Responsive + Server Side, a multi-device targeting web application optimization strategy

by

Jóhann Lindell

LIU-IDA/LITH-EX-G--15/003--SE

2015-02-06
Examensarbete

Responsive + Server Side, a multi-device targeting web application optimization strategy

av

Jóhann Lindell

LIU-IDA/LITH-EX-G--15/003--SE

2015-02-06

Handledare: Fredrik Alserin, Sigma IT & Management
Examinator: Henrik Eriksson
Summary

The number of units with different screen sizes and resolutions accessing web applications is constantly growing. Responsive + Server Side (RESS) is a new approach to serve optimized content, functionalities and media files, using well-known techniques. Optimizing web applications could benefit both users and service providers, in terms of network traffic and power consumption. This thesis compares RESS with other common used solutions to target multiple devices in both performance and development/maintenance costs. Five different applications were built and two performance tests were performed. A conclusion that RESS is a good candidate for projects where performance optimization is important was made. RESS is also easy to add to existing web applications using responsive web design to optimize image and media files.
# Contents

Summary ................................. i

Contents ................................ ii

List of Figures .......................... iv

List of Tables ........................... v

List of Code Snippets .................. vi

Abbreviations .......................... vii

1 Introduction .......................... 1
  1.1 Motivation .......................... 1
  1.2 Purpose ............................ 1
  1.3 Thesis .............................. 2
  1.4 Limitations ......................... 2

2 Theory ............................... 3
  2.1 Responsive web design ............. 3
    2.1.1 Cornerstones of Responsive web design .......... 3
    2.1.2 Benefits with Responsive web design .......... 4
    2.1.3 Problems with Responsive web design .......... 4
  2.2 Server side adaptation ............ 5
    2.2.1 Cornerstones of Server side adaptation ........ 5
    2.2.2 Benefits with Server side adaptation ........ 5
    2.2.3 Problems with Server Side adaptation .......... 6
  2.3 Responsive + Server side .......... 6
    2.3.1 Cornerstones of Responsive + Server side .......... 6
    2.3.2 Benefits with RESS ....................... 7
    2.3.3 Problems with RESS ....................... 7
  2.4 WebPageTest ........................ 7

3 Method ............................... 8
  3.1 Implementation ...................... 8
    3.1.1 Third party libraries ......................... 8
    3.1.2 Server .................................. 9
    3.1.3 Responsive + Server Side ..................... 9
List of Figures

2.1 Example on how RESS works. The cookie can be created at different points depending on implementation. .......... 6

3.1 Server top level directories showing all applications built in this project . 9
3.2 Process of creating the 'RESS' cookie. ............................. 10
3.3 Each media file, images and videos, is reproduced in three different widths to be displayed on web browser with different widths. ............... 12

4.1 This is how the application looks like on a mobile device in the RESS and the m.SSA applications. From the left: start page with a list of Sigmas different companies, list of offices for a certain company, a list of employees at an office, contact information to an employee, and finally a video connected to the employee. The video page is on a separate view in the RESS and m.SSA applications, while the others show the video on the employee contact information view. ................................. 16
4.2 The four views as they are displayed in the RESS, RWD, RWD2RESS and SSA on a desktop with a width of 1,200 pixels. From the left, the first row shows a list of Sigma companies and a list of offices of a certain company. The second row shows a list of employees and contact information with the video for an employee. ................................. 16
4.3 The time it takes for each application to load and run all scripts on a first non-cached start page view and a cached repeat view on the same page. . 18
4.4 The amount of downloaded bytes for the start page first in a non-chached view and then followed by a cached repeat view. ................................. 19
4.5 The image download size for RESS, RWD and RWD2RESS on an emulated iPhone 6, and a desktop with two different browser window sizes, 700x400 and 1500x800 pixels. ................................. 20
4.6 The complete video download size for RESS, RWD and RWD2RESS on an emulated iPhone 6, and a desktop with two different browser window sizes, 700x400 and 1500x800 pixels. ................................. 20

C.1 General file structure for the applications implemented. The cookieHandler.js and the employee-video.php are specific for RESS and m.SSA. In some applications the directives and templates are HTML files. The underlying structure for images and videos differ between applications. .... 32
List of Tables

3.1 First page performance settings: iPhone 4 ................................. 13
3.2 First page performance settings: Desktop ............................... 14

4.1 Loading times for fully loaded start page .............................. 17
4.2 Downloaded bytes for a fully loaded start page .................... 18
4.3 Downloaded bytes for a fully loaded start page .................... 19
List of Code Snippets

2.1 A linked style sheet with a media query targeting all devices with a max-width of 500 pixels .................................................. 4
2.2 A media query in a CSS file using @import and @media with the same rules as in code snippet 2.1 ........................................... 4
3.1 The getDeviceFeature(...) function here used to fetch different data from server for desktop and mobile users. ........................................ 11
3.2 The code that determines which media file should be loaded. The variable $mediaPath is used in <img> and <video> source attributes. ................. 12
A.1 The JavaScript object cookieHandler. It is used for handling the RESS cookie. ................................................................. 28
B.1 The script that uses the JavaScript object cookieHandler to create the cookie in RESS and RWD2RESS. ................................. 31
Abbreviations

AJAX  Asynchronous JavaScript And XML
CSS  Cascading Style Sheet
CSS2  Cascading Style Sheet Level 2
CSS3  Cascading Style Sheet Level 3
DOM  Document Object Model
HTML  HyperText Markup Language
HTTP  HyperText Transfer Protocol
RESS  Responsive + Server Side
RTT  Round Trip Time
RWD  Responsive Web Design
SEO  Search Engine Optimization
SSA  Server Side Adaptation
URL  Uniform Resource Locator
W3C  World Wide Web Consortium
Chapter 1

Introduction

This report is the final thesis on bachelor level in computer science at the Department of Computer and Information Science at Linköping university, Sweden.

1.1 Motivation

Today the diversity of different devices, smart phones, tablets and digital book readers, is growing constantly. Each one comes with its own screen dimensions and screen resolutions. They provide different features, and new capabilities are added for every new generation. This snake’s nest has challenged web application developers to target as many gadgets as possible with an optimized user experience.

A survey from 2010 shows that a download time between one to five seconds made nearly a third of users to abandon the site, and more than half of the mobile users expect that their mobile would load as fast as their home computers [1]. For businesses depending on revenue from their website, loading time and user experience are important issues.

