Linköping University | Department of Computer Science
Bachelor thesis, 16 ECTS | Computer Engineering
2016 | LIU-IDA/LITH-EX-G--16/077--SE

Designing a custom-made minimal CRM web application for performance and varying screen sizes

Tobias Lundgren

Supervisor : Erik Berglund
Examiner : Erik Berglund

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Abstract

In a world where the using of applications to simplify task has been more popular in both business and private. The need of an custom-made solution will be very interesting. But it is not enough in many cases, for satisfies its purpose. Sometimes the application also has to achieve good performance and work on multiple devices in a proper way.

This thesis investigate what parameters that can used to analyze performance by measure the page load time and what methods that can be used to achieve good performance. This thesis also investigates how we can get our application to adapts varying screen sizes on different devices.

Acknowledgments

Thanks to Göran Agardh and Rickard Hedman at Letsfaceit Nordic AB, for helping out to design the background for this thesis. Thanks to my supervisor Erik Berglund for great and fast guidance in this project. I would also like to thank Atle Werin for proof reading my thesis and providing me with helpful feedback.

Contents

List of Figures

List of Tables

Notation

Abbreviations	Description
API	Application Programming Interface
APP	Application (Web application in this context)
CRM	Customer Relationship Management
JS	JavaScript
PLT	Page Load Time
UI	User Interface

1 Introduction

The using of web applications to simplify tasks has been more popular in both business and private. The technique has become more easily accessible during the recent years[10]. But far from every task can be solved with the available applications today. As a result for that, the demand for custom-designed solutions are still highly interesting. In addition to resolve the client's needs, the application should also keep good performance with short load times.

This thesis will show you how we built an web application that achieving good performance with responsive web design. The purpose is to evaluate the applications performance and loading times.

This section includes the motivation why we should create a custom-made web application that are designed to achieving good performance and responsive design.

The delimitations chosen for this work will also be included.

1.1 Background

Letsfaceit Nordic AB (LFI) has around 35 employees and distributes cosmetic and makeup products. They got their own brand IDUN that is one of the biggest in mineral makeup. LFI is also an distributors for brands like *Real Techniques*, *Ecotools* and *WetnWild*. The company has enjoyed a tremendous growth and has been suffering on the sales.

LFI has a couple of sellers that daily visits some of their about 2,000 customers around in Sweden. The sellers shall report what has been conducted in the stores and what should be done to the next visit. These data were today recorded in an big *excel-file*[1]

This makes it difficult to fill in the file and organize data from the sellers visits in an easy way. So the company wanted to examine if a *Customer Relationship Management* system would facilitate their management.

LFI expands more and more, both in Sweden and on international markets; This means that the problem is increasing and may have the effect of a bottleneck in their business for the future.

The company's CEO decided to done some research about what system would cover their requirements to facilitate their management. They looked at many well-known CRM sys-

[1] https://products.office.com/sv-se/excel

tem's, but they wasn't found something that solved their problem, so an existing system has been excluded to use.

They wanted a custom-made solution to solve their problem. A system that only has the approaches that is related to their business to make it simple by not having functions that wouldn't be used to save important time in order to achieve economic efficiency.

External requirements

The development of web applications and the implementation of the project has been based on a number of external basic requirements.

1. The web application will be developed to work on an Apple iPhone 6 screen and an 13" laptop.
2. The web application will be implemented in HTML, JavaScript, Bootstrap, jQuery, CSS.

1.2 Research questions

This thesis contributes one main question:

1. *How do we built the minimal CRM web application to achieve good performance and works on multiple devices?*

   The main question can further decompose to these two more focused questions:

   a) *How do we design the minimal CRM web application to achieve good performance?*

   b) *How do we design the minimal CRM web application so it adapts to varying screen sizes?*

The main goal with this thesis is to study what approaches that can be used to evaluate the performance and load times of our custom-made CRM web application. Thus to get load time and performance calculations, and partly to make recommendations that could lead to performance improvements. We will also investigate how do we build the application to make it work on different platforms like desktops and mobile devices.

1.3 Delimitations

Developing web applications is a really wide area. Therefore this thesis will only handle parts that are relevant for the purpose of this dissertation.

