logically would take longer due to that the distance is far greater. The result of the test is shown below, in Figure 5.17.

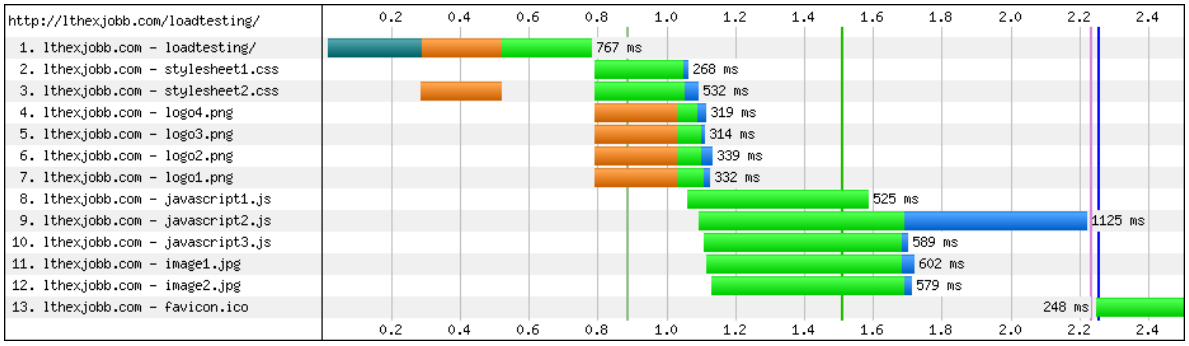


Figure 5.17: First view without CDN. Sydney, Australia.

Loading the page in Australia takes more than 2.2 seconds which is much longer than in the US. In this case a CDN may come in handy.

Figure 5.18 presents the waterfall chart of downloading the same web page as above. This time the static content, in this case images, style sheets and scripts, is located on a CDN. No big overall difference from Figure 5.16 without CDN.

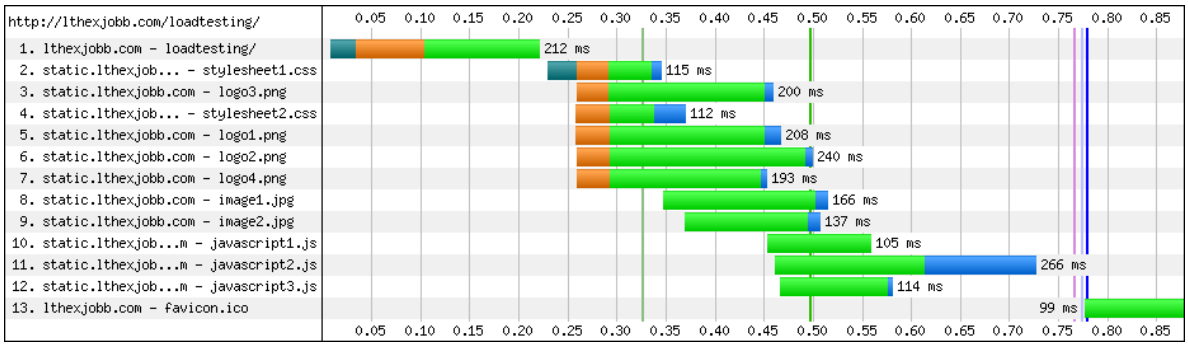


Figure 5.18: First view with CDN. Dulles, USA.

In Figure 5.19 the client is back in Australia and this time has the statics on a CDN. The content is downloaded in a bit over 1.3 seconds which is a huge save compared to without CDN.

