Responsive Web Design and Optimizing Loading Times on Mobile Devices for Enhanced Web Presence

Erik Modé

June 27, 2014
Abstract

Surfing the web has moved from the stationary computers to mobile devices such as smartphones and tablets. As the technology moves further and more devices come out on the market this is a trend that will continue. This makes it more important for web developers to adapt and make their websites fit not only for one device. The website needs to fit all devices. Since many of these devices are using wireless connections, the website performance has also become even more important in order to give the users a good experience.

Responsive web design is an approach for designing and developing web pages. By using the three techniques of flexible grid layout, flexible images and media, and media queries, websites can get a design that fits all devices. Responsive web techniques will also make the web page perform better on each device since it makes the web page adapt to its environment. There are several performance techniques that can be applied to a website in order to increase performance and loading time. Responsive web design including performance techniques can together make a great impact on a website's loading speed and user experience.

This thesis embraces this approach for web development. By developing a design theory, a website emerges where these approaches of web development have been implemented. During the development of the website the experiences implementing this approach and techniques is discussed. The website developed for this thesis was able to finish loading below 4 seconds on a mobile device using a 3G connection. We argue that responsive web design and performance techniques can help a website immensely to increase its user experience and performance.

Keywords: web development, responsive web design, web optimization, web performance, web for mobile devices
# Contents

1 Introduction ................................................................. 5
   1.1 Aim ........................................................................ 6
   1.2 Demarcations .......................................................... 6
   1.3 Method ..................................................................... 6
   1.4 Outline ..................................................................... 7

2 Responsive Web Design ......................................................... 8
   2.1 Benefits of Responsive Web Design ............................... 10
   2.2 Limitations of Responsive Web Design ......................... 10

3 Performance Techniques ....................................................... 11
   3.1 Pictures .................................................................... 11
      3.1.1 Compress images .................................................. 11
      3.1.2 Sprites .................................................................. 12
      3.1.3 Picture fill ............................................................. 12
      3.1.4 HiSRC .................................................................. 12
      3.1.5 SVG images .......................................................... 13
   3.2 Reduce HTTP Requests ................................................ 13
      3.2.1 Combine files ....................................................... 13
3.2.2 Lazy loading .................................................. 14
3.2.3 Icon fonts .................................................... 14
3.2.4 Avoid Redirects ............................................... 14
3.3 Deflate ............................................................ 15
3.4 Cache .............................................................. 15
  3.4.1 Content delivery network ................................. 16
3.5 "Minimizing" files ................................................. 16
3.6 The Justificatory Knowledge ................................. 17

4 Creation of a Website ........................................... 19
  4.1 Creating a Shell .................................................. 19
  4.2 Making it Responsive ............................................ 20
    4.2.1 Applying the Flexible Grid ............................. 20
    4.2.2 Applying flexible images ............................... 20
    4.2.3 Applying Media Queries ................................. 21
  4.3 Making it Optimized ............................................ 23
    4.3.1 Optimizing Images ....................................... 23
    4.3.2 Applying Picturefill .................................... 25
    4.3.3 Applying Lazy Loading .................................. 26
    4.3.4 Optimizing the Video .................................... 28
    4.3.5 More HTTP request reduction ........................... 29
  4.4 End Result of the Website ................................... 29
    4.4.1 How Responsive is the Website? ........................ 30
    4.4.2 How Optimized is the Website? ........................ 30

5 Concluding Remarks ............................................ 33
5.1 Conclusion ......................................................... 33
5.2 Discussion ......................................................... 33
5.3 Future Research .................................................... 35

A HTML Source Code ............................................... 38
B CSS Source Code .................................................... 43
C JavaScript Source Code ............................................ 46
D Screenshots of the website ......................................... 49
Acknowledgements

I would like to thank the people that have been helpful to me in my work writing this thesis. I would like to especially thank Sebastian Ekström who provided me with technical expertise, and my supervisor Andreas Hamfelt. Thank you for your help and guidance.
Chapter 1

Introduction

Nowadays surfing the web have moved from stationary computers to be more and more done on mobile phones and tablets. With the technology moving forward, more mobile devices hits the market every year. It is clear that this is a trend that will not slow down. But when using mobile devices for surfing the web, getting a good internet connection can be a problem. Especially where the 3G networks are overburdened or you are located on the countryside, or in undeveloped countries where the mobile network coverage is not as good as in bigger cities. This causes problems with slow loading times surfing the web with mobile devices.

Statistics show that the top 1000 web pages averages around 1.4 MB (HttpArchive, 2014). The third generation turbo 3G can have a maximum speed of 14.4 Mbps (1.73 MB/s). That means the average site would take just under a second to download. But we all know that the top speed is very seldom reached on our phones. In USA the average speed is around 2 Mbps or 0.24 MB/s (Sullivan, 2012). There it would take around 5 seconds to download the same average site.

Slow loading times are also due to phones memory and CPU. Because even though the web page is downloaded, the phone still needs to process all code and images. Because a phone’s memory, CPU, and cache are much smaller than a desktop computer’s, it takes longer time for a phone to process the information.

Surveys have also shown that slow loading speeds is one of the biggest factors in user frustration (Lohr, 2012). They have also shown that 1 in 4 people abandon a page that takes 4 seconds or longer to load (Everts, 2012). 4 seconds have been identified as the threshold of acceptability for retail web page response times by Akamai (2006). Between 4 to 8 seconds have also been identified as the attention span for people using mobile devices (Oulasvirta et al., 2005). Another article also claims that only one second slowdown in page loading could cost a company like Amazon 1.6 billion dollars each year (Eaton, 2012). Another example is that Google would lose 8 million searches every day if their service would slow down with only 400 milliseconds (Brutlag, 2009). It is clear that major websites like Google and Amazon would lose big amounts of revenue if their websites perform bad.
A solution to the long loading time problem could of course to make the internet speeds better. But that means a lot of technological improvements for mobile devices, as well as big infrastructural investments in expanding and improving the mobile networks’ coverage. The latter point is a big problem, especially in remote areas and undeveloped countries due to the high costs.

An easier and better solution would instead be to try to minimize the size of each web page. By using certain techniques web programmers could make a web page’s size significantly smaller. An example of this is when Youtube managed to make their web page go down from 1.2mb to only 98kb (Zacharias, 2012). By using responsive web design, the size of the page can be smaller but it also optimizes the user experience of the web page for all devices, since it gives the website a flexibility to adapt to them (Mohorovicic, 2013). The loading speed is also a factor that can be used for competitive advantage. With better features, appealing user interface, and fast loading times, will give a website the edge over its competitors (Souders, 2008).

1.1 Aim

The aim of this thesis is to develop a responsive and optimized website in accordance to design theory (Gregor and Jones, 2007). In order to fulfill this aim, constructs of responsive and performance web development techniques will be implemented into a website. The website loading time aims to be measured under 4 seconds, in accordance to the acceptable loading times for websites (Akamai, 2006).

1.2 Demarcations

This research will focus mostly on front end development of websites. Back end and server side implications on the website performance will not be taken into consideration. No user usability study is performed on the website developed.

1.3 Method

The research method used for this paper is design theory in accordance to Gregor and Jones (2007). This framework is a conceptual model based for information technology (IT). When conducting design theory research, purpose and scope of the theory that applies to the artefact in question must be described. Also, constructs, and principle of form and function should be described. The artefact mutability must also be stated and testable propositions must be made, with justificatory knowledge. Principle of implementation must also be stated and a proof that the artefact has been made in form of expository instantiation.
1.4 Outline

This paper starts with describing responsive web design and why it is important. It continues to go into further detail on those techniques that help improve the web page performance. Next it describes the process and experiences developing a homepage including the earlier described techniques, to learn how this website can become responsive and optimized. The last chapter describes the concluding remarks of the research and discuss further interesting areas of research.