Only the implementation parts that was basis for important decisions or comply with a purpose will treat in this dissertation.

In other words, only the actual implementation methods will be addressed and the evaluation will examine methods that are relevant to web application performance and how to build an application with responsive web design. We will not analyze each function but only the total page load times.

2 Theory

Creating web applications with good performance that suits multiple different devices and screen sizes are are a wide area. In this thesis we investigate some development approaches to measure and evaluate the performance and some developing methods to achieve responsive design.

2.1 Web applications

The section relates to what a Web Application is and how it is defined in the project.

An application is made up of sequences of code as programmed with a programming language to execute instructions as measured over a computer. All instructions together as one program solves a problem.

Together this is called an executable computer program. [1]

Web Applications is a collection name for the client to server application that runs in a web browser. The structure of building web applications consists of dynamic elements based on the user, time, inputs and so on. Developers use JavaScript to implement both logic and data as well as the interface in their web applications [7].

The architecture of web applications consists of multiple distinct modules that are built to perform a well defined sub task. Each of these modules is independent and separated from each other but still work together as a single executable program. [8]

A simple version of the architecture for an web application consist of only two modules, the front-end and the back-end. In this structure, the front-end module manages the user interface application. This application are built in HTML and CSS languages. Where the HTML files consists of the interface elements. And the CSS files are used to style the HTML interface elements. This combined with JavaScript are used to improve the applications utility and provide an attractive appearance.

The back-end module can in turn be divided into two parts. The first part takes care of the handling for the data that the application needs to store and structuring this. This is often done in relational databases and using the language SQL to reach and manipulate the database. [4]

The second part takes care of the communication between the client and the server either by using object oriented language as Java or Ruby or scripting languages (e.g., PHP). [13]

Code conventions

To make the source code of the web application easier to understand, code conventions can be followed. The conventions for HTML, CSS and JavaScript are easy to enforce because it's the same coding standard as Java [8].

Databases

Databases can be useful to improve the features in an web application. Databases can be hard to maintain because the parts are not as flexible as other parts of the code. It is significantly helpful by investing a lot of time in designing the database structure and document it to avoid to make changes to the database layout when you begin collecting data. [8] The data accesses from the database via API's that are programmed to perform certain lines of database manipulation commands [4].

User experience

User experience is the term that means what experience the user can get from the application. User experience can be usable even outside the application if it for example has features like push notifications or to wake up an application during certain conditions. This conditions can be like a location change or a specific time.

User experience can also be experienced outside of an application, by e.g. get push notifications if any conditions have changed. The user experience can be divided into two primary categories:

The context – Elements that not can be controlled or changed in the application but must be understood of the users. It can be like the affordance[1] of the hardware related to the application or *user interface* conventions (how the user is accustomed to managing a UI). This part will affect the users expectations how the application is to be used. This expectations can differ from user to user.

The implementation – Elements that can be controlled or changed like the design and performance of an application, or features that can integrate with the platform such as the accelerometer, or notifications.

These properties are the basis for how the application will be perceived by the user, and how many possible ways the user can choose to interact with the entire system. [5] Responsive web design for the user interface enriching the *user experience* and can be used for cross-platform development[6].

2.2 Cross-platform development

This section describes what cross-platform development and what means by web-based development.

Platform independent, from the English "cross-platform" means that the application can run with the same code into different devices without having to be rewritten [7].

One of the main reasons for not developing a specific application to all platforms, is that all platforms differ significantly and do not work the same on all devices.

Cross-platform development approaches have been developed to increasing productivity by address this problem. This is solved by letting the developer build an application for a range of platforms in one step and avoiding repetition by write differ applications to all different platforms

Because there are well-functioning tool for cross-platform development, it is not certain that you have a lot to gain by developing a separate applications for different devices [7].

[1] Affordance means the property that invite to be used in a certain way.

Web-based solutions

The web-based solutions uses the browser as the interpreter of the code, this gives to have interoperability of all devices with web browsers. This is important that the browser of the device is the compatibility with the code you write. Web-based solutions are written in languages like HTML, CSS and JavaScript thus makes it easy to develop one application that suits multiple devices. Further, they are often easier to deploy when no software installation are needed [7].