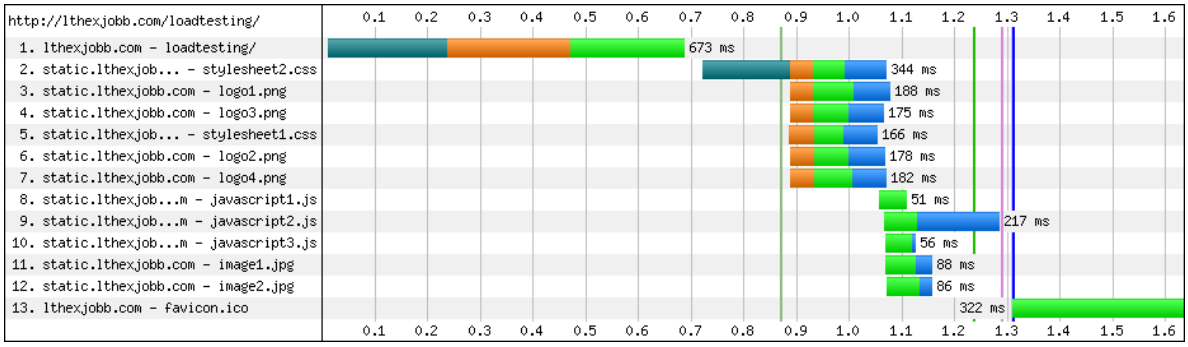


Figure 5.19: First view with CDN. Sydney, Australia.

5.6 CSS

One of the optimizations done on style sheets was the combining of files. Since this is described in section 5.2 this will not presented in this section.

Minification of style sheets is also presented earlier in section 5.4 and will not be included here.

5.7 JavaScript

As well with CSS, combining of JavaScript is presented in section 5.2 and minification is shown in section 5.4.

Someting that is not presented earlier is the result of the placement of JavaScript. The following images present the waterfall charts when the importing tags is placed in different positions in the HTML document.

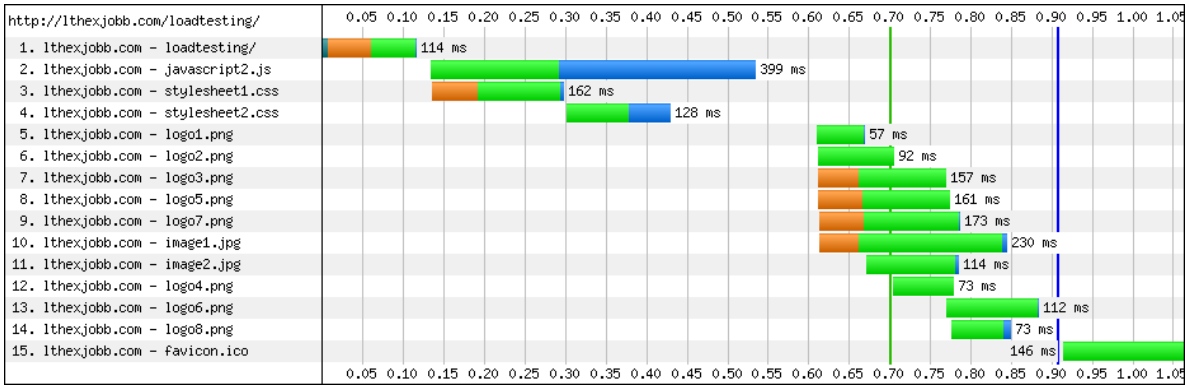


Figure 5.20: JavaScript put in the head, before style sheets.

Figure 5.20 above shows how a script put above the style sheets, in the head of the HTML, blocks everything after it. The first image, logo1.png, is not downloaded until 0.60 seconds.

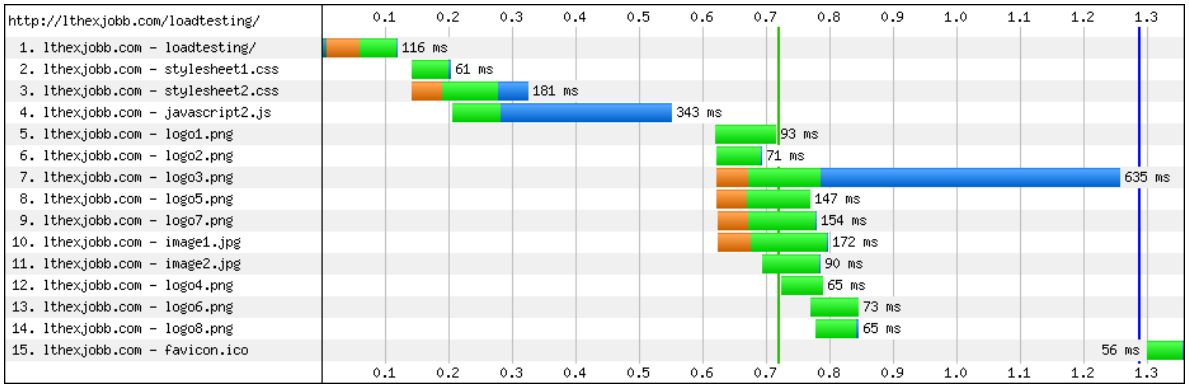


Figure 5.21: JavaScript put in the head, after style sheets.