In 2004 W3C identified three different categories of approaches to target multiple types of devices: multiple authoring, single authoring, and flexible authoring [2]. Multiple authoring means that a different version of the content for each type of device, while single authoring means that there is a single version of the content. Flexible authoring is where the programmer has total freedom of combing the single and the multiple authoring approach. There are some commonly used techniques that uses these approaches, and they all have their advantages and disadvantages. Among the most popular today are Responsive web design (RWD)[3][4] and Server side adaptation (SSA)[4]. RWD is a single authoring technique, while SSA is a multiple authoring technique [2]. In September 11, 2011, Luke Wroblewski described an approach to multi-device web design in his blog [5]. He called this approach RESS, an abbreviation for Responsive + Server Side. It tries to combine the benefits of RWD and SSA.

1.2 Purpose

The objective of this thesis is to explore the RESS technique, and compare it to other techniques.
1.3 Thesis

To fulfill the purpose of this thesis, some key questions should be answered.

- What kind of projects are suitable for RESS?
- How is the workload to develop a RESS application in comparison with a RWD or a SSA application?
- How is the workload to maintain a RESS application in comparison with a RWD or a SSA application?
- Can a RESS implementation enhance the response time for mobile users?
- Can an existing RWD application be transformed into a RESS solution?

1.4 Limitations

This project is limited to evaluating the workload for developing a RESS web application compared to developing RWD and SSA, and possible benefits of this optimization technique.

Security and conflicting optimization techniques will be briefly discussed.
Chapter 2

Theory

This chapter describes the different web design techniques that will be examined alongside with their advantages and disadvantages.

2.1 Responsive web design

RWD adapts the content layout of a web page to the browser window; the web page layout responds to its context. RWD is prepared for device generations to come, unless there is a paradigm shift in the technology and standards. The expression “responsive web design” was first coined by Ethan Marcotte in an article in A List Apart, in May 2010 [6]. In this article, and a book that followed up the article, he presents how to combine three existing technologies to create web applications for an optimal viewing experience in multiple devices: flexible grids, flexible images and media, and media queries [3].

2.1.1 Cornerstones of Responsive web design

The term fluid grid is an approach where positioning of DOM elements and other layout is done with CSS. Fluid means that relative sizing is used, e.g. text size is expressed in ems, and DOM elements width and height are expressed in percentage instead of pixels. With relative sizing, E. Marcotte made a layout based on a grid system expressed in percentage, which can be used by designers to get a coherent design on different sized client screens. Ems are based on the base font size. This can be expressed in a formula:

\[
displayed\text{FontSize} = base\text{FontSize} \times ems
\]

Since one pixel can be different size depending context, ems are to prefer when developing a multi-device application.[3]

E. Marcotte used the term flexible images and media for relatively sizing images and other media files, using percentage instead of pixels. The CSS rule max-width or width applies to most fixed-widths elements, e.g. <img>, <video>, <embed> and <object>, and is very useful for front-end layout for these elements.[3]

Media queries has been a part of the standard since CSS2. They are used to gain information about client device features and type. Since CSS2.1 there are nine different media types that are recognized: all, braille, embossed, handheld, print, projection, screen, tty and tv [7]. Media features became a W3C recommendation in 2012 with CSS3. The current recommended features are width, height, device-width, device-height, orientation, aspect-ratio,
device-aspect-ratio, color, color-index, monochrome, resolution, scan and grid [8]. Probably is width the most common feature used in RWD. A media query can be written in both HTML, as a query in a linked style sheets media attribute, see code snippet 2.1, and in css, either using the @import rule or by putting media queries directly in a style sheet, see code snippet 2.2. A media query consists of a media type and can include logical expressions that check for the conditions of particular media features.

```
<link rel="stylesheet" href="mobile.css" media="all and (max-width:500px)">
```

**Code Snippet 2.1:** A linked style sheet with a media query targeting all devices with a max-width of 500 pixels

```
@import url("mobile.css") all and (max-width:500px);
@media all and (max-width: 500px) {...}
```

**Code Snippet 2.2:** A media query in a CSS file using @import and @media with the same rules as in code snippet 2.1

There is also a draft for an API for media query listeners in JavaScript. It is called the “MediaQueryList Interface” [9]. It is a way to respond to changes through scripts. This feature is still in an experimental stage, and Firefox 6.0 is the only browser that supports the “MediaQuery Interface” at time of writing.

### 2.1.2 Benefits with Responsive web design

In an article presented at the 36th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). Opatija, Croatia, S. Mohorovičić presented some benefits and downsides with the RWD approach. He states that implementations of RWD are “one, content focused, device-independent website, long-term money and time savings, easy maintenance, better Search Engine Optimization (SEO) managing, more consistent user experience and usability”.[10]

One big advantage of RWD is that it uses only one code base and one URL. Developers do not have to create individual systems for each device targeted. This means that updates and improvements only have to be implemented in one place, reducing risk for errors and the need for increased number of tests to be run. A single URL also makes sharing a link to others, e.g. in social networks, easier, since the site is implemented for all devices[10].

Since the interface resizes automatically with the browser window, meaning that desktop users also gets a good content presentation if the web browser window changes size or dimension; no panning will be needed. With RWD, mobile users do not have to pin, pan and zoom to navigate on a site. This makes RWD very good for content consumption.

### 2.1.3 Problems with Responsive web design

Though the most commonly used web browsers support CSS3 media queries in their most recent versions, clients with older versions will not be able to parse the media queries correctly.[11]

It seems as developers use RWD mainly for layout issues, and not really deal with performance optimization. There are indications that most RWD sites do not offer page
size optimization for different screen resolutions and sizes. In March 2012 and 2013 Guy Podjarny conducted a simple experiment by running 347 responsive web sites in four different window sizes on webpagetest.com. He found that “the majority (72%) were roughly the same size on the smallest and biggest screens” [12]. He found three reasons for this: download and hide, download and shrink, and excess DOM[13]. In short they are different sides of the same coin. Responsive sites download the same files no matter which device is being used. To be able to serve a full-fledged experience on a desktop, there will be excess DOM elements in the HTML for a mobile user, that will be hidden for a mobile user. Fluid images and videos are downloaded in desktop version to all devices in a resolution that cannot be appreciated by a mobile user, which adds to loading time.

If the implementation of RWD is not well designed, a RWD site with a lot of content will be very long. It will be hard to overlook on a mobile device with a small screen, and will take a lot of scrolling to view all information. This is pointed out by B. Frost when he compares B. Obamas and M. Romneys official presidential campaign sites from 2012 [14].

2.2 Server side adaptation

In the early 2000’s, mobile telephones often had a memory limit of 10kB for a HTML page. Therefore less text, and in fact no images could be displayed, and an alternative way of showing web pages were crucial to serve early mobile phone web surfers. The need for device detection was foreseen by the designers of HTTP. Already in 1996 the user-agent header was introduced [15], and this was then the only option for web application developers to adapt content to their mobile users. This has proven to be a successful, and widely implemented solution. As an example big players as youtube.com, run by Google, redirects directly to m.youtube.com, and facebook.com redirects to m.facebook.com, both delivering different files to the client [16][17].

