Chapter 1 – Introduction

Can a web site be made responsive without having the user agent do all the work?

# Background

Modern web sites contain a multitude of functionality and rich, interactive content. With this rich content and interactivity there comes a cost in the form performance issues. Mobile devices are quickly becoming peoples main channel for accessing content on the web [[1](#_ENREF_1)], much of which is either not designed for or is poorly implemented for use on such devices. Web applications are often highly optimized on the back end, i.e. on Web servers, for the sake of scalability through database tuning, clustering, customized data caching and so on, which allows them to handle a large number of requests. Although this performance tuning helps the applications service a large number of users, the users themselves do not experience these optimizations in any tangible manner. Users are interested in their own request, and if it is slow, the quality of the user experience is severely diminished.

With mobile devices comes a severe reduction in the amount of screen space web developers can utilize to present content, along with reduced processing power and memory. Considering that web pages now need to work in both desktop and mobile contexts, developers have started coming up with ways to simplify and streamline the process of creating web pages that adapt to the environment in which they are being viewed. The most popular among these is “Responsive Web Design” (RWD) – suggested in the book by the same name [[2](#_ENREF_2)]. Roughly speaking it aims to make web sites “respond” to the context in which they are being viewed. This is primarily achieved through something called “Media Queries” in CSS. Media Queries can be used to detect certain attributes of the device rendering the web page, e.g. screen width and height, which then can alter the layout of the web page to fit the result of the query. It is not without its problems, though, and it has been noted that it is not a “silver bullet” for mobile Web design. Doing all of the adaptation on the frontend causes file sizes and business logic to grow much larger than previously, as all clients now receive all HTML markup, CSS, JavaScript and media [[3](#_ENREF_3)].

## Web performance

Steve Souders, author, creator of the web browser performance plugin YSlow and engineer at Google, suggests that because of this, we should focus on improving the response time on the front end, i.e. the web browser. The front end, he says, stands for 80-90 percent of the response time [[4](#_ENREF_4)]. He suggests a list of best practices aimed at improving the performance of web pages through front-end optimization:

I set out to capture these best practices in a simple list that is easy to remember. The list has evolved to contain the following 14 prioritized rules:

1. Make fewer HTTP requests
2. Use a content delivery network
3. Add an Expires header
4. Gzip components
5. Put style sheets at the top
6. Put scripts at the bottom
7. Avoid CSS expressions
8. Make JavaScript and CSS external
9. Reduce DNS lookups
10. Minify JavaScript
11. Avoid redirects
12. Remove duplicate scripts
13. Configure ETags
14. Make Ajax cacheable

- Steve Souders

Many design philosophies and best practices have surfaced with the advent of the mobile Web; RWD is, as mentioned, one of the most popular among these. While this method is practical for developers in terms of giving them an easy way to make their Web pages adapt to their environment, it leaves the whole job of making the Web page responsive to the front end. This is somewhat contradictory to Souders’ idea of optimizing the front end of Web applications, because that is where the largest chunk of the response time is spent in such a solution. RWD is reliant on heavy use of CSS and sometimes also needs to use JavaScript to hide and alter content to fit each device. Both of these things go against Souders’ principles. Doing these kinds of alterations on the front end also does not reduce the amount of HTTP requests, as these are defined in the HTML markup that is sent to the user from the back end.

On the opposite side of Souders we have people like Kate Matsudaira, a previous technical lead/manager at Amazon and Microsoft, who suggests that to improve the performance of the mobile Web we need to improve the back end [[5](#_ENREF_5)]. She says that because of the limited system resources and bandwidth, we need to minimize connections and data across the network, images and other media by leveraging technologies such as LocalStorage (an HTML 5 technology for storing data on the client) and caching, as well as allowing the server to correctly identify the limits of the device making a request. While her article focuses on API design for the mobile Web, it touches upon an interesting question: how can we improve the detection capabilities of the server to improve Web performance?

## Making the backend smarter

Jon Arne Sæterås, Mobile Web evangelist, blogger and product director at Mobiletech suggested in his post “Next steps of Responsive Web Design”, that letting the device do all the hard work of being responsive is neither fair, right or smart [[6](#_ENREF_6)].

It is not only the design of the web site and the layout of content that needs to be adapted of enhanced; the idea of being responsive, adaptive and enhancing, must be implemented in the whole value chain.

– Jon Arne Sæterås

In its current form, RWD sends the same markup, CSS, JS and images to all devices, regardless of their capabilities. What he means by his statement is that more parts of the Web hierarchy (server, CMS, editor etc.) must be able to respond to the capabilities of the device making the HTTP request. Being able to move much of the work over to the back end can potentially reduce much of the current load on devices viewing Web pages designed using RWD. Involving the whole “value chain”, as he calls it, will result in Web pages consuming less time and resources on the front end, leaving the responsibility of things like image scaling and markup processing to the back end.