Above, in Figure 5.21, the only difference from Figure 5.20 is when the style sheets is downloaded. The third image, logo3.png, seems to take longer to download. This shows how different the internet can be from time to time.

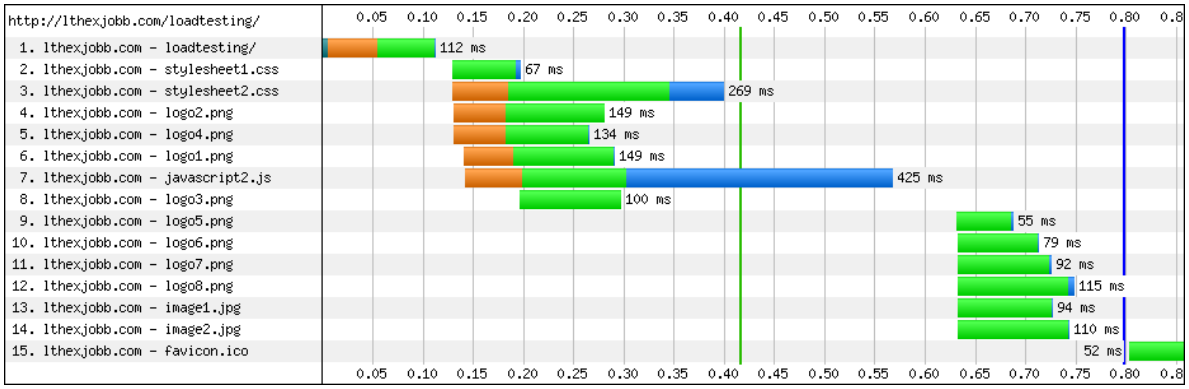


Figure 5.22: JavaScript put in the middle.

When putting the script in the middle, like in Figure 5.22, everything after it is blocked and will not be downloaded until the script is fully downloaded, parsed and executed.

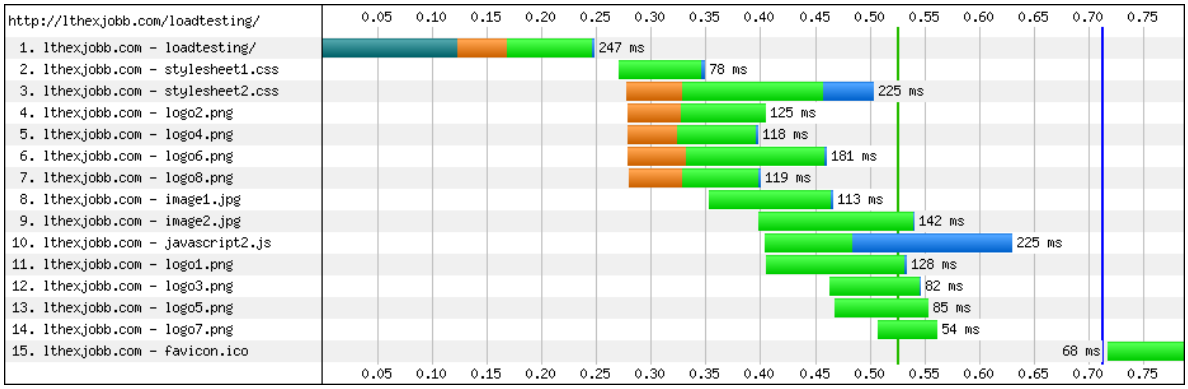


Figure 5.23: JavaScript put att bottom.

In Figure 5.23 the script is put at the bottom of the HTML document and does not block any other resource, except for the favicon. This saved 20 milliseconds as opposed to when the script was in the top of the document.