2.2.1 Cornerstones of Server side adaptation

The key for SSA is the user-agent header sent with the HTTP request. A common technique is that the user-agent header is analyzed, and the user is redirected to a URL, often called “m-dot sites”, which will serve files that are optimized for current device. It could be worth mentioning, that using RWD is fully possible within SSA applications.

2.2.2 Benefits with Server side adaptation

One significant benefit with SSA is that full adaptation is possible. You can have very fine-grained control over content and features served to devices of different type or size. Developers can optimize size of larger media files, as well as only serving necessary HTML, CSS and JavaScript, thus removing excess code and DOM elements, see 2.1.3 above. Mitt Romneys official website for the presidential election 2012 used SSA to redirect mobile users to a dedicated mobile site [14]. Compared with Obamas corresponding site, which used RWD, the load time was cut in more than half. With SSA, developers have the opportunity to both optimize user experience and network performance.

The user experience can be more positive with a mobile optimized site. The Health Science Library at the University of Colorado Anschutz Medical Campus did a usability
study after implementing a mobile version of its website. They concluded that the optimized mobile site improved information retrieval effectiveness and efficiency, and found indications of a more satisfying user experience.[18]

2.2.3 Problems with Server Side adaptation

Among the disadvantages when using the redirection model is the need for maintenance of more than one code base, the number depending on how many different devices will be targeted. It will also require an updated device library to be accurate in the analyze of the User-Agent string. In its purest form it does not provide real-time data to help serve tailored data, e.g. using GPS-coordinates for getting local information.

Since a SSA solution has at least two URLs, sharing in social media could be an issue. If the device detection and redirection does not go deep enough in the application, links shared by mobile users will send desktop users to the mobile version and vice versa. [14]

2.3 Responsive + Server side

Responsive + Server side (RESS) is a fairly new and untested way of creating a web application that targets multiple devices. It is an optimization technique, which tries to eliminate the drawbacks that come with RWD and SSA. L. Wroblewski ideas are not to be confused with “Hybrid approach” [19]. While Hybrid Approach is more of a holistic approach[20], RESS uses one set of page templates and can be implemented to different degrees. Regarding the page templates L. Wroblewski makes exception for key elements like page header and footer[5]. This suggests that RESS could be implemented both single and multiple authoring. RESS has been gaining popularity in the community recently. For example, the University of Notre Dame launched their new website, https://www.nd.edu, 1 April 2012, using the RESS approach[21].

2.3.1 Cornerstones of Responsive + Server side

RESS relies on a handful of well known techniques. It uses RWD, device detection, HTTP cookies, and client feature detection.

For layout purposes flexible images, fluid grids and media queries are used to take advantage of what RWD does best.

Device detection is based on analyzing the User-Agent string that is a part of the HTTP header in an URL request. It could either be a simple parsing of the User-Agent
string to tell if it is mobile, or known features for the client device can be fetched from a device detection database.

**HTTP cookies** are used for storing state information, to overcome that HTTP is stateless. It is an easy way of sharing data between client and server. The use of HTTP cookies in RESS was suggested by A. Andersen in a RESS tutorial[22].

**Client feature detection** is a direct result of A. Andersen’s tutorial[22] use of cookies. The client can fill in blanks from the device detection with JavaScript, e.g. browser window width and height can not be known by the device detection database.

### 2.3.2 Benefits with RESS

The benefits you get from RESS are a mixture of what is presented by RWD and SSA. As mentioned in section 2.3.1, RESS includes RWD, so automatically all benefits of RWD are present. A single URL is easier to share in social media. The user experience can be improved in two ways: data optimization and functionality optimization. Optimizing data would be serving only the necessary HTML, CSS and JavaScript to the client. Optimized functionality could be a different behavior and added or removed functions.

### 2.3.3 Problems with RESS

There can be cases where device detection returns a false positive, or a device is not possible to identify[23]. New devices are continuously being introduced to the market, so the device detection database has to be up to date.

RESS could interfere with caching, another performance optimization, if the page is cacheable. Caching services relies mainly on the uniqueness of the URL. G. Podjarny addresses this issue in his book “Responsive and fast”. He breaks it down to two parts (even though the caching process is more complex than that): first-party cache and third-party cache. First-party caches are the ones within a websites control, and third-party caches outside a websites control. G. Podjarny also presents some tools and techniques for avoiding optimization conflicts between RESS and caching.[23]

The use of cookies could pose a security risk. It is well-known that cross-site scripting, a form of code-injection attacks, where the cookie information can leak to the attacker. There are some security mechanisms that can be used to increase cookie security. The flag `HttpOnly` blocks all attempt to access the cookie from non-HTTP APIs. The `Secure` flag makes sure that the cookie only is sent over HTTPS connections [24]. There have been no formal results that proves the security that these techniques provide [25].

### 2.4 WebPageTest

WebPageTest is an open source project under the BSD license. It is developed and supported by Google, but was originally developed by AOL. It is a web application performance tool, that is available online. It is periodically available for download on github, to be run on a local machine. Companies and individuals provide testing infrastructure such as different hardware and network connections for the web service. [26]

When testing a web page on webpagetest.org you enter the url you want to test, which test location to use and which browser and operating system to run the test. There are also more advanced options, e.g. connection speed and round trip time, number of tests to run, and if you want to capture the process on video. There are also possibilities to add custom scripts and block scripts. [27]
Chapter 3

Method

In this chapter the implementations of the different applications are described. This is followed by the tests that were run to measure the performance of the different applications.

3.1 Implementation

This section begins with a brief introduction to third party libraries that were used, and why they were chosen. It is followed by descriptions on how the web applications were implemented, and in what environment they are running. First the RESS application was created, and a main part of the work produced in that process could be used in the other applications. Other applications built were a RWD, SSA along with a m.SSA, and finally the RWD was transformed into a RESS, called RWD2RESS. The file structure in the applications is shown in appendix C.

3.1.1 Third party libraries

All applications that are built use CodeIgniter, Bootstrap and AngularJS. The RESS and the RWD2RESS applications also use 51Degrees and Modernizr for device and feature detection.

**AngularJS** *(version 1.3.0-rc.3)* is an open-source web application client framework, mainly for creating single-page applications. It applies a MVC pattern, with HTML as template language allowing developers to extend standard HTML syntax. It is back-end independent, since all code is run in the web browser. See [https://docs.angularjs.org/](https://docs.angularjs.org/) for more information. It was chosen out of curiosity to learn a new framework.