In accordance with Gregor and Jones (2007) research method design theory, the *purpose and scope* is stated in chapter 1. In chapter 2 and 3 the *justificatory knowledge* is described, in particular in 3.6, including the *constructs* of this theory. Chapter 4 includes the *principles of form and function* and *principles of implementation*. It also states the *testable propositions* and this design theory’s *expository instantiation*. The concluding chapter 5 states the results of this design theory and discuss the *artefact mutability*. 
Chapter 2

Responsive Web Design

This chapter will describe the concept of responsive web design.

Responsive web design is the collective name and unified approach of three already existing web techniques, flexible grid layout, flexible images, and media queries. It was first mentioned by Ethan Marcotte (2010). He says that a responsive methodology is more than altering the website layout upon monitor sizes. It is about designing the web pages to fit the smallest monitors first and then progressively enhance the design of the web page as the monitors becomes larger. This is opposite to the traditional design where web pages are usually designed fixed to fit a desktop monitor and then scaled down for smaller devices. Instead, responsive web design will make the web page respond to its viewers and how the viewers are viewing the site. By letting the web pages be able to be flexible, the web developers don’t have to make separate versions of the web page depending on devices. The same web page respond to the device and the layout builds up accordingly. The responsive web design principles give the developers a great tool for designing web pages that works, and looks great in a wide range of devices. Responsive web design is also called by other names. Such as fluid design, liquid design, adaptive layout, cross device design and flexible design to name a few (Frain, 2012).

Responsive web design has three key ingredients (Marcotte, 2011).

- A flexible grid based layout,
- Flexible images and media,
- CSS Media queries

Flexible grids are columns that have relative widths, related to their containing elements, using percentages (%) or “ems”(em) as their units of width. Percentages are self-explanatory, but “ems” are a little harder to understand. An em is equal to the current font size. For example, if the font size is 12px in a document, 1em is equal to 12px. When we use these dimensions, the grid can resize as the view port is changing. This is instead of using fixed static pixels. The grid system also helps to give the homepage a consistent layout. The grid should be tailored for the
content and the page dimensions of each individual home page. Therefore, a grid system can look different for each web page. Figure 2.1 show an example of how a grid system could look like.

![Example of a grid system](image)

Figure 2.1: An example of how a grid can look like. (Marcotte, 2011)

Flexible images are images that can move and scale proportionally in relation to their container and view port size. Besides resizing the images, they can also have multiple versions for different resolutions or network speed. This will keep the images looking good over different sizes of view ports as well as improving the performance of the web page depending on the device.

Media queries are a module in CSS that allows the developers to build different layouts for the web pages using the same HTML. They are conditional statements that can identify media types and the size of the view ports. This enables the web page to change layout depending on the screen size used by the particular device. By using media queries the developer can change the design of the web page depending on the view port size. The same HTML and CSS file can therefore be used for the different designs adapted to the different devices view ports. Down below is an example of media queries in CSS. With that code the appearance of the web page can be changed depending on its width. In this example there is one design for fewer than 500 pixels width, between 500 and 800 pixels width, and over.

```
@media screen and (max-width:800px)
{
...
}

@media screen and (max-width:500px)
{
...
}
```

Responsive design is mainly to create a good design that is flexible depending on user devices and thereby increase the user friendliness. But to make the web page responsive can also increase its performance as well.
2.1 Benefits of Responsive Web Design

The benefits of responsive web design are many. Responsive web design is content focused; it makes the web pages device independent. In the long term it will save time and money, because it’s easier to maintain. It also gives the websites a better and more consistent user experience and usability.

Responsive web pages also have a single URL serving the same HTML, unlike fixed web pages that have one for URL for the mobile version and the desktop version. This makes changing the content much easier since only one version has to be changed. One URL also makes sharing the web page’s URL, on for example social networks, much more user friendly since the people clicking that link come to the right version of the page regardless of the device they are using. Responsive web design makes the web pages maintain their flexibility and user experience. It also enables consistency on the web pages on all different devices. Since the market of devices grows bigger and bigger and new devices and platforms for web browsing appear, give more loyal customers and thereby improve the company’s market share (Mohorovicic, 2013). Google also recommends the industry the best practice of using responsive web design (Google, 2014c).

2.2 Limitations of Responsive Web Design

Responsive web design does not only have advantages. Making a page responsive usually means more work than creating a normal web page. Since responsive web design project are more time consuming the cost will naturally increase. Making a responsive web page can be up to 10-20% more expensive (Mohorovicic, 2013).

Another problem with responsive designed web pages is that they are made using the latest HTML and CSS techniques. For example, media queries are not compatible with older web browsers and mobile devices. Since all web browsers do not support the new web standards and therefore web pages that uses advanced modules might not work correctly on those browsers. (Kadlec, 2012) Responsive images are one of the unsolvable problems of responsive web design but there have men many attempts to solve it, until a real web standard is settled down by W3C (World Wide Web Consortium) (W3C, 2013).

Responsive web design is also not well suited for advertisement. Since it is about minimalistic design it is harder to place banner advertisement within the responsive design than it is in fixed designed pages. Advertisement is what many web pages rely on as income source and for them this might be a problem.

But even though responsive web design does have its limitations, the benefits outweigh the limitations. For most organizations, a responsive web designed site would give improved results and long-term savings (Mohorovicic, 2013).
Chapter 3

Performance Techniques

In this chapter some techniques that can be used to increase the website performance will be described. Some of these techniques are a part of the responsive web design approach that is described above, and are related to the three key ingredients, flexible grid, flexible images, and media queries.

The idea of these techniques is to decrease the size of the web page. They also decrease the amount of information that need to be processed before it can be showed to the user. In the end a website using some of these techniques would load faster and give the user a more smooth experience.

3.1 Pictures

Images are the biggest source of weight that makes web pages heavy. The average web page is, as earlier described, 1.4 MB. 1 MB of that consists of images (HttpArchive, 2014). So by optimizing the images, a lot of weight can be saved in order to improve the loading time of the web page.

3.1.1 Compress images

The best way of minimizing the size of the images is to compress them. Compressing an image can save up to 80% of the image size. There are many online tools that provide this feature, such as “TinyPNG” (TinyPNG, 2014), Kraken, or “Image Optimizer” in Visual Studio, and so does many CMS (Content management system) tools. Compressing images can often be done without a loss of quality of the images. Many image files have extra data embedded in them that is not needed (Sowell, 2013).
3.1.2 Sprites

Next thing, one can do, is to use image sprites to group multiple images into a single encoded file. Since the images are bundled up in one file, only one download need to be made, instead of one for each image. This saves a lot of HTTP requests and thereby bandwidth, especially if there is a lot of images on the web page. But do it carefully, because it can be difficult to maintain those images and the sprite can become too big when encoded, if there is too many images in the same sprite. Therefore, the sprites need to be planned carefully (Wroblewski, 2012).

3.1.3 Picture fill

Another trick is to use “Picturefill”. It’s a substitute for the proposed picture element that hopefully will be implemented in HTML in the future (W3C, 2013). Usually the image tag is used for images, but if there are multiple versions of an image on the web page, the web browser will download all the images, even if only one of them is shown. This is not very good for the performance and not very bandwidth friendly, and this is where Picturefill comes in. It allows you to specify low, versus high resolution on your images. The browser will then be able to choose the best image that fits in to the screen of the used device, and download only that one (Sowell, 2013). For an example of how this looks like in the HTML code, see the lines of code below.

```html
<span data-picture data-alt="Image text">
  <span data-src="small.jpg"></span>
  <span data-src="medium.jpg" data-media="(min-width:400px)"></span>
  <span data-src="large.jpg" data-media="(min-width:800px)"></span>
  <span data-src="extralarge.jpg" data-media="(min-width:1000px)"></span>
</span>

<!-- Fallback content for non-JS browsers. -->
<noscript>
  <img src="small.jpg" alt="alternative image text">
</noscript>
</span>
```

3.1.4 HiSRC

HiSRC (short for High Source) is a jQuery plugin that helps to make images responsive and helps to reduce even more weight from the files. The developers can make the images detect a fast or slow network. Depending on the network connection speed, the images will load in a low or high resolution. It can also choose the resolution for an image depending on the resolution on the user device’s screen (Schmitt, 2014). A code example of HiSRC can be seen in the lines of code below.
3.1.5 SVG images

One more trick that can be used is SVG images. SVG stands for “scalable vector graphics” and is standard for vector images. SVG provides lightweight images that are based on XML to draw vector shapes. SVG images are often smaller in size than their corresponding JPG and PNG counterparts. But complex shapes in the image will create a more complex and extended SVG mark-up. SVG is most easily used for simple shapes, for example a circle. Unfortunately, older versions of Android do not support SVG (Sowell, 2013).