There was also done a test on post-onLoad download. Presented below is a web page download with one style sheet and one script downloaded after everything else with a delay on 1 second.

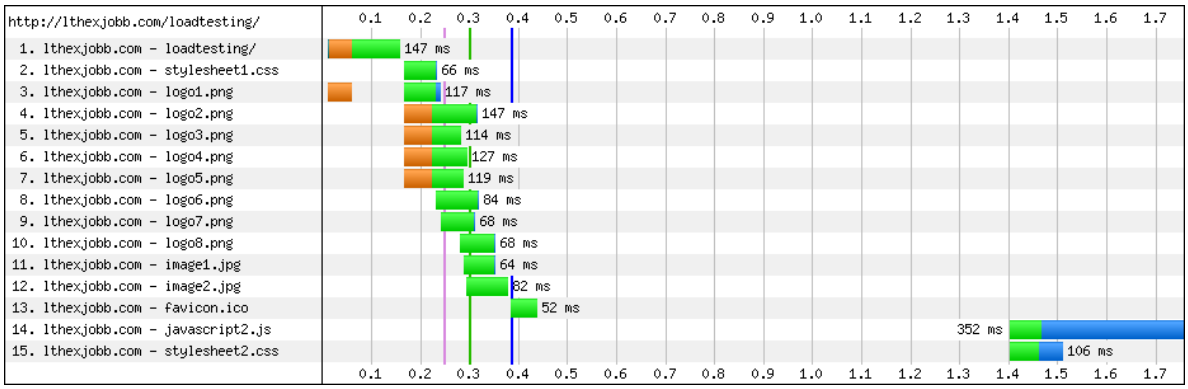


Figure 5.24: Post-onLoad download of style sheet and script.

5.8 Images

Due to the fact that there are different image formats used on the web today, it generated quite a lot tests and results. In this part a few test cases of each format is presented.

Jpeg is the format recommended for photos. The images below are showing a comparison between two identical photos. The only difference is that the right image in Figure 2.25 have been optimized by lowering the quality of the image. The image have also run through the lossless image optimization tool called JpegTran.

Figure 5.25: Comparison between the original (319kb) and optimized (144kb) image.

The full optimization of the Figure 5.25 lead to a decreased size with 175kb. The right image in Figure 5.25 lost 54.9 % of its size by the optimization with an almost non-recognizable quality reduction.

Due to the images having a tendecy to be large resources on a web site, it is important to handle them carefully. A recomendation is to turn a big Jpeg into a progressive Jpeg so it seems that it is rendered faster. The Figure 5.26 below shows the result when progressive Jpeg is compared to a normal Jpeg.

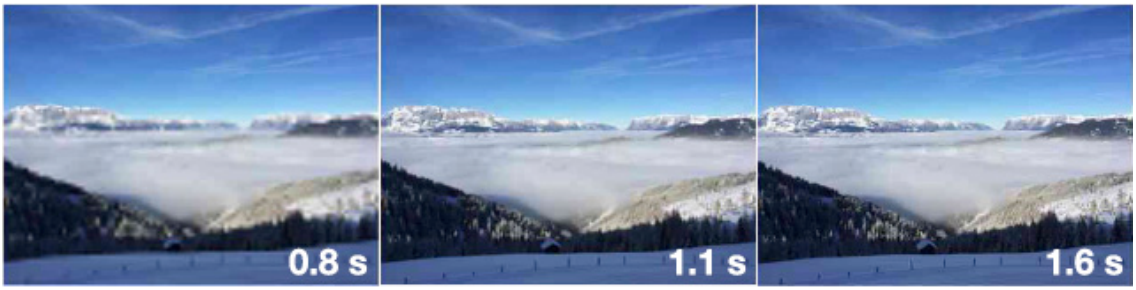


Figure 5.26: Rendering of a progressive Jpeg.

The Figure 5.26 shows three points of when the progressive Jpeg renders, alongside with the time it took. As seen in the images the quality increases by time. When not using progressive images, the image is rendered in full quality from top to bottom. This can be slow on big images or slower internet connections.

Because it is a good praxis to convert GIF images into PNG, optimization of GIF images will not be presented in this thesis.