**Bootstrap** *(version 3.2.0)* is a front-end-framework, built by Twitter. It is designed to help build user interface elements with help of CSS3. It also has built-in support for RWD, which can save substantial development time when implementing a RWD application. Bootstrap has more advanced features, but in this project Bootstrap was chosen to help implement RWD features. For more information, see [http://getbootstrap.com/](http://getbootstrap.com/).
Modernizr (version 2.8.3) is a feature detection JavaScript library. It runs on page load and creates a JavaScript object with the result. Modernizr has also more advanced features, e.g. some CSS interaction, but in this project it is only used for adding information to the RESS cookie, see 3.1.3.1. For more information, see http://modernizr.com/. Modernizr was chosen since it was suggested by A. Andersens RESS tutorial [22].

51Degrees For device detection 51degrees is used server side. The free version of 51degrees provides 40 features for 30 000 device combinations, and has an API for CodeIgniter. For more information, see http://51degrees.com/. 51Degrees was chosen since it had an open source version, and presented an easily configured plugin to CodeIgniter, the back-end that was used.

CodeIgniter (version 2.2.0) is used as back-end system, and is shared between all systems. It is a PHP based CMS, built upon a MVC pattern, though the views are handled by angular, see above. Among its advantages is that it is small, and comes with some built-in features for cookie handling. It is fairly easy to extend with own and built-in libraries, helpers, hooks and classes. For more information see https://ellislab.com/codeigniter. Because of previous experience working with CodeIgniter, it was chosen to save time.

3.1.2 Server

All solutions are located at the same server, see figure 3.1 for the top folder structure. The server is hosted by one.com and is located in Denmark. It is a Linux web5 3.14.18-20140908-0724-6aa2a98 with Apache version 1.3. All data is stored in a MySQL server, Client API version mysqli 5.0.11-dev - 20120503.

3.1.3 Responsive + Server Side

The RESS project is not a RESS application as described by L. Wroblewski. The templates are divided in two parts, one for mobile and one for desktop devices, giving them different design and functionality.

3.1.3.1 Creating the cookie

To be able to implement the RESS features a cookie is created with known and useful features. The process can be divided into nine steps, described below and illustrated by figure 3.2. 51Degrees supply an API for CodeIgniter, but for the client a special cookieHandler object has been created, see appendix A.1.

1. Client makes a request for a page in the RESS application. The index.php file checks if there is a RESS cookie. If the cookie exists the angular application is loaded (step 9).
2. If there is no RESS cookie, a loading page and script for creating a cookie is served to client.

3. The script makes a synchronous XMLHttpRequest, get_cookie_data(), to the server to get device features.

4. The server asks 51degrees what device features the client has, based on the user-agent string.

5. 51degrees responds with a mapped object, with the features as keys. An extra custom feature is added. It is a string telling what device type it is, and it is used for finding paths to script, styling and media files.

6. The client gets a response with the 51degrees object as a json encoded string.

7. A cookie is created on the client with the response as value. Then, the features in the cookie marked ‘Unknown’ are updated, with help from the Modernizr framework. Here are also some additional features added to the cookie.

8. Then a reload is made, but now with a valid cookie. Basically going to step 9 through step 1.

9. When there is a valid HTTP cookie, the angular application is served to client.

Figure 3.2: Process of creating the 'RESS' cookie.
3.1.3.2 AngularJS application

The separation of desktop and mobile happens in both server and on the client. All view markup is written in php files, and mobile and desktop clients are differentiated at the request for the angular application on the server. The php and html mixed template files use the 'deviceType' cookie attribute to load the correct HTML. The JavaScript cookieHandler object (see appendix A.1) has a function, getDeviceFeature(feature, cookieName). It is used for different database queries, where mobile clients ask for less information, see line 3 and 8 in code snippet 3.1.

```javascript
SCCtrl.controller('sigmaConnectCtrl', function ($scope, $http) {
  if (cookieHandler.getDeviceFeature('deviceType') == 'mobile') {
    $http.get('/CodeIgniter/index.php/home/get_branches_name_id',
      {cache: true}).
      success(function(data) {...}).
      error(function(){...});
  } else {
    $http.get('/CodeIgniter/index.php/home/get_branches_info',
      {cache: true}).
      success(function(data) {...}).
      error(function(){...});
  }
});
```

Code Snippet 3.1: The getDeviceFeature(...) function here used to fetch different data from server for desktop and mobile users.

3.1.3.3 The CSS

Along with the CSS that comes with bootstrap, there are two customized CSS files; one for mobile and one for computers. Here the 'deviceType' attribute in the RESS cookie is used to determine which one to use. The version for computers uses media queries. There are two custom breakpoints in this solution. One following the bootstrap convention at 768 pixels, and one at 1020 pixels for layout purposes in the employee view.

3.1.4 Responsive Web Design

The application implemented as a responsive application is basically the desktop version of the RESS solution. It uses similar CSS and the angular application only differs in two regards; there is no validation if the device is mobile or not, and therefore the files containing the markup does not need PHP rendering first.

3.1.4.1 AngularJS application

In the RWD application the AngularJS application is simpler than in the RESS solution (see 3.1.3.2). It consists of HTML files. It uses one view less, since the video is presented on the page with the employee. Much of the markup and logic from the RESS application could be used. All logic concerning the HTTP cookie is removed.

3.1.4.2 The CSS

For the RWD application, the CSS for computers, see section 3.1.3.3, could be used.
3.1.5 RWD to RESS

For turning a RWD application into a RESS application, first the RWD application, see section 3.1.4, was cloned. The cookieHandler, see appendix A, was added as a script in the angular application index file, along with a XMLHttpRequest, get_cookie_data, see appendix B). It is basically the same process as in figure 3.2, but the reload is removed. Here the application is loaded directly like in RWD. All images were edited into different sizes, and organized in three folders, see figure 3.3.

The breakpoints for media files were here set following the video formats in the application. In that way the same path variable can be used to locate all media files. In template after template which include an image or media file, the same code snippet, see code snippet 3.2, was added. Then three different sized images were added to the corresponding folder in figure 3.3. These template files were also changed from HTML to PHP files.

```
<?php
$mediaPath = 720; //default imagePath if no cookie is set
if(isset($_COOKIE['RESS'])){  
  $width = json_decode($_COOKIE['RESS'])->width;  
  if($width > 720) 
    $mediaPath = 1080;  
  elseif($width < 480) 
    $mediaPath = 480; 
}>
```

**Code Snippet 3.2:** The code that determines which media file should be loaded.

The variable `$mediaPath` is used in `<img>` and `<video>` source attributes.