An disadvantage with web-based solutions is that they have limited access to the underlying hardware and data resources. Further *Spyros Xanthopoulos and Stelios Xinogalo, 2013* describes that the web applications are browser-based and can't be installed, thus extra time is needed because the source code must downloaded before you can use the app. This also means that the system has to be connected to the Internet. And the app may be unavailable after a network failure, which is one of the biggest drawbacks.[15]

2.3 Responsive design

Many devices has different screen size and different resolutions. Developing for each platform would be impractical if it wasn't for the responsive web design approach. Responsive web design technique adapts the design to the user's platform and can be used for cross-platform development. [3] This approach is based on an abstract thinking some main techniques are fluid layouts (Katrien De Graeve, 2013).

Fluid layouts are built with CSS3 that includes *media queries* that have listeners to react on environment changes like max-width, device-width and orientation in which your page is running through scripts. The layout design are build with fluid grids that have columns with element class specified in the CSS style sheet with *width* based on percents of the screen size and *float* settings to move and transform elements in the layout automatically. [3]

Listing 2.1: Example of responsive columns

```
1  .col-one-third{
2      width: 33%;
3      float: left;
4  }
5  .col-one-half{
6      width: 50%;
7      float: left;
8  }
```

One important key issue that responsive design has to solve is to get it work with images. There are many image resize techniques to change the images proportionately, and multiple of them are easily done. Ethan Marcotte's article on fluid images are the most popular option. It's based on an experiment by Richard Rutter as uses the CSS's **max-width** property.

Listing 2.2: Responsive images with CSS

```
1  img{
2      max-width: 100%;
3  }
```

This option will set the images maximum width to 100 % of the element size. This option gives thus the effect of an responsive image that fits an desktop screen but also the screen on a smaller device. [3] [6]

An front-end framework can be used as a toolkit to build a site responsive by follow certain guidelines and documentation for the framework. Bootstrap is an front-end framework that are easy to customize with standard conventions from well-defined classes with clean and practical documentation. [11]

2.4 Web application performance

The section explains the theory of web page speed and performance. This section also include guidelines for making web pages faster. In addition to the appearance aspects, the performance is it another big factor that contributes the user experience of web applications.

The work-flow how browsers load web pages can be used to measure web performance. The *HTML Parser* transform the raw HTML to a Document Object Model (DOM) tree. The HTML Parser is the main part of the load process. The nodes in the DOM tree represent the HTML tags and are an middleware between the interface and the DOM nodes.

Web application performance consider two measures to characterize web page load times (PLT).

The first measure consists of the time it takes for the web browser to fetch sufficient content and parse HTML to start rendering the page.

When the initial HTML parse are completely loaded and parsed, the *World Wide Web Consortium's* (W3C) DOM event *DOMContentLoaded*[2] is fired. It does not take account into embedded style-sheets, images and sub-frames to finish loading.

The second measure part consists of the total time to fetch all content and render the web page completely. After the fully-loaded page, the other WC3 DOM event *load* is fired. [14] [2]

The above W3C DOM events can be used as metrics for measuring web performance. The *DOMContentLoaded* event is the most popular web performance metrics because it is more appropriate (Mozilla Developer Network, 2016).

Web services success is a correlation by the page download delay, the time data spend access and display content from the request within the web site [9] [12]. Metrics are needed to help organizations generate more effective web sites by making performance measurements.

The effect of load time affects the user experience thus how satisfied the users are. There are also very important to keep good performance on the site because it is commercial services that measure the page load time and uses it to rank the web site [2].

The most websites loads many CSS and JavaScript objects; (Butkiewicz et al., 2011) invention after some research of which factor that are most important for the PLT's by analyze which metrics that prevents the page to be rendered. They states that the number of requested objects is that factor that impacts the load times most. The number of distinct servers that are used to download content also increases the time how quickly a page can be loaded because it needs multiple HTTP/TCP connections.