The PNG has the sub types PNG8, PNG24 and PNG32. What is presented here is the optimization of PNG8. Showing the two images in Figure 5.27 with a comparison between an optimized and a non-optimized graphical image.

Figure 5.27: Comparison between original (12.6kb) and optimized (11.2).

The result of the lossy optimization is a reduction of 1.4kb which results in a 11.1% reduction.

When optimizing the images with a lossless technique there will be no difference in quality. That is why only the test data is presented here. When both PNGCrush and PNGOUT were used on the image the reduction in size was 1.2kb, which is a 9.8% reduction.

To minimize the number of requests containing images, CSS Sprites were used. The result of this technique was in addition to the reduction of the number of requests also a smaller sprite due to the reduced number of color palettes. An example of how a sprite may look like is shown in Figure 5.28.

Figure 5.28: Four images combined into one Sprite.

5.9 Course Materials

The results above were the main parts of the course materials which were to be delivered to Eduement. The tests gave an understanding of what gives the most gains performance wise. All of the aspects above are more or less represented in the presentation materials to give the broad base that is needed to understand web performance.

The topics included are:

- Analysing of waterfall charts
  - How to generate a waterfall chart
  - Finding the bottleneck
  - Tools
- Http requests
  - How can the number of requests be minimized?
  - Combining/merging of style sheets, scripts and images
  - Post onLoad downloading
  - Compression by Gzip

- HTML & CSS
    - How to optimize HTML?
    - CSS selectors
    - Minifying/compressing style sheets
    - Combining style sheets
- JavaScript
    - How can JavaScript be optimized?
    - Interaction with the DOM
    - Repaint/rerender
    - Event delegation
    - Code tips
    - Script loading
    - Position in the HTML
    - Merging/combining, splitting and inlining
    - Minifying/compressing scripts
    - Ajax
    - Preemptive loading
- Images
    - How can images be optimized for the web?
    - The different image formats
    - Lossy optimization/compression
    - Lossless optimization/compression
    - Tools
    - Interlace/progressive
    - CSS Sprites
    - Less used formats

Below are slides from some of the modules.

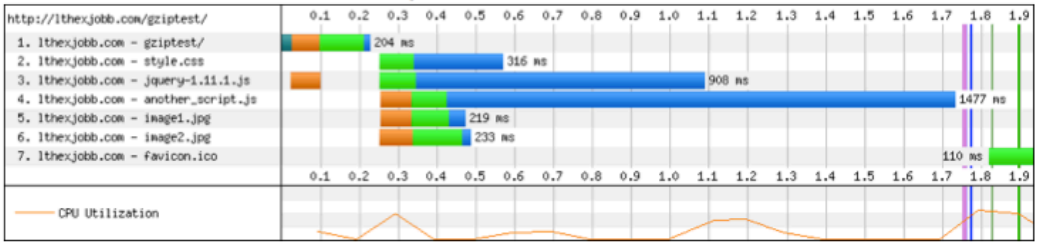
HTTP

HTTP is the protocol used when the browser is communicating with the server

Whenever your browser asks for a file a HTTP request is done and the web server sends back a response with "instructions" to download the file

Example

Compression disabled



Gzip enabled

65% saved in size, roughly 1 second in time

CSS Sprites

Four images -> four HTTP-Requests

Total size: 42 kb

One master image -> one HTTP Request

Total size: 19 kb

Always do a lossless optimization on a sprite to reduce the size even more

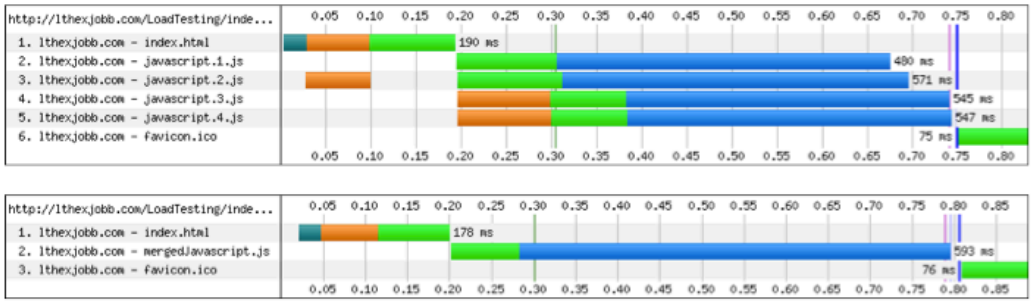
Combining images with similar colors in on sprite often reduces the file size

Waterfall Chart

How do we analyze the waterfall chart when measuring performance?