3.1.6 Server Side Adaptation

The SSA consists of two applications, one for desktops called SSA, and one for mobiles called m.SSA.

3.1.6.1 SSA

The desktop version for SSA is the same application as RWD solutions (section 3.1.4), except the slight difference that it redirects to m.SSA if it detects a mobile client. For detecting if it is a mobile unit 51Degrees is used.
3.1.6.2 m.SSA

The mobile SSA application is the mobile version for the RESS application (section 3.1.3). The biggest differences are that no HTTP cookie is created and therefore the JavaScript object cookieHandler is not loaded. The same content is loaded to all device who send a request to this URL.

3.2 Performance

Two different tests have been conducted. First the performance of the start page has been measured. Focus is set on how long time there is activity, i.e. download and script activity, and how many bytes has been downloaded to the client when the page is fully loaded. In the second test, every template in the application is visited, to sum the total amount of bytes downloaded to client. The first test was run on all five applications, while the second test was run on RESS, RWD and RWD2RESS.

3.2.1 First page performance

This tests were to measure the performance data of the different applications. To gather data webpagetest.org was used. For each application two tests were performed, one as an iPhone 4 with iOS 5.1, and one as a desktop with Chrome browser (see table 3.1 and 3.2 for settings). There is a significant difference between the internet connections; the iPhone has a shaped 3G connection with 1.6 Mbps/768 Kbps and a 300ms RTT, while the desktop uses a fiber-optic connection with 20/5 Mbps and a 4ms RTT. Each test was run 9 times, both with a non-cached first view followed by a cached repeat view, where the browser is closed in between runs. The tests for RESS and RWD2RESS had to be repeated a few days later due to a bug in the cookieHandler.js.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test location</td>
<td>Dulles, VA USA (IE8-11, Chrome, Firefox, Android, iOS</td>
</tr>
<tr>
<td>Browser</td>
<td>iPhone 4 iOS 5.1</td>
</tr>
<tr>
<td>Connection</td>
<td>Shaped 3G (1.6 Mbps/768 Kbps, 300ms RTT)</td>
</tr>
<tr>
<td>Number of Tests to Run</td>
<td>9</td>
</tr>
<tr>
<td>Repeat View</td>
<td>✓ First View and Repeat View □ First view</td>
</tr>
</tbody>
</table>

Table 3.1: First page performance settings: iPhone 4

All test were run nine times with a first view and a cached repeat view
Table 3.2: First page performance settings: Desktop

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test location</td>
<td>Dulles, VA USA (IE8-11, Chrome, Firefox, Android, iOS</td>
</tr>
<tr>
<td>Browser</td>
<td>Chrome</td>
</tr>
<tr>
<td>Connection</td>
<td>FIOS (20/5 Mbps 4ms RTT)</td>
</tr>
<tr>
<td>Number of Tests to Run</td>
<td>9</td>
</tr>
<tr>
<td>Repeat View</td>
<td>☑ First View and Repeat View □ First view</td>
</tr>
</tbody>
</table>

Set up for webpagetest.org for desktop.

All test were run nine times with a first view and a cached repeat view.

### 3.2.2 Total application download

In this test all templates in the RWD, the RESS and the RWD2RESS applications were tested. Cookies and cache was cleared in between each visit. To gather data Google Chrome built-in developer tools were used. All applications included in the test were visited with as a desktop with screen dimensions 700x400 and 1500x800 pixels, and as an iPhone 6, using the emulation tool in Google Chrome. All test were run once and performed on a local version of the web applications on the computer performing the tests. The tests were performed in the following steps:

1. Clear browser history. Delete cache and cookies
2. Open application start page
3. Click on button ‘Sigma IT & Management’
4. Click on button ‘Växjö’
5. Click on ‘R2D2’
6. Click on ‘Click to watch R2s video’ (only for RESS application)
7. Click on video play button, wait until video has finished
Chapter 4

Result

This project resulted in five different applications that present similar data to the user. Different aspects of performance was measured. The results from the implementation and from the performance tests are presented in this chapter.

4.1 Implementation

The implementation resulted in five different applications, presenting the same functionalities in different ways to the user. All applications use the same CodeIgniter back-end, see figure 3.1, and the same database to retrieve data.

The basic idea for the application is to find contact information for employees in a company or organization. Following steps have to be run to get the information:

- Front page displays a list of Sigma companies
- User clicks/taps on a company and a list of offices is displayed
- User clicks/taps on an office and a list of employees is displayed
- User clicks on an employee and contact information is displayed

In terms of content you can divide the applications in two groups: a general version which serves the same content to all devices, and a mobile version. The mobile versions are a RESS application served to a mobile and m.SSA. For the RESS mobile and m.SSA a decision to not serve any images except for the logotype in the navigation bar was taken, as shown in figure 4.1. Also the video that every employee shows is presented as a link in the contact page in RESS mobile and m.SSA, and the actual video in a separate view, when the video is shown in the contact page in the other applications, see figure 4.2. These decisions were made to keep these versions light in content, and also separate these from the RWD applications in functionality. This also gave two RESS implementations with two different design approaches: RESS uses the multiple authoring approach and RWD2RESS uses single authoring [2].

The separation of the mobile version in RESS is done in the templates, with an if statement. That was done to keep a clean separation between the two versions so the HTML would be kept clean and easy to read and not cluttered with too many if statements. The possibility to have a mobile and a desktop template was not chosen since an uniformity in the file structure was desired.
Figure 4.1: This is how the application looks like on a mobile device in the RESS and the m.SSA applications. From the left: start page with a list of Sigmas different companies, list of offices for a certain company, a list of employees at an office, contact information to an employee, and finally a video connected to the employee. The video page is on a separate view in the RESS and m.SSA applications, while the others show the video on the employee contact information view.

Figure 4.2: The four views as they are displayed in the RESS, RWD, RWD2RESS and SSA on a desktop with a width of 1,200 pixels. From the left, the first row shows a list of Sigma companies and a list of offices of a certain company. The second row shows a list of employees and contact information with the video for an employee.

Since RESS is very different in nature in its mobile and general version, a loading
page is shown while the cookie is created, since the content of the cookie is used for determining what content to serve. The loading page is only shown when there is no cookie present. The cookie is created on the client with a synchronous XMLHttpRequest to the server. It had to be synchronous since the cookie is manipulated in two steps; first the cookie data is fetched and the cookie created, and then the cookie is updated. The update can not happen until the cookie is created.

## 4.2 Performance

Below results from testing are presented. The raw results of the measurements can be found online, see appendix D for the URLs.