SVG Images can be used together with so-called data URI’s, where the image is sent through code and not as an image file. Data URI’s use a special encoding scheme that is called base64 which was originally designed for email. It translates binary data into a text string that is compatible with ASCII. By removing one image and making it a part of the code, it therefore saves a HTTP request since the image doesn’t have to be downloaded. Instead, the client’s CPU needs to translate the code back to an image. Because of that, this needs to be used carefully, since complex images will take a lot of power from the CPU to render. This is especially important for images on mobile web pages, since the CPUs used in mobile phones are not as powerful as they are in desktop computers (McLachlan, 2013).

3.2 Reduce HTTP Requests

HTTP requests are when a client requests a file from a web server. Every file that needs to be loaded on a webpage adds precious extra milliseconds each in loading time. In order to improve the web page load time, reducing the number of files and thereby HTTP requests is of much importance.

3.2.1 Combine files

The easiest thing to reduce the amount on a web page, is to combine files so that the client only have to make a few requests. Combine the CSS and JS files into ideally one file of each. Two or
three files of each are okay too, but the fewer the better (Sowell, 2013). In the picture section 3.1, it is described how images can be combined using sprites.

3.2.2 Lazy loading

Another good trick to reduce the HTTP requests is not to load images until they are inside the users view on the screen of their device. Images and other files will only load when the user reaches the part of the page where they can be viewed. This technique is called “lazy loading”. This also applies to libraries. It’s not necessary to load a JavaScript library that is targeted for a iOS device on an Android device or a desktop, and vice versa (Magazine, 2013).

3.2.3 Icon fonts

By using so called icon fonts, the website can use fonts instead of images for icons. The icon fonts allow the icon to be scalable without degradation. There is a number of free icon fonts online (e.g. www.fontello.com). These services allow the developer to customize their icons as well as generate the CSS code for the icons. By using icon fonts the website can reduce the amount of HTTP requests since there is no image files that need to be loaded for the icon (Sowell, 2013).

3.2.4 Avoid Redirects

A redirect is when the server returns a response with another URL, which the browser should contact, in order to get the content it was expecting. Usually one redirection leads to another. For every redirect the browser need to go over the procedure of all the HTTP requests with all its latency. This slows down the loading time a lot, and the time for the user to reach the expected content. An example of a website using redirects is when the user visits a website that has both a desktop version and a mobile version. The assumption is that the user would enter the site based on the version they wanted. But the reality is that user’s usually just type in the host name of the website. That means that the server need to figure out what version is the correct one for the user. This will lead to at least one redirection to the correct version of the website, often even more.

Avoiding redirects is vital for the website performance. That is why it’s important to implement responsive web design on a website because then only one domain is used for all versions. The web application serves both desktop and mobile versions of the website and the design is purely based on the device. Thus, the website with its HTTP requests only needs to be loaded one time (Zakas, 2013).
### 3.3 Deflate

Another important technique to reduce the loading time for a web page is to use deflate compression. Deflate compression is a data compression algorithm that is not covered by any patents, which has led to a widespread use of it, for example ZIP (Salomon, 2004). Deflate is also used by Gzip which is one of the most popular compression applications for web traffic. Simply described Gzip compresses the files on the server, sends them over the internet, to the client where they are decompressed. The file size can be decreased by up to 60-80% and because the file size is decreased by so much the transfer goes much faster. Gzip is supported by most browsers and servers (Sowell, 2013). In the listing down below is a http request to Uppsala University’s homepage. In there we can see that when the browser sends its requests to the server it also tells the server that it accepts Gzip encoding. But both client and the server must handle Gzip for it to work. The server must be setup to handle the Gzip encoding requests and thus make the Gzip work.

```
GET http://www.uu.se/ HTTP/1.1
Host: www.uu.se
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
        image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36
            (KHTML, like Gecko) Chrome/33.0.1750.146 Safari/537.36
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8,sv;q=0.6
```

### 3.4 Cache

The best way to reduce loading times is to not download at all. This is what caching is all about. The first time a client’s browser requests files from a server, the files need to be downloaded either from the server or a caching proxy, which is in between the server and the client. If the files can be cached, the client doesn’t have to download them again next time it requests them, or at least parts of the files. Using cache can even make a web page available to browse offline. This is done by using different kinds of HTTP headers such as “If-Modified-Since”. This header allows the browser to ask the server if there is a new version of a file, if not it uses the cached one. The Expires header let you set a timer for how long the browser should use the cached files until the timer has expired and then it downloads the cache again. This is useful since the browser does not need to use the network during this time, unlike the “If-Modified-Since” header (Sowell, 2013). In the listing below is the HTTP response from the request made earlier to www.uu.se. In here we can see an example of the If-Modified-Since header that is used by the web page. We can see a date of when the file was modified last time, and thus the browser knows if it needs to download them again or not.
3.4.1 Content delivery network

As said in the part above, a caching proxy server can be used. A more common name for this is CDN, which stands for content delivery network. This is a special kind of servers that are spread out on several locations that specializes in distributing file, like images, CSS files, JavaScript files and videos to be able to spread those more easily on a broader geographical scale (Sowell, 2013). For example if I want to visit a Chinese server located in China, my Swedish client might be able to get some of the important files, such as JavaScript etc. from a CDN in Stockholm. This helps the loading time to go much faster as well as minimizing the load on the main server. Another real life example is that all jQuery plugins are able to be downloaded throughout CDN’s. Many web pages uses CDN’s by Google where they distribute several libraries (Google, 2014a). A web page can also host their files on such servers around the world in order to increase performance. Geographical proximity is something that affect performance greatly and therefore content delivery networks are something that gets more common nowadays as internet grows bigger (Armstrong and Ward, 2013) (Sowell, 2013).

3.5 "Minimizing" files

The last thing the developer can do to save some extra bytes on the web page size, is to minimize, or as it is also called, "minify" the HTML, CSS, and JavaScript files. Minifying means that all the unnecessary bytes are removed from the files. These unnecessary bytes are things such as extra spaces, line breaks, and indentation. The white-spaces etc. are there for the developer to make the code more easy to read and write. But when the development is finished minifying the files will make them more compact and thereby smaller in size. This does not only increase the downloading speed because of the smaller file sizes, it also increases the parsing and execution time.

The minifying can be extended even further in the CSS and JavaScript files by changing the variable names into shorter versions. But of course the HTML files need to be updated in order to keep the web page working. The optimization of the web page’s loading time by minifying the files becomes greater the bigger the web page is and the more line of code it has (Google, 2014b). An example of minified CSS code can be seen below.
Before minifying the CSS, the code usually contains a lot of white spaces and line breaks to make the code more easy to read. After minifying the CSS all the unnecessary space is removed for optimization, as can be seen below.

```
.main-header { 
  height:80px;
  line-height:80px;
  position:absolute;
  top:0;
  width:100%;
  display:block;
  z-index:5
}
```

### 3.6 The Justificatory Knowledge

In these two chapters (2, 3), the background theory has been described about responsive web design and performance techniques. It is with this background knowledge we will be able to develop our design theory and website. It is with this we are able to make our justificatory knowledge. We will choose parts of the background theory to structure the website. We will use the concepts and techniques that are reasonable to be implemented into the artefact, which is our website. Down below is the summary of what we will make use of to create the structure of our website.