Click to add title

In a smaller site, merging may not be the answer



Always test your site with different browsers!

Minifying CSS

```css
@charset "utf-8";

/*Sets margin to zero and styles the body*/
html, body {
margin: 0px;
padding: 0px;
border: 0px;
color: #000;
background: #fff;
}

/*Sets the standard font-family for some html types*/
html, body, p, th, td, li, dd, dt {
font: 1em Arial, Helvetica, sans-serif;
}

/*Sets headline standard font to arial*/
h1, h2, h3, h4, h5, h6 {
font-family: Arial, Helvetica, sans-serif;
}

/*Sets standard font-size for headlines*/
h1 { font-size: 2em; }
h2 { font-size: 1.5em; }
h3 { font-size: 1.2em ; }
h4 { font-size: 1.0em; }
h5 { font-size: 0.9em; }
h6 { font-size: 0.8em; }

/*Styles the links on the page*/
a:link { color: #00f; }
a:visited { color: #009; }
a:hover { color: #06f; }
a:active { color: #0cf; }
```

```
@charset "utf-8";body,html{margin:0;padding:0;border:0;color:#000;background:#fff
}body,dd,dt,html,li,p,td,th{font:1em Arial,Helvetica,sans-serif}h1,h2,h3,h4,h5,h6{font-family:
Arial,Helvetica,sans-serif}h1{font-size:2em}h2{font-size:1.5em}h3{font-size:1.2em}h4{font-size:
1em}h5{font-size:.9em}h6{font-size:.8em}a:link{color:#00f}a:visited{color:#009}a:hover{color:
#06f}a:active{color:#0cf}
```

No comments and minified (no spaces, all code on one line)

Size: 388 bytes

Reducing the size by 27%

6 Conclusion and Possible Future Work

In the conclusion of this thesis the subproblems from the introduction will be brought up once again. This time with comments on how each problem was handled. As an ending of this chapter, a description is presented of how the work would have been extended if more time were given and what to do differently in the future.

How do HTTP, HTML, CSS, JavaScript and images affect performance?

During the work of the thesis it was discovered that the handling of the HTML, CSS and Javascript could lead to severe bottlenecks when loading the web site. The knowledge of what pitfalls existed and how they could be avoided was very important. Images are usually the biggest resource of a web site, therefore an important aspect when dragging down the overall size of a web site.

What tools can be used to properly measure the web performance and how is the result interpreted?

The tools used were Yslow, Pagespeed and webpagetest.org. After acquiring a greater knowledge of the tools, webpagetest.org was used in the majority of the time because of the automated testresults generated as waterfall charts.

Can web optimization be automated?

The aspects where optimization could mostly be automated was Http size, images, CDN and compressing code. But when handling the Http requests, caching and loading resources in certain manner that best results needs to be achieved, it should be done manually.

How should a company proceed to optimize it's already existing web site?

The optimization part vary a lot depending on the size of the web site and how it is used. The general rule is to find the bottleneck of the web site through testing and analysing. Then the bottleneck should be eliminated with techniques used in this thesis.

How can caching be implemented properly and what are the pitfalls?

When caching, it is important to only cache the static content of a web site. The pitfall of caching dynamic content could be avoided for example by versioning the content to force an update.

Can a checklist be developed for guidance on how to optimize a web site?

A checklist can only be developed in such extent that they give general guidelines on common problems of a web site’s performance. For further optimization it is important to analyze the specific web site.

The thesis ran during a limited space of time and lead to prioritizing the topics that was most fundamental to learn in a course about web performance optimization. For possible future work upon this thesis our main objective would be expanding the topic about Ajax for managing asynchronous Http requests. The topic about handling the mobile platform as addition to the desktop platforms would also be added if there was more time.