### 4.2.1 First page performance

Table 4.1 shows the average activity time and the median for all applications, both for first view and the repeat view. Activity time is measured from first initial request sent until no network or scripting activity is ongoing in the web browser. The results are also shown as a graph in figure 4.3. Table 4.3 shows the result from the measurements of downloaded bytes in average and median, both for first view and repeat view. Results from table 4.3 are shown as a graph in figure 4.4.

<table>
<thead>
<tr>
<th>Application and device</th>
<th>First view (s)</th>
<th>Repeat view (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\bar{x}$</td>
<td>$\tilde{x}$</td>
</tr>
<tr>
<td>RESS PC</td>
<td>3.053</td>
<td>3.305</td>
</tr>
<tr>
<td>RESS iPhone</td>
<td>5.808</td>
<td>5.348</td>
</tr>
<tr>
<td>RWD PC</td>
<td>2.614</td>
<td>1.901</td>
</tr>
<tr>
<td>RWD iPhone</td>
<td>10.397</td>
<td>10.337</td>
</tr>
<tr>
<td>RWD2RESS PC</td>
<td>2.820</td>
<td>3.030</td>
</tr>
<tr>
<td>RWD2RESS iPhone</td>
<td>13.087</td>
<td>12.447</td>
</tr>
<tr>
<td>SSA PC</td>
<td>2.073</td>
<td>1.930</td>
</tr>
<tr>
<td>SSA iPhone</td>
<td>11.979</td>
<td>12.595</td>
</tr>
<tr>
<td>m.SSA PC</td>
<td>1.455</td>
<td>1.433</td>
</tr>
<tr>
<td>m.SSA iPhone</td>
<td>6.351</td>
<td>6.213</td>
</tr>
</tbody>
</table>

Average ($\bar{x}$) and median($\tilde{x}$) loading time, and standard deviation ($\sigma$) for the starting page of each application for a non-cached first view and a cached repeat view.
Figure 4.3: The time it takes for each application to load and run all scripts on a first non-cached start page view and a cached repeat view on the same page.

Table 4.2: Downloaded bytes for a fully loaded start page

<table>
<thead>
<tr>
<th>Application and device</th>
<th>First view (KB)</th>
<th>Repeat view (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\bar{x}$</td>
<td>$\tilde{x}$</td>
</tr>
<tr>
<td>RESS PC</td>
<td>536</td>
<td>546</td>
</tr>
<tr>
<td>RESS iPhone</td>
<td>323</td>
<td>322</td>
</tr>
<tr>
<td>RWD PC</td>
<td>512</td>
<td>539</td>
</tr>
<tr>
<td>RWD iPhone</td>
<td>417</td>
<td>417</td>
</tr>
<tr>
<td>RWD2RESS PC</td>
<td>560</td>
<td>560</td>
</tr>
<tr>
<td>RWD2RESS iPhone</td>
<td>395</td>
<td>397</td>
</tr>
<tr>
<td>SSA PC</td>
<td>499</td>
<td>517</td>
</tr>
<tr>
<td>SSA iPhone</td>
<td>348</td>
<td>348</td>
</tr>
<tr>
<td>m.SSA PC</td>
<td>393</td>
<td>410</td>
</tr>
<tr>
<td>m.SSA iPhone</td>
<td>312</td>
<td>312</td>
</tr>
</tbody>
</table>

Average ($\bar{x}$) and median ($\tilde{x}$) downloaded bytes, and standard deviation ($\sigma$) for the starting page of each application for a non-cached first view and a cached repeat view.
Figure 4.4: The amount of downloaded bytes for the start page first in a non-chached view and then followed by a cached repeat view.

4.2.2 Total application download

The results of running the total application and viewing the entire video is divided in two parts. In figure 4.5 the total downloaded bytes in images is presented. In figure 4.6 the total downloaded bytes in videos is presented.

Table 4.3: Downloaded bytes for a fully loaded start page

<table>
<thead>
<tr>
<th>Application and device</th>
<th>JavaScript (KB)</th>
<th>Images (KB)</th>
<th>Video (KB)</th>
<th>Total (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESS iPhone 6</td>
<td>274</td>
<td>13.6</td>
<td>238,000</td>
<td>238,288</td>
</tr>
<tr>
<td>RESS PC (700x400)</td>
<td>274</td>
<td>1,100</td>
<td>397,000</td>
<td>398,374</td>
</tr>
<tr>
<td>RESS PC (1500x800)</td>
<td>275</td>
<td>1,100</td>
<td>692,000</td>
<td>693,375</td>
</tr>
<tr>
<td>RWD iPhone 6</td>
<td>264</td>
<td>1,100</td>
<td>692,000</td>
<td>693,364</td>
</tr>
<tr>
<td>RWD PC (700x400)</td>
<td>264</td>
<td>1,100</td>
<td>692,000</td>
<td>693,364</td>
</tr>
<tr>
<td>RWD PC (1500x800)</td>
<td>264</td>
<td>1,100</td>
<td>692,000</td>
<td>693,364</td>
</tr>
<tr>
<td>RWD2RESS iPhone 6</td>
<td>282</td>
<td>390</td>
<td>238,000</td>
<td>238,672</td>
</tr>
<tr>
<td>RWD2RESS PC</td>
<td>282</td>
<td>728</td>
<td>397,000</td>
<td>398,010</td>
</tr>
<tr>
<td>RWD2RESS iPhone</td>
<td>282</td>
<td>863</td>
<td>867,000</td>
<td>868,145</td>
</tr>
</tbody>
</table>

Download size in KB for images and video in a total application view of three applications, run in three different contexts: iPhone 6 emulated in Google Chrome, and Chrome browser with two window settings, 700x400 and 1500x800.
Figure 4.5: The image download size for RESS, RWD and RWD2RESS on an emulated iPhone 6, and a desktop with two different browser window sizes, 700x400 and 1500x800 pixels.

Figure 4.6: The complete video download size for RESS, RWD and RWD2RESS on an emulated iPhone 6, and a desktop with two different browser window sizes, 700x400 and 1500x800 pixels.
Chapter 5

Discussion

This chapter discusses the results from the two performance tests, and then the method.

5.1 Result

This is not a full sized study so all large deviations make a big imprint on the result. First the results will be discussed and then desktop and iPhone results are compared and discussed separately, followed by a discussion that compares RESS, RWD and RWD2RESS.

5.1.1 First page performance

The performance results for the first page was mostly as expected. The tests were performed on a public server in Denmark that ran the applications, and the client devices were located in Dulles, VA, USA. The biggest deviations are found in the time. Since there is no correlation between differences in time and size of downloaded bytes for each individual run, network traffic problems like packet losses are probably the cause for these fluctuations. This behavior is to be expected in reality for a user.