1. Responsive web design
   (a) Flexible grid layout
   (b) Flexible images and media
   (c) Media Queries

2. Performance techniques
   (a) Image and video compression
   (b) Sprite
   (c) Picture fill
   (d) SVG images
   (e) Base64 encoded image
   (f) Combine files
   (g) Lazy loading image
   (h) Minifying files
Using these concepts and techniques we will be able to create a website, that will be responsive and optimized. Combined together the benefits from all techniques should make the website in such way that it should be able to load under 4 seconds. In the next chapter the experiences of developing the website is described. When the concepts and techniques have been implemented into the website we can test if the website will perform as good as we aimed for. This is in accordance to our design theory’s testable propositions. In section 4.4 of the next chapter we test and evaluate the finished website.
Chapter 4

Creation of a Website

This chapter will describe our way of developing a responsive web page in order to optimize its loading times. We will develop this website and design theory by implementing the constructs described in the earlier chapters, and stated as the justificatory knowledge in 3.6. They will be implemented in a step by step manner and along the way we discuss our experiences doing so. In this chapter the principles of form and function and principles of implementation of the website will also take shape. We will exemplify our development with expository instantiation and in the end of this chapter we will evaluate our websites responsiveness and optimization with our testable propositions.

First we will create a shell of the web page, second we will make it responsive, and thirdly we will optimize it in order to increase the loading speed even further. The appendices can be viewed for more details and understanding when reading this chapter.

4.1 Creating a Shell

In order to make the website we firstly need some content to work with. Described in the introduction chapter was that optimized sites was important for the earnings of e-commerce sites. Therefore, a prototype of a web shop was created. To make it visually appealing a video and pictures were added, as well as good looking fonts and icons.

Right now our prototype of the website might have some kind of content. But it’s not at all where we want it to be. The website has become heavy with all the visuals we put in to make it better looking. In chapter 2 it is described that when applying the responsive web design concept, the mobile should come first. The website should be designed to fit mobile devices first and be able to scale up to bigger screens such as desktops. Right now this prototype lacks this approach.
4.2 Making it Responsive

It is time to apply the concept of responsive web design to the website. Chapter 2 describes the three parts of this concept, making a flexible grid, flexible images, and media queries. The approach of mobile first should also be kept in mind. Mobile first is about to always think about to make the page work well on mobile devices, before larger devices. This also helps to make the website compatible with older devices that do not support JavaScript and media queries.

4.2.1 Applying the Flexible Grid

The flexible grid was described in chapter 2 as columns that have relative widths, related to their containing elements. Making them flexible can be achieved by using percentages or ems for their dimensions instead of the pixels, which are inflexible. So naturally in order to make the page flexible we want to use the flexible dimensions and not the inflexible. All dimension variables in our CSS file should be percentages and ems. Down below is a simple example.

width:100px;
Should be:
width:100%;

The idea with the flexible columns is that they should be able to change depending on the device view port size. On a small mobile device it’s most convenient to have the columns on top of each other, vertically, creating a scroll down design. This will make it easy to scroll down through the content of the page, without making the columns squished together. On a desktop screen the columns have enough room to be beside each other, horizontally, which creates an easy overview of more information within the same view port. But to be able to make our columns change behaviour like this we need to use media queries. This will be explained in 4.2.2.

A flexible grid system framework can also be used, e.g. Bootstrap (http://getbootstrap.com/). Then all the widths of the columns don’t have to be manually applied. But there is other disadvantages coming when using this. Firstly all columns will have set class names. For example, a column class in Bootstrap could be ”.col-md-4“ meaning that four columns will be used. When the website later will fit into a phone it will use only one column that takes up the full width of the screen in our design. Then the class names will be incorrect. If the grid system is written manually, the class names will be correct and make more sense for the developer giving him or her better control over it.

4.2.2 Applying flexible images

Most websites have images because it makes them look better. A Picture tells more than a thousand words. A website with pictures looks much more appealing than one which only has text. Because of this our web page will of course also use pictures.

But there are problems with making images responsive, as explained in the earlier chapters. These are problems we need to deal with. Firstly we should solve the images size. Just like our
grid, we want our pictures to be flexible in order to fit the design for all devices. By adding the
flexible dimensions for our images, just like we did to the grid, they are able to shrink, and grow,
depending on its container. Down below is a simple example of how we do this.

<!-- HTML -->
<img src="static/img/small2.jpg"></img>

/** CSS **/
img {
  max-width: 100%;
  height: auto;
}

The second problem to solve with the images is their resolution. To use a very high resolution
image would look good on all the devices. But on smaller device, having very high resolution,
images becomes redundant due to the small screen of the device, even if it has a high pixel
density screen. We also need to take into account that the higher the resolution an image has,
the bigger its file size becomes. Therefore, it would be better to have smaller resolutions on the
image files as the device screen goes down in size, and vice versa. The solution we used to do
this is called "Picture Fill". How that works, and how we resolved it is described in the next
section. 4.3.2.

4.2.3 Applying Media Queries

The website is pretty flexible already with the flexible grid and images. But the key to make our
website responsive is to use media queries. It will take the flexibility to a whole new level. With
them the website can identify the media type and the size of the view port in question.

We want the web page to change layout so that it will look good and work properly on all
possible devices. The easiest way to do this is to find breakpoints where the website start to
look bad in the different sizes. The easiest way to find these breakpoints is to slowly scale up
and down the browser window on the computer. Then we can see at what measurements of the
web browser, the website starts to look bad. For example maybe some text or images start to
disappear behind other content, or overflow others. It is right before this happens, we want to
apply our media queries. It will make the layout of the content change so it can continue work
and look good together in the new screen size.

Let’s implement this to our website. The website should be a prototype for a web shop, so
there are three articles, of coffee beans, for sale on the page (See a screenshot of the web page
in AppendixD). On a desktop screen it would look good for the three articles be beside each
other horizontally. But on a smaller screen of a mobile device they would be squished together if
placed that way. On smaller screens it fits better to place them under each other, vertically. In
the line of codes below we can see how this can be implemented on our site using media queries.
This can also be viewed in the parts of the source code that is in Appendix B.
In this code we can see that we have a media query where the break point is at 60em. When we are below this width, which is slightly above an iPad screen size, the width of our articles is 100%. This means that our articles will take up all the width space in their container. Therefore, the articles will be on top of each other, vertically. Above the size of 60em, the width of the articles is instead 33.33%. At this screen size they will instead move beside each other horizontally, since they now only use a little less than a third of the space.

We also used media query on the top bar menu. On a smaller screen of a mobile device it is better practice having a menu button instead of having the menu links being vertically on a bar. Because then the links would be squished together. This time we choose to have the breakpoint at 43em. When the screen size gets to the breakpoint, we want the menu bar with the four links to become a menu icon, and vice versa. When the size is bigger than the breakpoint we write in the CSS, ”display: none;” which means that menu icon will not be shown, and instead the menu bar will appear. How this is implemented can be seen in the code below.

In a similar manner the rest of the web page can change appearance according to screen size. We will apply this on our menu and buttons, etc. But we will also use media queries for optimization. More about this is described in 4.3.

As described in the earlier section we would like to use a resolution for the images that fits better for the device that it is viewed on. Since media queries can be used to sense the resolution of the device, it can also be applied for images. By having different resolutions on the image on our server we can in the media query choose which one to load depending on the devices’ resolution. See the lines of CSS code below.
But in order to get new sources for our image, we need to use the "background-image" tag. Because it is only if we use this tag we can change the source for our image in the CSS, where the media queries need to be written. This leads to a problem. The background image tag should not be used for content images. It should rather be used for styling, for example in header, and perhaps the body. Also, if the website content should be able to be managed in a CMS (Content Management System), no such tags can be applied to the images by the CMS. Since three of our images on the web page is content images, we should not use this tag. This means that we cannot use media queries to change the source for our images. We need to resolve this problem in another way. In section 4.3.2 our solution for this is described.