As a review on what has been accomplished throughout the process of creating a well elaborated course materials, it was a good idea to divide the work into phases. The divison of the work generated for us a better understanding of each topic.

The information gathering part of the thesis provided an overview on what aspects that was important, in order to acheive a well optimized front-end.

To better understand and deliver well-elaborated course materials, testing was the most important phase. The testing was after all what was proving that the theoretical information was correct. Testing was also providing information that will be more comprehensible in Eduments future course.

A final part on how proper testing is done and how the results were analysed was applied into the course materials. Also some good practices from global companies was added. This, in order to give future course participants guidance to how they could them selves analyse testing results and properly optimize on their own.

After all, the purpose of the whole topic of web performance optimization is to provide a better user experience. It was following this point of view the topics were selected for the thesis and the course materials.

7 Terminology

Client
A client is a computer hardware or software that access a service made available by a server.

Conditional GET
A conditional GET is an HTTP GET request that may return an HTTP 304 response (not modified since last GET).

DOM tree
The tree created by the browser when parsing HTML. Every element is represented by a node in the tree.

DNS lookup
A DNS Lookup is when a device asks a DNS server for the IP address of the targeted server.

Element (HTML)
An individual component of an HTML document.

Event
Triggered by the browser or the user. E.g. when clicking on an object.

Expiration date
The date a cached resource expires. After this date a new version of it will be requested.

Far future header
A header that will not expire in the near future. Used when caching.

First view
The view of a web page the first time it is visited.

Gzip
A compression method used when sending data on the Internet.

Host
The server where the requests are sent to. Stores the web site.

.htaccess
A directory-level configuration file supported by several web servers. It allows for decentralized management of web server configuration.

HTML document
The content of the HTML file.

Huffman coding
An entropy-encoding algorithm used for lossless data compression.

Interlacing
A method of encoding a bitmap image such that a person who has partially received it sees a degraded copy of the entire image.

Lossless compression
When compressing images without quality loss.

Lossy compression
When compressing images with a quality loss.

Metadata
External information about the data.

Minifying
Compressing code by stripping it of all comments and white spaces.

Parsing
The process of analysing a string of symbols, either in natural language or in computer languages.

PNG chunk
Contains certain information about the image.

Second view
The view of a web page when it is visited the second time.

Style sheet
The CSS. Used to style web page.

8 References

Google (2013) *Web Performance Best Practices*. https://developers.google.com/speed/docs/best-practices/rules_intro [2014-05-16].

GTmetrix (n.d.) *YSlow: Add Expires headers.* http://gtmetrix.com/add-expires-headers.html [2014-05-16].

Minnick, C. (2013) *Is HTML5 Safe to Use?* http://www.minnick.com/is-html5-safe/ [2014-06-19].

Senior, W & Wahlin D (2009) *Building High Performance Web Applications*. Microsoft Corporation.

Simonstl (2009) *How to Minimize HTTP Requests to Speed Up Web Pages*. http://answers.oreilly.com/topic/489-how-to-minimize-http-requests-to-speed-up-web-pages/ [2014-05-16].

Smith, P. (2013) *Professional Website Performance.* John Wiley & Sons, Inc.

Souders, S. (2007) *High Performance Web Sites*. O'Reilly Media.

Souders, S. (2009) *Even Faster Web Sites*. O'Reilly Media.

Strangeloop Networks (2012) *2012 Holiday Performance Guide.* Strangeloop Networks.

W3Schools (n.d.a) *HTML Introduction*. http://www.w3schools.com/html/html_intro.asp [2014-05-18].

W3Schools (n.d.b) *CSS Introduction*. http://www.w3schools.com/css/css_intro.asp [2014-05-18].

W3Schools (n.d.c) *JavaScript Introduction*. http://www.w3schools.com/js/js_intro.asp [2014-05-18].

W3Schools (n.d.d) *AJAX Introduction*. http://www.w3schools.com/ajax/ajax_intro.asp [2014-06-19].

White, J. (2012) *Why you should use a CDN for website optimization*. http://jackwhitey.hubpages.com/hub/cdn [2014-05-18].

Yahoo! (n.d.) *Best Practices for Speeding Up Your Web Site*. https://developer.yahoo.com/performance/rules.html [2014-05-16].