If you compare the desktop results in the first view, the RESS application is the slowest. This is expected since it uses some time to create a HTTP cookie, and then makes a reload to download the actual application. The RWD2RESS application does not have the same reload and the cookie is created after page load, with default set image and media size until the cookie is created. This could possibly explain the slightly faster activity closure time even if both RESS and RWD2RESS create a cookie.

The iPhones all have a slower response. The largest contributor to this is probably the smaller bandwidth and longer RTT. Since there is hard to control hardware specifications, discussing the impact of that would just be speculations and a bit out of scope for this project. For all devices and applications the repeat view is faster, in almost every case cutting the time in half. This is caching in work. In opposite to the PC tests the RESS and the m.SSA application is done loading first of all applications. This is probably due to the fact that they do not deliver any images to the front page, as all other applications.

5.1.2 Total application download

The outcome of visiting all templates in the RESS, RWD and the RWD2RESS applications had an overall expected outcome. That the total image size for all RWD
applications and the desktop versions for RESS is larger than for the RWD2RESS application, see figure 4.5. This is due to that the image sizes were optimized in the process of creating RWD2RESS, then also minimizing the largest image. These results help to illustrate and make conclusions as long as this is kept in mind. RESS in its mobile version does not serve any images except for the logo in the navigation menu, and therefore, is very small in comparison. The reason that the RESS has the same size in download for both desktop screen sizes is that there was no optimization for different screen sizes. For RESS there is only a separation for mobile and other devices.

Also the downloaded video size, I expected RWD2RESS to serve the same video size as the RESS, since they had the same optimization, see figure 4.6. I also expected the larger screen (1500x800) to result in the same video size for RWD and RWD2RESS. The difference here that the client is served with three different videos, for different systems, and they differ a bit in size. When running RWD2RESS the client has chosen a different version. Also here the result illustrate the effects of the different applications.

5.2 Method

It was a challenge to find documentation regarding RWD, SSA or RESS. Blogs explaining theory and tutorials are available on internet, and there have been published some books on the matter, as an extension to the blog posts. Since RWD, SSA and RESS all are based on common technologies, I have focused more on reading documentation on the parts that are the foundation for the different approaches.

To use webpagetest.org a measurement tool was, in hindsight, a mistake. It would have been better to use a tool with more fine grained control that could present more detailed data when deviations that could not be explained occurred.

This study tries to compare different solutions to optimize user experience and network traffic in common web applications. The applications implemented are not optimized in every sense, due to time limitations. For instance are image files not compressed, and no real effort has been made to optimize the size of image and media files. The aim was to keep all applications equivalent in regards of content size, so that a comparison between the applications would be meaningful.
Chapter 6

Conclusion

The purpose of this project was to compare common web design techniques, with a focus on Responsive + Server Side. Five different applications have been built and some measurements have been made with some promising results, and one question left with an unclear answer.

One of the major tasks was to find out what kind of projects are suitable for RESS. RESS is a fast and easy way to implement responsive images and media files, and the save in data downloads for RWD2RESS are significant compared to how most RWD applications are implemented, see figures 4.5 and 4.6. To implement responsive images would benefit both users and service providers. Less network traffic would decrease energy consumption due to less workload, and also decrease the data traffic bill for both parties. Therefore RESS is a candidate to be considered when the end-users are likely to use a variety of devices and performance is an important factor. The reduction in network traffic depends on, besides implementation, how large the application content is. The heavier the content in an application is, the more beneficial RESS gets. RESS should also be taken into consideration when an application that shall serve different functionalities to different devices is desired. There are not really any security issues with the data stored in the HTTP cookie, so no extra security measures has to be taken into consideration when choosing RESS.

It is hard to compare the workload of developing and maintaining a RESS application to RWD and SSA. It depends on the level of optimization. If you just want to optimize image and media files, there is not much extra work put into the implementation. There will be some extra work in the design process, when decisions have to be made on how many breakpoints there should be and how many different sizes the images and media files should be provided. Also in the quality assurance (QA) phase there will be some extra workload: every breakpoint and device must reply with the right content. If you really want to separate the code like was done in the RESS application in this project, you basically have two code bases to design, develop and maintain. Regarding maintenance you will have the extra QA steps when updating or improving the application in respect to RWD. In comparison to SSA you will not have much of a difference if you have separated the application into mobile and a general version.

Whether a RESS implementation can enhance the response time for mobile users depends on how you choose to implement the mobile version. The RESS application in this project was faster, main reason being that only one image is downloaded instead of seven in the first page performance test. As the test results showed the first page performance time for the RWD2RESS was longer. It is hard to tell what causes a second extra activity time. It is probably the extra script, the cookieHandler, that has to been
loaded and run, and the slower network traffic impact might scale from the PC tests. Then it would just be a first page problem, and the gain from optimizing size of content will first come clear as you use the application more. Further testing has to be done to answer this question.

It was quite easy to transform the RWD application into a RESS application, since the cookieHandler script already was implemented. This transformation can be implemented, once the cookie is created, in small steps, template by template, image by image, media file by media file. That is a huge advantage if you are working on a bigger project, since you do not have to do everything at once, but can prioritize the changes after how much they effect performance of the application. If a multiple authoring approach, for different functionalities or content, is desired with a RWD project as starting point, it would be technically easy to implement. This could also be done in small steps, template by template.

**Future work**

There are some possible issues with conflicting optimizations techniques, like caching [23]. A thorough investigation that identifies conflicting areas and proposes solutions could be a next step in the examination of RESS.

To develop a basic RESS framework that eases the implementation process would be desirable. The focus should be on the client part, JavaScript, since it is back-end independent. The code produced in this project only provides basic features and functionalities. Adding some JavaScript HTML DOM EventListeners so that the HTTP cookie can be updated, e.g. if browser window size or device orientation changes, could be useful. To make it independent from device detection libraries, that part could be left out, or be a settings option, since they might have different naming standards of features. It could also be interesting to have settings opportunities to what features to detect.
Bibliography