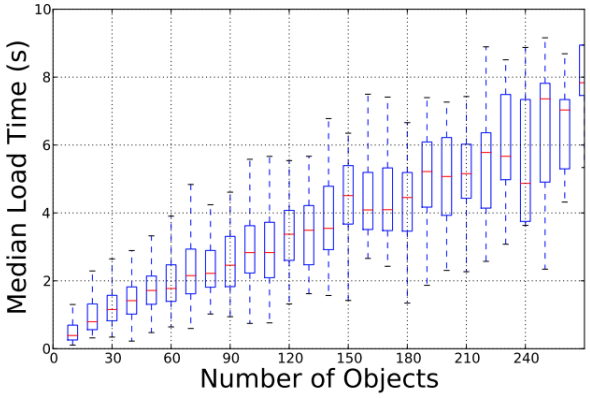


Figure 2.1: Correlation between Load Time and number of objects to fetch. [2]

[2]https://developer.mozilla.org/en-US/docs/Web/Events/DOMContentLoaded

One of their test was to check how many servers that were requested by the servers by looking at qualified domain name e.g., *bar.foo.com*. The results was that 25-55 % of the websites require to contact over 10 distinct servers. This means that even lightweight sites can be slower. But the impact on download time is relatively low. [2]

Optimize site performance

Google Developers has listed some best practises for speeding up the load time. The list how to do:

Compressing JavaScript – Minimize the amount and size of the JavaScript files is the best way to reduce the response time when the page loads. This can be done by remove unused code like spaces and comments, removing unneeded variables and rename local variables and function to short names, and other optimizations to shrink the code. It exists multiple tools to compressing JavaScript files. (Robert Bowdidge, Software Engineer, 2015)

Optimizing CSS – By make the CSS files that sends to the browser smaller, the website loads faster. Some best practise are to use single rule CSS by for example combine these rules:

```
1         h1 { color: black; }
2         p { color: black; }
```

to a single rule:

```
1       h1, p { color: black; }
```

(Jens Meiert, 2015)

Prefetching resources – means that resources loads the file before it's needed. For example if a user request one image in a range, the user will probably also request next image in the range. By changing the order that resources are fetching can often get the web pages benefit. The website shouldn't fetching data from a new page before the current page is done. (Shawn Brenneman, 2015)

2.5 Evaluation methods

This section includes the theory behind the approaches in the method chapter. How to evaluate performance, load times and how the app adapts to varying screen size.

Chrome DevTools timeline

Chrome DevTools Timeline panel is the best place to start analyze the application for performance issues (Kayce Basques, 2016). This tool makes it possible to see the Timeline for how the website was loaded. You can see the total time spending on the different stages (Loading, Scripting, Rendering, Painting, Idling and Other). The time-line shows the DOMContent-Loaded event(blue line in 2.2), time to first paint(green line in 2.2) and load event(red line in 2.2). Chrome DevTools Timeline has also an JavaScript Profile evaluate function which can be activated. This function recording all JavaScript functions that was called and shows them in a *flame chart*.

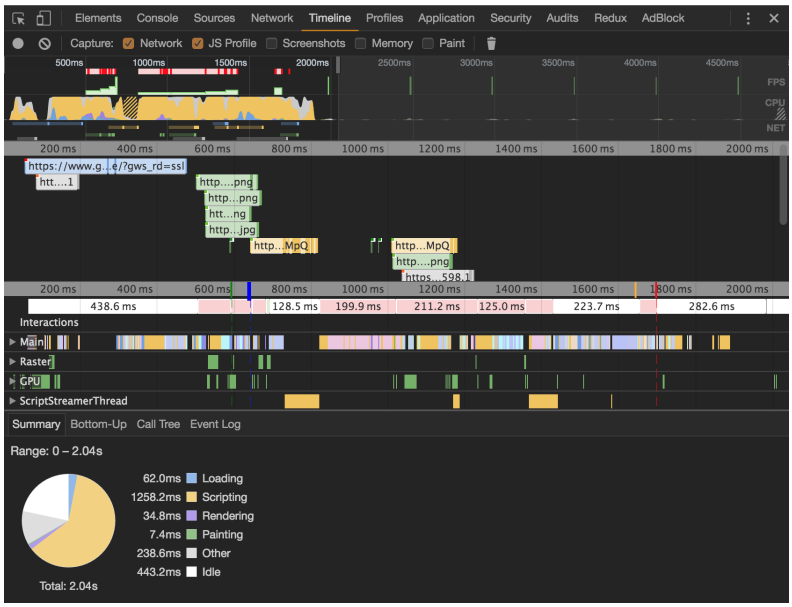


Figure 2.2: A screenshot of Chrome DevTools Timeline results from recording https://www.google.se (Catched 25 October, 2016)

