Chapter 5 – Discussion

Throughout this thesis we have presented the theories and concepts that underlie the idea of the Detector plugin. In this chapter we will look at the merits of the implementation and discuss its advantages in the context of improving performance of Web sites, with focus on the future friendly Web[[1](#_ENREF_1)].

# Future Friendly Web

# The advantages of using RESS and Detector

The growing popularity of using mobile devices to browse the Web has lead to innovations in Web development such as RWD. This concept quickly became popular among Web developers, especially front-end developers, as it provided a simple way to do feature-detection on the front-end that made it easy to tailor the style of Web pages to the viewport of each individual browser. The concept does have its problems, though, which has been pointed out by several prominent Web developers[[2](#_ENREF_2), [3](#_ENREF_3)]. The crux of the concept is that it relies entirely on CSS, and specifically Media Queries, which are situated entirely on the front-end. It ends up having the same monolithic markup for every device and also leads to all CSS, scripts and media being sent to every device, as the server is completely oblivious to the capabilities of the requestor. This can lead to performance issues, especially when dealing with larger web pages, scripts and other media. While the layout is changed and elements may be hidden from users on mobile devices by using media queries, all the content defined in the page’s markup is still downloaded by the browser. Large images and scripts that may never be visible to the user will make a page load slower and spend more bandwidth than necessary. Considering how bandwidth is still at a premium in the mobile context, it’s unfortunate that this widely adopted development method can be detrimental to the user experience.

To reduce the amount of bandwidth used, lighten the workload on the front-end and improve the user-experience further, it is necessary to delegate at least some of the responsibility to the back-end. Sending the same resources to a mobile device and desktop is counter-productive to what RWD is trying to achieve, namely a future friendly Web experience. Making entirely separate templates for different form-factors is often called “Device Experiences” because it changes the browsing experience of the page depending on the device. This has been standard fare for many years, and does reduce bandwidth use by limiting the amount of data sent by having it explicitly designed for the requesting device class. These kinds of pages can be made semi-responsive by utilizing certain RWD techniques, such as fluid grids, to adapt to small changes in screen size. They are not responsive in the sense Ethan Marcotte meant when he introduced the concept of RWD, though, and because of their mostly static design; they are definitely not future friendly. Even though RWD has become extremely popular since its introduction in 2010[[4](#_ENREF_4)], the report that 82 percent of Alexa’s 100 top sites use server-side detection to tailor some amount of their content[[5](#_ENREF_5)] lends credence to the idea that using some kind of server-side detection is still useful. This is where the concept of RESS comes in.

As described in chapter 2, RESS suggests combining RWD with specific components of the markup rendered server-side to improve bandwidth usage, performance and user experience through optimizations done on the back-end. Luke Wroblewski, who came up with the concept, claims that it is effective amongst other things because it allows authors to only create one set of markup with components defined as templates, without having to worry about it working on different devices[[2](#_ENREF_2)]. This avoids splitting the code base, which would happen in the case of using Device Experiences. The theory surrounding the RESS concept is sound, and should definitely be taken seriously as a stepping-stone towards a solution to creating completely future friendly Web sites. While RESS is mostly a theoretical concept, Detector implements it practically. It also tackles a few of the problems mentioned by Wroblewski in his original article[[2](#_ENREF_2)], such as how to define device classes and improving upon the accuracy of device detection by combining server-side UA detection with client-side feature detection.

Detector improves server-side detection by including client-side feature detection when encountering unknown UA’s, in line with Alex Russel’s suggestion that feature tests only should be run in this case[[6](#_ENREF_6)]. The addition of these kinds of feature tests means that the server can be aware of exactly what capabilities the requesting UA has without any prior knowledge. It removes the need to maintain any kind of DDR, as the system itself is capable of figuring out the features of new UA’s dynamically. This is naturally limited by the feature tests that the system uses, but it is also obvious that updating the tests, which in this case is handled by Modernizr, is a lot less time consuming than updating a central DDR every time a UA is changed to support new features or a brand new type of UA is released. Considering this, the claim can be made that this kind of server-side detection is a leap forward in terms of creating a future friendly system which is not limited by the data stored in a DDR.

Detector also suggests a system for handling and easily defining device classes in what Olsen calls “browser families”, something that is only mentioned in passing in Wroblewski’s article, even though it is key to making a RESS system work. These families are what define the content the server renders in each individual component of a Web page using RESS. His method of defining families is both robust and user friendly both of which are important for a system to enjoy widespread use. It also makes it easy to define new families should the need arise, for example if a new kind of device class is introduced. The modular design of the browser family system allows it do be easily extensible, and thus future friendly by adapting alongside the client-side feature tests.

The concept of server-side feature detection, along with the browser family system allows Detector to adapt to changes and new UA’s without relying on DDR’s that need to be maintained. The components in RESS give Web pages built using it an inherently modular design, which allows them to be extended to support new families by creating a new family definition and component markup where needed. Olsen also mentions in his article that while these concepts come out of trying to solve mobile issues, they should not be pigeonholed as mobile solutions. RESS and Detector are not strictly mobile solutions and can help by offering a robust platform for building future friendly Web sites and apps, he believes. Considering the merits of these concepts, along with the server-side feature detection and browser family systems, Olsens’s claim does not seem too far-fetched.

# Choosing Enonic

# The plugin

*Why RESS and Detector?*

*What were Dave Olsen’s arguments compared to Wroblewski’s for the RESS concept?*

*What are the arguments against Detector, RESS and UA detection in general?*

*What are the arguments for them, in general?*

*Is the concept “future friendly”? Why?*

*Why Enonic?*

*Was there a notable performance impact from using the plugin?*

*Why and how is the plugin better than the native device classification in Enonic?*

*What are the drawbacks from using the plugin when it comes to developing web sites in Enonic?*

*What will it take to make it work with existing sites in Enonic?*

*What technical problems may show up when using the plugin “in the wild”?*

*How does the plugin tie into the concepts of “mobile first”, “responsive web design” and “device detection”?*

*What work is left when it comes to the plugin?*

*How can the Detector concept be altered to work independently of the frameworks, CMS’s and programming languages used on the backend of Web sites?*

# Bibliography

[1] 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/>

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

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

[4] E. Marcotte, "Responsive Web Design," in *A List Apart* vol. 2012, ed, 2010.

[5] R. Cremin, "Server-Side Mobile Web Detection Used by 82% of Alexa Top 100 Sites," in *CircleID* vol. 2013, ed, 2012.

[6] A. Russel, "Cutting The Interrogation Short," in *Infrequently Noted* vol. 2012, ed, 2011.