Appendix A

cookieHandler.js

```javascript
var cookieHandler = {
    // object with functions that can update or set device features
    // that 51Degrees might not have information about
    capMap51D : { ... },
    // object with functions that can add extra features that
    // 51Degrees do not cover
    capMap: {
        'csstransforms3d' : function () { return Modernizr.csstransforms3d; },
        'width' : function () { return window.innerWidth; },
        'height' : function () { return window.innerHeight; }
    },
    /∗∗
    ∗ cookieExists(cookieName)
    ∗ searches for a cookie named cookieName by splitting the sting of
    ∗ all cookies
    ∗ and then returns true or false.
    ∗ @param cookieName
    ∗ @return true if cookie exists, else false
    ∗ /
    cookieExists: function(cookieName) { ... },

    /∗∗
    ∗ getCookie(cookieName)
    ∗ serches for a cookie named cookieName by splitting the string of
    ∗ all cookies
    ∗ and then returns the cookie without name
    ∗ @param cookieName
    ∗ @return the wanted cookie as a string if found, else false
    ∗ /
    getCookie: function(cookieName) { ... },

    /∗∗ getDeviceType() 
    ∗ Returns the device type as stated in the cookie
    ∗ @param feature name of the feature
    ∗ @param cookieName name of cookie to examin. Default value 'RESS'
    ∗ /
    getDeviceFeature: function(feature, cookieName) {
        //argument check
        if(!feature)
            return 'mobile';
        cookieName = (cookieName ? cookieName : 'RESS');
```
//get cookie and feature
var cookie = this.getCookie(cookieName);
if(!(cookie) {
    var message = "cookieHandler:getDeviceFeature(feature, cookieName):Cookie "+ cookieName + ": does not seem to exist. Returning status 'Unknown' to feature " + feature + ":";
    console.error(message);
    alert(message);
    return "Unknown";
})
features = this.getDeviceFeatures(cookie);
return features[feature];
}

/**
getDeviceFeatures(cookie)
* parses the stringified cookie value and returns it as JSON object
* @param Cookie which should contain device features stringified
* @return object with device features
*/
getDeviceFeatures: function (cookie) {
    return JSON.parse(decodeURIComponent(cookie));
},

/**
getFeatureStatus(feature)
* Evaluates status of feature client side and returns status
* @param The feature to update
* @return found feature status, else 'Unknown'
*/
getFeatureStatus: function (key) {
    if(this.capMap51D.hasOwnProperty(key)){
        return this.capMap51D[key]();
    }
    return 'Unknown';
},

/**
updateCookieFeatures(cookie)
* Goes through all device features in coookie, and tries to update the unknown.
* @param A cookie to update with features
* @return void, updates the cookie
*/
updateCookieFeatures: function(cookie) {
    var features = null;
    features = this.getDeviceFeatures(cookie);
    while(!(features){}
        for (var key in features) {
            //go through all features and update the ones we are missing
            if(features[key] == 'Unknown') {
                features[key] = this.getFeatureStatus(key);
            }
        },
    //Add Modernizr detectables not in 51Degrees
    for(var key in this.capMap) {
        //go through all capabilities we are interested in
        features[key] = this.capMap[key]();
    }
    //overwrite cookie
Appendix A. *cookieHandler.js*

---

```javascript
document.cookie = "RESS=" + JSON.stringify(features) + '; expires=' + (new Date().getTime() + 1800) + '; path=/';
}

/**
 * setCookie(cookieName, value, expires, path)
 * @param cookiename: name of the cookie to be set, default "NoName"
 * @param value: cookie value
 * @param valid: how long the cookie should be valid, default ca one year
 * @param path: where the cookie is valid, default '/'
 */

setCookie: function(cookieName, value, valid, path) {
  if(!path) { path = '/'; }
  var expires = new Date();
  if(!valid) // set valid to roughly a year
    valid = 3600*24*365*1000;
  expires.setTime(expires.getTime() + valid);
  expires = expires.toGMTString(); // to make the cookie persistent
  if(!cookieName) { cookieName = "NoName"; }
  document.cookie = cookieName + "=" + value + ";expires=" + expires + ";path=" + path;
}
```

**Code Snippet A.1:** The JavaScript object `cookieHandler`. It is used for handling the RESS cookie.
Appendix B

Script that creates cookie

```html
<script>
$(document).ready(function() {
  $.ajax({
    url: "//CodeIgniter/index.php/home/get_cookie_data",
    async: false
  })
  .done(function(data) {
    cookieHandler.setCookie('RESS', data, 3600*24*365*1000, '/');
    if(!cookieHandler.updateCookieFeatures('RESS'))
      alert("Something went wrong with the cookie creation");
    else
      location.reload();
  })
  .fail(function() {
    alert("Something went wrong with the get_cookie_data ajax call");
  });
</script>

Code Snippet B.1: The script that uses the JavaScript object cookieHandler to create the cookie in RESS and RWD2RESS.
Appendix C

Application file structure

![Application file structure diagram]

Figure C.1: General file structure for the applications implemented. The cookieHandler.js and the employee-video.php are specific for RESS and m.SSA. In some applications the directives and templates are HTML files. The underlying structure for images and videos differ between applications.
Appendix D

Links raw test data

Below are the URL’s to the raw data of the first page performance tests.

<table>
<thead>
<tr>
<th>Application and device</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESS PC</td>
<td><a href="http://www.webpagetest.org/result/141027_4C_FWZ">http://www.webpagetest.org/result/141027_4C_FWZ</a></td>
</tr>
<tr>
<td>RESS iPhone</td>
<td><a href="http://www.webpagetest.org/result/141027_03_MWK">http://www.webpagetest.org/result/141027_03_MWK</a></td>
</tr>
<tr>
<td>RWD PC</td>
<td><a href="http://www.webpagetest.org/result/141010_ND_CEP">http://www.webpagetest.org/result/141010_ND_CEP</a></td>
</tr>
<tr>
<td>RWD iPhone</td>
<td><a href="http://www.webpagetest.org/result/141010_1Y_MA5">http://www.webpagetest.org/result/141010_1Y_MA5</a></td>
</tr>
<tr>
<td>RWD2RESS PC</td>
<td><a href="http://www.webpagetest.org/result/141027_G1_GAG">http://www.webpagetest.org/result/141027_G1_GAG</a></td>
</tr>
<tr>
<td>RWD2RESS iPhone</td>
<td><a href="http://www.webpagetest.org/result/141027_14_MW8">http://www.webpagetest.org/result/141027_14_MW8</a></td>
</tr>
<tr>
<td>SSA PC</td>
<td><a href="http://www.webpagetest.org/result/141010_TQ_CF3">http://www.webpagetest.org/result/141010_TQ_CF3</a></td>
</tr>
<tr>
<td>SSA iPhone</td>
<td><a href="http://www.webpagetest.org/result/141010_M6_MAH">http://www.webpagetest.org/result/141010_M6_MAH</a></td>
</tr>
<tr>
<td>m.SSA PC</td>
<td><a href="http://www.webpagetest.org/result/141010_EK_CFS">http://www.webpagetest.org/result/141010_EK_CFS</a></td>
</tr>
<tr>
<td>m.SSA iPhone</td>
<td><a href="http://www.webpagetest.org/result/141010_Q1_MAN">http://www.webpagetest.org/result/141010_Q1_MAN</a></td>
</tr>
</tbody>
</table>
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Jóhann Lindell