Chrome DevTools elements

The elements tool have features to design web applications so it adapts varying screen sizes. The *styles tool* provides the possibility to live modifies the DOM elements and change the CSS styles so the settings can be tested live before the changes are implemented in the app. The *device toolbar* provides the possibility to manually customize the screen size live that the application will run on. The inspector mode is a feature that gives you the possibility to hover over an DOM element and see the implemented settings for the HTML and CSS styles. Chrome DevTools has a *Local modification* tool that saves all the local live changes.

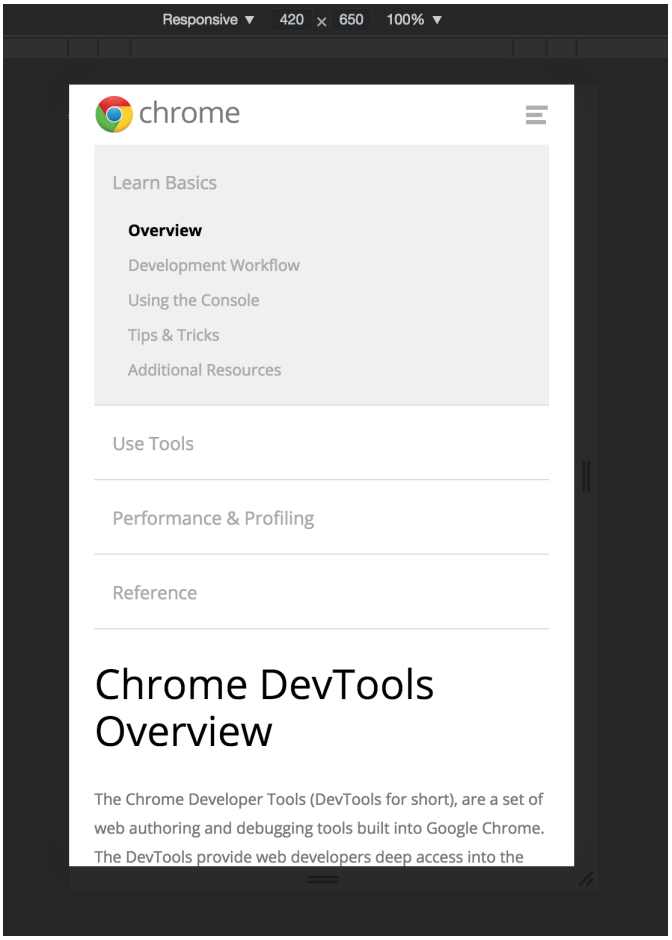


Figure 2.3: A screenshot of Chrome DevTools Device mode toolbar examine https://developer.chrome.com/devtools (Catched 25 October, 2016)

3 Method

This chapter focuses on the methods used to analyze the performance and also the responsiveness of the web application. How measurements are made and how we evaluated it.

3.1 Analyze performance

What approaches can be used to to provide good performance for the web application?

To measure the performance for the web application we evaluated the PLT by record the *DOMContentLoaded* event because it is much more appropriate (Mozilla Developer Network, 2016). We also recorded the total PLT by record the W3C DOM *load* event 2.4 to see how much it differs on load time for only HTML to completely parse and how long time it takes for the application to load the entire page.

By create a dataset with all the PLT records we will draw up charts to easier get an overview of the PLT's. The dataset contains of total ten measurements each. Five measurements for the login page and logged-in and each recorded values for the *DOMContentLoaded* and *load* event. We also recorded the same measures on an live remote server to compare how much the PLT differ on an local environment against the live environment.