Responsive Design + Server Side Components (RESS) is a method of making the server participate more actively in RWD. It suggests having Web pages split into components that are altered by the server depending on the type of user agent (UA) it detects as the requestor [[7](#_ENREF_7)]. This idea goes along with Sæterås’s notion of making more of the “value chain” smarter, in this case the Web server. It allows the server to more accurately tune what is sent to the client, amongst other things giving it the ability to optimize bandwidth usage when responding to mobile devices.

## Future Friendly

Being able to make a web site adapt to all present and future devices is a daunting proposition. Making it “future proof” is near impossible, as there is no way of telling what kind of devices people will be using to access the Web in the future. In 2011, several of today’s biggest Web gurus got together and proposed a new Web design philosophy: be Future Friendly, not future proof [[8](#_ENREF_8), [9](#_ENREF_9)]. It suggests that being “future proof” is unnecessary (and near impossible), and that focusing on what actually brings value to your site will help it survive longer. It further proposes focusing on content first, orbiting around the data and having dynamic feature detection systems that do not rely on manual updates. Through following these principles, they claim, Web pages can be made ready to function on future devices without developers having to constantly maintain and update their pages.

# Thesis description

In this thesis I present an implementation of a possible solution for making web pages responsive without having all of the work done on the front end, based on the RESS concept and inspired by a RESS-like system called “Detector” made by Dave Olsen [[10](#_ENREF_10)]. This is done with the aim of improving Web performance and the user experience on both desktop and mobile devices. The focus has been on the mobile aspect as this is where the biggest gains can be found in the context of improving performance, and thus the user experience. The implementation is a “server-side feature detection” plugin for the Enonic CMS, which aims to accurately detect the features supported by each individual UA. This is meant to allow the HTML document served to the user to be tailored on the back end before ever reaching the requesting UA. The idea is that by tailoring the HTML on the back end, the performance on the front end can be improved, following the concepts of Souders and Matsudaira, amongst others, and making the Web pages built in the CMS more “Future Friendly”.

## Structure

The thesis is structured in a “bottom up” fashion to provide all necessary background knowledge before delving into the details of the implementation. The point of this is to draw a clear picture of what I worked with and why I was doing it, as well as providing a historical context to the problems developers face today.

Chapter 1 is meant to provide a quick introduction to the problem we are discussing and what I have implemented.

Chapter 2 presents the technologies and concepts that this thesis is based upon, as well as the related work in this field.

Chapter 3 presents the Enonic CMS, on which I have developed plugin – how it works and how it supports plugins.

Chapter 4 discusses the details surrounding the implementation of the plugin, what choices I made during the development process and why, as well as how I worked and what problems I encountered.

Chapter 5 details how I did the performance testing of the plugin and also presents the results of the tests.

Chapter 6 discusses the merits of the plugin, the results of the performance tests and attempts to look at the results in the context of related work in the same field. Finally we summarize our key findings and discuss whether or not the implementation was successful.

The thesis can be read in three parts: the first part containing the background information needed to understand the implementation, the middle part can be considered the practical part, describing the process of implementing the system and the performance tests conducted on it to establish the impact it might have on making the Enonic CMS’ device classification system more Future Friendly. Finally, the last part is a discussion where we discuss the implementation, its merits and the results from the performance tests. It also looks at the concepts themselves and how they tie into the implementation.

# Bibliography

[1] M. Murphy and M. Meeker, "Top mobile internet trends," *KPCB Relationship Capital,* 2011.

[2] E. Marcotte, *Responsive Web Design*, 2011.

[3] J. Grigsby, "CSS Media Query for Mobile is Fool’s Gold," in *Cloud Four Blog* vol. 2012, ed, 2010.

[4] S. Souders, "High-performance web sites," *Commun. ACM,* vol. 51, pp. 36-41, 2008.

[5] K. Matsudaira, "Making the mobile web faster," *Commun. ACM,* vol. 56, pp. 56-61, 2013.

[6] J. A. Sæterås, "Next steps of Responsive Web Design," in *mpulp* vol. 2012, ed, 2011.

[7] L. Wroblewski, "RESS: Responsive Design + Server Side Components," in *LukeW* vol. 2012, ed, 2011.

[8] L. Wroblewski, "Future Friendly," in *LukeW* vol. 2013, ed, 2011.

[9] L. J. Wroblewski, Scott; Frost, Brad; Keith, Jeremy; Gardner, Lyza D.; Jehl, Scott; Rieger, Stephanie; Grigsby, Jason; Rieger, Bryan; Clark, Josh; Kadlec, Tim; Leroux, Brian; Trasatti, Andrea. (2013). *Future Friendly*. Available: <http://futurefriend.ly/>

[10] D. Olsen, "RESS, Server-Side Feature-Detection and the Evolution of Responsive Web Design," in *Mobile in Higher Ed* vol. 2012, ed, 2012.