4.3 Making it Optimized

So far the website has become quite responsive. But we have still not reached our goal optimizing it yet. The goal is to not only make the site responsive but to also optimize it in order to make it load faster, especially for mobile devices. In chapter 3 techniques for doing so are described. Now we will try to apply some of them on our website, to make it enough optimized to work well on mobile devices.

4.3.1 Optimizing Images

As described in the introduction, images are the biggest cause for web pages heavy weight. So in order to get our website optimized, images is where we start.

The first thing to do when optimizing the images is to compress them. By using free online tools for compressing the images a lot of size can be saved. For this web page, Kraken (2014) was used, and about 60-75% of the file size was removed. Lossy compression was used in order to save extra bytes. The quality loss was not enough to weigh against the benefits of the file size that was removed.

Sprites is described in chapter 3.1.2, and for our smaller icons this is a good way to reduce HTTP requests from individual image files. That is because these icons are put together in the same file. In figure 4.1 the sprite that is used on our website can be seen.
Figure 4.1: This is how the sprite looks like. All three icons is put into the same image file.

In the buttons for the articles, we want to use the shopping cart. This is implemented in the CSS file by stating the position coordinates of the shopping cart within the image file using the background-position tag. If we would like to use the same icon again, somewhere else in our design, we could just give those coordinates again. This would save a lot of unnecessary HTTP requests, since it does not have to be loaded again. The line of code below show how this was implemented for our website.

```css
.button--primary {
    background: #bbb url(/img/sprite.png) no-repeat;
    background-size: 34px auto;
    background-position: 35px 15px;
}
```

An interesting note about these lines of code is that we used the background tag for stating the image source. This is something we tried with our content images in section 4.2.3, but later choose not to do. However, since the sprite is for styling and not content based, it is still within good practice.

Another way for reducing the size weight of the images is to use SVG images, as described in chapter 3. On our smaller images such as the icons this would fit pretty well. Using SVG images actually reduces the total size of the images compared to using the sprite. But on the other hand another problem arises, because now we will get a HTTP request for each icon, instead of one as we had with the sprite. The size reduction might not be enough compared to the few more HTTP requests they cause.

A third way to optimize these small images, the icons, would be to use Base64 encoding. That is also described in chapter 3. It creates a long string of ASCII code of the SVG file. This means that instead of downloading the file with a HTTP request to the web page, the code creating it is already inside the HTML file. But this needs to be applied carefully. If the image is too complex, the Base64 code would become very long. Thus, it would take longer time for the device CPU to render the code. This needs to be considered especially for the weaker CPU’s in mobile devices. So also here a comparison of advantages and disadvantages need to be made. Since the icons we use for our website are not very big, nor very complex, Base64 would work well, without have to worrying too much about the CPU’s capacity.

Base64 encoding is probably the best choice for these icon images on our website. Since it’s only four (including the three icons plus the menu icon) icons (including the three icons plus the menu icon) that are not very complex it will not put much pressure on the CPU for rendering. Also, we save four HTTP requests. If we would use more icons on the other hand the CPU capacity will be a problem. In that case a SVG sprite would be the best option, since it gives us one small size file that can be used for all icons. In the HTML source code in Appendix A all three solutions can be viewed, as a proof of concept.
Now we have paid much attention to the icon images, which are about styling our site. But on our website we also have bigger images for content that we want to look good, and at the same time be optimized over all devices. In order to do that we need to apply some other techniques, and in section 4.3.2 and 4.3.3 below, how these techniques are applied will be discussed.

### 4.3.2 Applying Picturefill

Picture fill is described in chapter 3.1.3. Using this technique we can let the website choose the resolution of the image depending on the size of the screen on the device that the page is viewed upon. When scaling down the image resolution on the smaller screens we are also removing a lot of kilobytes from the file.

In order to apply this to this website, firstly we want to make four versions of each picture. An extra-large with the width of 1000 pixels, a large with 800, and a small one with 240 pixels width. The height is related to the pictures’ width. But since the three pictures are going to be in a row beside each other on the bigger screens, we want to keep them with the same dimensions in order to fit well next to each other. Also, since we choose to keep the pictures beside each other in the big screen design we made the extra-large versions only 1000 pixels wide. If they would be alone, cover the screen from side to side, it might have looked better to have an even higher resolution.

Now that there are four different versions of each of the three pictures, it is time to implement them in our website. To do this we are going to use the Picturefill responsive image approach. It allows the site to recognize the screen size of the browser, and load the matching resolution image file accordingly. It uses media queries together with a JavaScript in order to make it work. In the lines of code below is the mark-up we use for our pictures in the HTML file.

```html
<span class="lazyload" data-alt="">
  <span data-src="static/img/small2.jpg"></span>
  <span data-src="static/img/medium2.jpg" data-media="(min-width:400px)"></span>
  <span data-src="static/img/large2.jpg" data-media="(min-width:800px)"></span>
  <span data-src="static/img/extralarge2.jpg" data-media="(min-width:1000px)"></span>
</span>

<!---- Fallback content for non-JS browsers. ---->
<noscript>
  <img src="static/img/small2.jpg" alt=""/>
</noscript>

</span>
```

In the mark-up we are able to write a minimum or a maximum value for each span element, which is accepted by all media queries in the CSS. The JavaScript loops through the pictures and finds out what sources that match the conditions. Using an open source JavaScript (by Jehl (2014)) makes it easy for us to make a rigorous implementation. See parts of the source code for the JavaScript in Appendix C. This is how the site loads the correct resolution file of the pictures.
This code actually looks a bit different from the one in the code example of Picturefill in chapter 3.1.3. Instead of a "data-picture" element we have made a class called "Lazy Load". This is because we later apply the Lazy load technique to our three pictures as well in order to optimize them even further. How lazy loading is implemented is discussed in the next section 4.3.3.

4.3.3 Applying Lazy Loading

Lazy loading is as described in chapter 3.2.2 and it’s a technique to load, for example images, only when they come inside the view port of the device. We want to implement this to our three pictures because it will save some weight and HTTP requests in the initial loading of the website. Since we placed them in the lower part of the site (again see Appendix D for screen of the site) they will not load until, or if, the user scroll down to that part of the website. The line of code below shows how the mark up in the HTML would look like if lazy loading would be applied by itself.

```html
<img class="lazy" data-original="img/example.jpg" width="640" height="480"/>
```

A JavaScript will then "listen" to the scrolling of the website, and when the image is within the view port an event will trigger the image to be loaded.

In order to implement lazy loading for the pictures, we need to create a class for the images with the HTML code using the normal "img" tag. But since we also want picture fill for our images which need a "data-picture" within the span tag this will cause a problem. A solution is needed that can fit Lazy loading and Picturefill for our pictures.

The solution applied on our website is to first apply lazy loading to the pictures. Then when the pictures are within the view port Picturefill can be applied. In order to implement them both we need to create a mark-up that fit the two techniques. In the lines of code in section 4.3.2 is how the mark up looks like. It looks similar to a pure Picturefill implementation expect for the "data-picture" that has been replaced with a class element called "lazyload". This is made so that the Lazy Loading JavaScript can be connected to this particular mark up. The event listener within the JavaScript can then call for the "lazyload" class inside the HTML and load the pictures.

But when the picture loads the Picturefill should also be applied. To do this we need to have the "data-picture" element within the mark up. So to do this we need to have a small script that ads this element to the mark up, when the Lazy Load event is triggered for the pictures. After the "data-picture" element is there the Picturefill can work properly, and thereby the right source file for the pictures can be loaded. In the lines of code below is parts of the JavaScript that makes this happen.
//Lazy loading
function checkvisible( elm ) {
    var vpH = $(window).height(),
        st = $(window).scrollTop(),
        y = $(elm).offset().top;
    return (y < (vpH + st));
}

//Activate Picturefill when the lazy image is in view.
$(window).scroll(function() {
    console.log('scroll');
    activateVisiblePicturefills();
});

$('document').ready(function() {
    activateVisiblePicturefills();
});

function activateVisiblePicturefills() {
    var redraw = false;

    $('.lazyload').each(function(i, elm) {
        var needsToDisplay = checkvisible(elm) && !$ (elm).attr('data-picture');
        if (needsToDisplay) {
            $(elm).attr('data-picture', '');
            redraw = true;
        }
    });
    window.picturefill();
}

The first function in this JavaScript is the scroll listener. When each Picturefill span come into the view port, the script will call for the "activateVisiblePicturefills" function which adds the "data-picture" element to its span container. It will then manually call the Picturefill function which lets the full Picturefill script run. Now the Picturefill can work properly and detect the pictures recently scrolled into view and download the correctly sized image files. Now we have combined both Lazy Loading and Picturefill to the three pictures. They will only be loaded when they come into the view port. The correct image resolution will be loaded depending on the screen size it is used upon.

To further improve this, would be to put a place holder for the pictures before they get loaded. This would make it look better since the pictures would not pop up into their place as suddenly as they might be perceived to do now in our design. To improve this even further a more rigorous implementation lazy loading could also be applied. There are many JavaScript libraries where Lazy Loading is further developed with extensions options such as threshold settings for the script to start. Then the image could be loaded a set of pixels earlier before they got into view. This would have been good for our application because then the sudden "pop up" of the pictures would happen outside of the view, and not being noticed. Also, these library scripts usually have a fall back for older version browsers where script doesn’t work. This is something that has been
applied to the Picturefill functions but not our Lazy loading. Both of them should have these in order to make the website work better on the old browsers. This is something that surely would have to be implemented in further development of the website.

4.3.4 Optimizing the Video

In the big screen view of our website design we have chosen to have a video in the background of the header. This is to make the visual appealing effect even higher on the desktop version, but also to show how a video can be optimized on the home page.

Compressing the video file is the first thing to do, and also the most obvious, in order to decrease the file size as much as possible. Secondly, the video should only appear on the desktop version of the web page. By using a JavaScript it is possible to make it only appear on the desktop version, if the screen size gets smaller. And if the browser is not compatible with JavaScript, the image is used instead. Since we manipulate the mark up in the HTML file when we want to show the video instead of the image, JavaScript is needed. CSS cannot be used to manipulate HTML and therefore media queries cannot be used in this solution.

HTML5 comes with the nice feature of streaming videos. The video element lets the video file be downloaded progressively to the user’s browser. Meaning that the video can start to play almost instantly in the browser and continue playing while the later parts are being downloaded. However, this still means that the video file needs to be downloaded. If the video is long the file will be large, giving a lot of weight to download. A solution to consider is to just have a short preview video in order to only download a smaller file. Then the user would be able to choose to see the whole video by a simple click. Then the full video could be streamed from another provider, such as Vimeo and YouTube. These providers have streaming solutions where the file is not really downloaded to the device by the browser. If applied to our website, the HTML implementation would look like the lines of code below. This includes the image throwback for smaller devices and non-JavaScript users.

```html
<img src="static/img/medium2.jpg"
     class="masthead-video--media"
     data-video-src="87701971"
     data-teaser-source="static/video/royal-b-roll"
     data-provider="Vimeo"
     data-video-width="500"
     data-video-height="281">
```
4.3.5 More HTTP request reduction

Now we have come pretty far in optimizing the website. A lot of HTTP requests have been removed already when the images got optimized. But a few more requests can still be removed. Using font makes the web page more visually appealing. By using default HTML fonts, HTTP requests can be saved, instead of having to downloading and using custom fonts. However, in our website we have chosen to use two custom fonts from Google fonts (http://fonts.googleapis.com). It will unfortunately give the website 2 more HTTP requests, but again, to make the website better looking and more appealing using these fonts helps. But the benefits of using Google fonts is that Google have a quite extensive CDN (Content Delivery Network), which make the download faster. Also, since a lot of web pages are using Google fonts, it is likely that the user have the fonts already cached in their browser, meaning that they don’t have to download them at all. Again visual design must be weight against performance, but because of the benefits with the Google fonts the performance lost is not that bad, considering the nice look the text get on the website.

When the website development is finished, the last touches of optimization can be applied. The last unnecessary HTTP request can be removed by merging our HTML, CSS and JavaScript files. In the end we want to have only one file of each. Especially when we are downloading and using JavaScript from libraries it is easy to get several individual files. But putting them within the same file is easy and it will only improve the performance of the website.

Lastly when the files have been merged together they can be minified as well, as described in chapter 3.5. Then the last bytes of weight will be saved from the website.

4.4 End Result of the Website

Now we have come to the end of our journey of creating a responsive and optimized website. It is time to see what has been achieved. In accordance to our design theory testable propositions we want to test if our website is able to perform as our predictions and aims. We will test the website with some tools in order to see how the website will perform both in load time and responsiveness.

The Google Chrome web browser has a developer tool where you are able to inspect web pages and its elements. The network usage of the web page can also be viewed and analysed. By using this tool we are able to get an overview of the HTTP requests, a rough measurement of loading time and the download size of the website.

Google also has a tool called ”PageSpeed Insights” (http://developers.google.com/speed/pagespeed/insights/) where the website performance can be tested. It takes the URL to our website and uses it twice, one time for a mobile user agent, and one time for a desktop user agent. The test gives a score that can range from 0 to 100 points. One score is given for speed, and one is given for user experience. PageSpeed Insights measures how the website can improve its performance on: (1) time to above-the-fold load, i.e., the elapsed time from the moment a user requests a new page, and to the moment the above-the-fold content is rendered by the browser.
(2) Time to full page load, i.e., lapsed time from the moment a user requests a new page to the moment the page is fully rendered by the browser. Above the fold is what can be viewed of the page within the view port when it's loaded.

We also want to test how long the loading time of the website is. By using an online tool named "Mobitest" by Akamai (http://mobitest.akamai.com/m/index.cgi) we can test our website loading time with a virtual mobile device, using a 3G connection. Another tool to measure the loading time is with the "Network Link Conditioner" feature of Apple’s "Xcode" that simulates a set of downloading speeds.

4.4.1 How Responsive is the Website?

Firstly, just by flexing the browser window on a desktop computer, the responsiveness can be viewed all over the screen sizes. Also, by trying the website on different mobile devices we can view the responsiveness over the different devices used.

Secondly the website is tested with Google PageSpeed Insights, described above. This test gives our website full points, 100/100, in user experience for mobile devices. Down below are the points measured that our website passed.

- Avoid plug-ins. Your page does not appear to use plug-ins, which would prevent content from being usable on many platforms.
- Configure the view port. Your page specifies a view port matching the device’s size, which allows it to render properly on all devices.
- Size content to view port. The contents of your page fit within the view port.
- Size tap targets appropriately. All of your page’s links/buttons are large enough for a user to easily tap on a touch screen.
- Use legible font sizes. The text on your page is legible.

This means that the development of our responsive web design on our website was successful. Because the responsiveness is good enough and the design works well on all devices, our site got a successful score on the user experience in this test.

4.4.2 How Optimized is the Website?

Maybe even more importantly, we want to test the loading time of our website. Again the easiest way to test the speed performance of the website is to try it yourself. By visiting our website on a mobile phone using 3G networks we see if the page load fast enough for our own preferences. But to test the speed more thorough we have to use some tools.

Google PageSpeed Insights also test and give points and feedback for the speed. This test gives our website 77/100 points in speed for mobile devices. It states that we passed 8 rules, but one
rule to consider fixing, and one that should be fixed. Below are the rules that we passed. As can be seen, most of those are also the techniques we applied through our development.

- Avoid landing page redirects
- Enable compression
- Minify CSS
- Minify HTML
- Minify JavaScript
- Optimize images
- Prioritize visible content
- Reduce server response time

The rule that should be considered fixing is "Leverage browser caching". This means that it would be good to apply expiry date or a maximum age on the HTTP headers for the static resources of the website. However, this is something that needs to be done server side, and not by the front end developers as our role is.

The rule that the test tool considered as "should fix" is to "eliminate render-blocking JavaScript and CSS in above-the-fold content". In our case it is only the CSS. What this mean is that, right now on our website, all the CSS need to be loaded and rendered from the files. When the user load the website the first view, that is also called the above-the-fold view, will be totally white without the features from the CSS. It cannot be rendered until it is loaded from the file. Usually this is something that is barely noticeable, especially on a desktop with a good network connection. But on a mobile phone that have a small connection the problem can occur. In order to fix this we have place the CSS code related to the above-the-fold designs directly in the header of the HTML file. Then the CSS can be rendered directly and will not have to wait for the full CSS file to be loaded. The rest of the CSS that is for the below the fold line can later be loaded and rendered the normal way. Doing this will give the first view of the site a faster and more optimized experience to the user. This is something that absolutely should be considered to apply to the website.

Using the developer’s tool in Google Chrome we can watch the network usage of the website, in both desktop size and scaled down to mobile size. On desktop size the website load is 3,2 MB including the video. On smaller devices the responsiveness of the website helps to decrease the size of it to around 198Kb, helping the website to load faster on the mobile devices immensely.

To measure the loading time of our website, we will use the two tools described earlier. Mobitest from Akamai, gives us the result of an average loading time of 1.98 seconds, with an average page size of 208,7kb. This was tested on a virtual iPhone5 using AT&T 3G network connection. In Xcode’s network link conditioner the downloading speed is simulated. The results of our website in this test is 0.642 seconds on a 40mpbs WiFi connection, 2.68 seconds on a 780kbps 3G connection, and 11.89 seconds on a 240kbps Edge GSM connection (also called 2G).
From the results of the loading time tests, we can see that our website managed to get in under the 4 seconds threshold stated in the introduction of this thesis. To load our website under the 4 second threshold was also the aim of this thesis. Even though the website could be optimized further, we could now consider our website to be well optimized for the mobile market.
Chapter 5

Concluding Remarks

This chapter contains the concluding discussion of the design theory and its implications. Interesting areas of future research will also be discussed.

5.1 Conclusion

We have developed a website using responsive web design and performance techniques. We have explored how to implement the responsive and performance constructs into the website, and analysed its implications. The exploration of creating the website is the first analytical evaluation of the design theory. The website design was able to adapt to the device span of mobile phones to desktop computers, and it was optimized over the span of devices. Delivering a heavy and more visual version for desktop computers, and a lighter version for mobile users, due to its often restricted bandwidth. The website was able to load under 4 seconds on mobile devices using a 3G network, as was the aim of the thesis.

5.2 Discussion

To make websites responsive and optimized for performance is arguably important in order to make a competitive web experience. There are a lot of benefits for companies, especially focused on e-commerce, if they would apply this approach. As described in the introduction of this thesis the users are keen to abandon websites that load under 4 seconds. Therefore, it is easy to understand that companies would lose revenue if their sites do not match these criteria. Now when surfing the web is more and more happening on mobile devices it is also very important to make the web experiences as good on them as on a desktop computer. Maintaining the website also becomes easier and cheaper since all content is in the same place, instead of spread out on different files for two versions of the website. This approach of making websites is something all would benefit from.
We would argue that this approach of developing websites is applicable for most web pages. The artefact’s mutability is big, and we believe it fits most kinds of web pages and web applications that can be accessed on desktop computers and mobile devices. As earlier discussed, it is also something that should be done for a more healthy and user friendly worldwide web.

Developing a website with this approach does take more effort in development. Compared to making a normal website there are more things to apply. This will increase the development cost of websites. But it’s important for companies to consider the benefits of putting the money into a responsive and optimized website. In the long-term the investment will be worth because of the benefits with such a website. A responsive and optimized website is much more compatible and sustainable on the market.

The website we developed is not as optimized, nor as responsive, as it could be. There are still more things to apply in order to improve its performance even further. But as we could see it was responsive and optimized enough to pass the tests. It was able to load under the 4 seconds aim even though it did not get full point in the PageSpeed Insight test. In the same test it got full point in the user experience. This shows that even though there is still more to do to improve the website, it is still enough to give a good result. This means that even though more effort needs to be put into the development of the website, it is not too much of an effort that is needed in order to pass these goals of performance.

Browser compatibility is important for a website in order to make it work for all users. It would be good to test the website on more browsers, such as Safari, Opera, Internet Explorer, etc. to see if our website would work well on all of them. The tests could also be extended to try the website on a bigger variety of mobile devices and test the website on Windows, Android and iOS devices. Then we would know if the website works well on all devices using different systems and browsers. It might be so that the code needs to be extended it order to make the design work properly in all browsers.

This design theory has mainly focused on making websites for mobile devices connected to 3G networks. In some countries 4G networks are being developed, increasing download speed further. At the same time new more powerful mobile devices are coming to the market. This means that websites can be loaded and rendered much faster. It also means that the approach of developing websites we have argued for might become somewhat unnecessary. But on the other hand for websites with an international target group it will take a long time until 4G connections will be reality. In fact in many countries, even 3G is not standard, and if it is, it does not cover all areas. People still have use 2G networks connections in many areas. It is also very common that people subscription for mobile data, and have to pay for every MB used, or having a limit of data that can be used. For them it is more expensive to use a non-optimized website. Therefore, it is still important to implement the responsive web design and performance techniques approach in website development. For large websites with a lot of content it is even more important to follow this approach, even though the fast progress of mobile hardware.

This design theory focused mostly on the front end development of the website. The focus on front end development was because of the assumed large implication on performance, described in the introduction. But back end and server side implication of the website performance would be interesting to take into account. They should not be forgotten and in order to have a well working website, all aspects of the system needs to work well. The chain, or the website, is not stronger than the weakest link.
5.3 Future Research

In a further development of this design theory testing the website more rigorously over more connections might be to consider. So far we have focused mostly on 3G connections for mobile devices. It would be interesting to see how the website would perform on a 4G connection and made further tests on 2G. We would then be able to make stronger arguments for the benefits of this website development approach. It would be interesting to extend the website with back end functionality in order to research more about the back end, and server side implications on the web performance on mobile devices.

In the ever growing market of devices that can be used to surf the web, it is clear that a responsive and optimized website is important in order to be sustainable and competitive. Therefore, it’s important with continued research and development of this approach. In fact, web performance is a pretty new subject, and a lot of scientific research is missing on it. The research is mostly driven by the industry, by such giants as Google, and Mozilla. We propose that more scientific research should be done on web performance in order to continue develop it and to give more diversity into its development.

Also, the industry in cooperation with World Wide Web Consortium (W3C) needs to make decisions and set standards for easier web development and management. The picture element that have been discussed in this thesis for example should soon enough be set as a standard, and go away from its stage of concept. There have also been discussions of using media queries for bandwidth. More effort needs to be but in to this by the organizations involved to make this happen. Browsers providers also need to update their browsers in order to keep them up to date with the standards. It will give the users a much better experiences, and it will facilitate for the web developers.


Peter McLachlan. Data uris are 6x slower than source linking, 2013. URL http://www.mobify.com/blog/data-uris-are-slow-on-mobile/.


Mark Sullivan. 3g/4g performance map: Data speeds for at and t, sprint, t-mobile, and verizon, 2012. URL http://www.techhive.com/article/254888/3g_4g_performance_map_data_speeds_for_atandt_sprint_t_mobile_and_verizon.html.


Appendix A

HTML Source Code

This is the source code from the HTML file of the website.

```html
<!DOCTYPE html>
<html class="no-js">
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
  <meta name="viewport" content="width=device-width,initial-scale=1"/>
  <title>Coffee Shop!</title>
  <link rel="shortcut icon" type="image/x-icon" href="static/img/favicon.ico">
  <link rel="stylesheet" href="static/css/style-0.1.0.minified.css">
  <link href='http://fonts.googleapis.com/css?family=Source+Sans+Pro:300,400,600,700,900' rel='stylesheet' type='text/css'>
  <link href='http://fonts.googleapis.com/css?family=Montserrat:400,700' rel='stylesheet' type='text/css'>
</head>
<body>
  <header class="main-header" role="banner">
    <div class="container">
      <h1 class="main-header--logo"><a href="#">My Coffee</a></h1>
      <a href="#" class="mobile-nav-trigger" id="mobile-nav-trigger"><img src="static/img/icons/menu.svg"></a>
      <nav class="main-navigation" role="navigation">
        <ul>
          <li><a href="#">Home</a></li>
          <li><a href="#about">About</a></li>
          <li><a href="#products">Our coffee</a></li>
          <li><a href="#contact">Contact</a></li>
        </ul>
      </nav>
    </div>
  </header>
</body>
</html>
```
<section class="masthead-masthead-video">
  <img src="static/img/medium2.jpg" class="masthead-video-media" data-teaser-source="static/video/royal-roll" />
  <h2 class="masthead-video-tagline">Fresh coffee of the highest quality</h2>
</section>
<main class="main-content" role="main">
  <div class="container section id="about">
    <h2>We love coffee!</h2>
    <p class="preamble">Consectetur adipisicing elit. Quis, nisi, porro quisquam tempora sequi dicta veritatis aperiam aut nesciunt necessitatibus totam consequatur quae rem magnam delectus ullam provident saepe architecto.</p>
  </div>
  <div class="article-list-section id="products">
    <div class="container">
      <article class="article-list--item-equalize">
        <h1>Dark roasted</h1>
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Quaerat, sit est possimus molestiae voluptatem adipisci.</p>
        <p class="button-container">
          <a href="#" class="button button-primary">Buy</a>
        </p>
      </article>
      <article class="article-list--item-equalize">
        <h1>Light roasted</h1>
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consectetur, cum. Ipsum dolor sit amet, consectetur adipisicing elit. At, autem?</p>
        <p class="button-container">
          <a href="#" class="button button-primary">Buy</a>
        </p>
      </article>
      <article class="article-list--item-equalize">
        <h1>Organic beans</h1>
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consectetur, ipsum dolor sit amet, consectetur adipisicing elit. Nisi, esse cupiditate repellendus obcaecati aliquam assumenda.</p>
        <p class="button-container">
          <a href="#" class="button button-primary">Buy</a>
        </p>
      </article>
    </div>
  </div>
</main>
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Neque aliquam provident similique iure dignissimos deserunt fugiat repellendus illum dolor?  

Section title

Section title

Section title

Section title

Section title

Section title

Section title

Section title
<li><h3>Our Service</h3>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Optio delectus eligendi similique cupiditate recusandae!
Minima. Lorem ipsum dolor sit amet.</p></li>

<li><h3>Subscribe</h3>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Optio delectus eligendi similique cupiditate recusandae!
Minima. Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Quasi, dignissimos, tenetur doloribus perspiciatis unde debitis?</p></li>

<li><h3>Coffee Ethics</h3>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Optio delectus eligendi similique cupiditate recusandae!
Minima. Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Totam, aut.</p></li>
Appendix B

CSS Source Code

This is a small part of the source code from the CSS file of the website. The code is not minified in order to make it easier to read. This code contains the CSS code for styling the articles of coffee beans, and the navigation bar.

```css
/* Articles */
.article-list {
    background: white;
    color: #4a4a4a;
}
.article-list--item {
    width: 30.66667%;
    margin-right: 4%;
    text-align: center;
    position: relative;
    padding-bottom: 4em;
}
.article-list--item:nth-child(3n) {
    margin-right: 0;
}
.article-list--item.third {
    margin-right: 0;
}
@media only screen and (max-width: 60em) {
    .article-list--item {
        width: 100%;
        margin-bottom: 4em;
    }
    .article-list--item:last-of-type {
        margin-bottom: 0;
    }
}
@media only screen and (max-width: 45.5em) {
}
```
.details-list .details-list--item {
    width: 100%;
    margin-bottom: 2em;
}

/*Navigation bar*/
.main-navigation {
    float: right;
}

.main-navigation ul {
    list-style: none;
    margin: 0;
    padding: 0;
}

@media only screen and (max-width: 43em) {
    .main-navigation {
        float: none;
        width: 100%;
        display: block;
        background: #222222;
        padding: 0;
        max-height: 0;
        overflow: hidden;
        -webkit-transition: max-height 0.2s;
        transition: max-height 0.2s;
    }
    .main-navigation.is-active {
        max-height: 600px;
    }
    .main-navigation li {
        display: inline-block;
        margin-right: 1.5em;
    }
    .main-navigation li:last-child {
        margin-right: 0;
    }
    .main-navigation li a {
        text-transform: uppercase;
        color: white;
        position: relative;
    }
}
Appendix C

JavaScript Source Code

This is a small part of the source code from the JavaScript file of the website containing the code for the "Picturefill" functionality. This code is originally made by Jehl (2014). The code is not minified in order to be easier to read.

```javascript
//PictureFill
(function( w ){
    "use strict";
    w.picturefill = function() {
        var ps = w.document.getElementsByTagName( "span" );

        // Loop the pictures
        for( var i = 0, il = ps.length; i < il; i++ ){
            if( ps[i].getAttribute( "data-picture" ) !== null ){
                var sources = ps[i].getElementsByTagName( "span" ),
                matches = []; // See if which sources match
                for( var j = 0, jl = sources.length; j < jl; j++ ){
                    var media = sources[j].getAttribute( "data-media" );
                    // if there’s no media specified, OR w.matchMedia is supported
                    if( !(media || ( w.matchMedia && w.matchMedia( media ).matches ))){
                        matches.push( sources[ j ] );
                    }
                }

                // Find any existing img element in the picture element
                var picImg = ps[i].getElementsByTagName( "img" )[ 0 ];
                if( matches.length ){
                    var matchedEl = matches.pop();
                    if( !picImg || picImg.parentNode.nodeName === "NOSSCRIPT" ){
```

46
picImg = w.document.createElement( "img" );
var alt = ps[i].getAttribute( "data-alt" );
if (alt !== null) {
    picImg.alt = alt;
}
else if( matchedEl === picImg.parentNode ){
    // Skip further actions if the correct image is already in place
    continue;
}
    picImg.src = matchedEl.getAttribute( "data-src" );
matchedEl.appendChild( picImg );
picImg.removeAttribute("width");
picImg.removeAttribute("height");
}
else if( picImg ){
    picImg.parentNode.removeChild( picImg );
}
}
}

// Run on resize and domready (w.load as a fallback)
if( w.addEventListener ){
    w.addEventListener( "resize", w.picturefill, false );
    w.addEventListener( "DOMContentLoaded", function(){
        w.picturefill();
        // Run once only
        w.removeEventListener( "load", w.picturefill, false );
    }, false );
}
else if( w.attachEvent ){
    w.attachEvent( "onload", w.picturefill );
}
}( this ));

// Match Media, a part of picturefill
window.matchMedia = window.matchMedia || ( function ( e, f ) { 
    var c, a = e.documentElement,
        b = a.firstElementChild || a.firstChild,
        d = e.createElement("body"),
        g = e.createElement("div");
    g.id = "mq-test-1";
g.style.cssText = "position:absolute;top:-100em";
d.appendChild(g);
    return function ( h ) { 
        g.innerHTML = '&#169;<style>media="'+h+'">#$mq-test-1.mediaWidth:42px</style>";' ;
    }
});
a.insertBefore(d, b);
c = g.offsetWidth == 42;
a.removeChild(d);

return {
    matches: c,
    media: h
}

}(document);
Appendix D

Screenshots of the website

Figure D.1: A Screenshot of how the website looks on a mobile sized screen. The design is continuously vertical.
Figure D.2: A Screenshot of how the website look on a desktop sized screen