By analyze the measurements we will get an complete result for the PLT in our application based on the *DOMConentLoad* and *load* event.

We use the tool Chrome DevTools Timeline 2.5 to done the measurements by record the *DOMContentLoaded* event (the blue line in figure 2.2) and the *load* event (the red line in figure 2.2).

The Chrome DevTool Timeline recording is a helpful tool to see how the scripts impacts on the performance to see if some parts in the code takes suspiciously long to evaluate. To get an more detailed analysis the *JS Profiler* can be enable to provide a more detailed performance analysis by see exactly which JS function are called and how long each evaluates. It is an effective method to evaluate the performance of your web application.

The Chrome DevTools Timeline are also a useful tool when developing applications to see were execution time are spent in your JS functions and what impact your JavaScript code has on the CPU by use the *CPU profiles* during run time. It is an effective for optimizing your web application's performance.

3.2 Analyze Responsivity

What kind of developing approach can be used to provide the design of the web application see how the UI adapts to the devices varying screen sizes?

We have also used the Chrome DevTools to measure how the web UI will adapts to the varying screen sizes. But this time the *elements mode* were used because this tool got all the necessary features.

The styles feature were used to determine what CSS settings that worked best together to get the effect of responsive design on different devices.

The inspector mode is used to easily find the HTML element that that would be examine.

The elements tool combined with the device toolbar are used to test the environment that the application will run on. We used the element tool to modifies the DOM elements and change the CSS styles live. And by use the device toolbar to manually change the screen height and weight settings like *figure 2.4*, we could examine how the application works and adapts the varying screen sizes.

Local modifications is another feature that we used to get a history of the local live changes in the code to be able to go back and find the perfect settings for the application.

Implementation

To implement the responsive web design to provide the application to adapt on varying screen sizes, we use an front-end framework like Bootstrap that facilitates the implementation.

4 Results

In this chapter we look at how we design the minimal CRM web application to achieve good performance and how it can adapts varying screen sizes.

4.1 System description

In this section we look at what our system consists of that are relevant for the performance and the responsive design. It also include the optimizations made to increase performance of the web application.

The creation of our minimal CRM web application are designed in a cross-platform development way based on our theory chapter and the external requirements. It consists of a bunch of important web-based parts to achieve good performance. We are using code lanuages as HTML, CSS and JavaScript, with the same coding standard as Java according to the theory 2.1.

The back-end are based on Node.js[1] because it is JavaScript based and scalable.

The following optimizations were implemented:

- Compressing JavaScript
- Optimizing CSS
- Prefetching resources

We are using *Gulp*[2] to built the application because it can be used to create automated tasks that compressing JavaScript and Optimizing CSS.

To get the Prefetching resources effect we investigate placed all the object as should view first at the top. And the objects that have the same priority further down.

This will give the effect of better performance because the user will get the valuable content first.

We chose to implement these because it was listed as a best practice to speed up the PLT 2.4.

[1] http://gulpjs.com/
[2] https://nodejs.org/en/

The system are built with the React.js [3] JavaScript framework for building web interfaces. We chose React because it is one of the most popular JavaScript frameworks today based on the GitHub trendings for JavaScript[4]. Another reason is that React is created by Facebook. That combine with the fact is open source is perfect for our purpose.

The application are used MongoDB[5] as database to improve the features of the web application 2.1.

To design the application to adapt varying screen sizes we implemented Bootstrap because it is a front-end framework that makes it easier to build our application to get responsive design and adapt varying screen sizes. The implementation describes in 4.3.

4.2 Performance measurements

The performance measurements are stored in the tables below. First we measured the login page. Thereafter we measured the logged-in page. The average, best, worst and difference for the PLT was calculated for all measurements, both *DOMContentLoaded* and the *load* event.

The login page were chosen because the login page consists of a little number of objects, more or less two input fields for username and password and an log-in button. The logged-in page were chosen because it consists of a large number of objects too see how the correlation between number of objects impacts the PLT.

The measurements were made on an just fresh booted Apple iMac 27-inch, 4 GHz Intel Core i7 256GB Flash Storage, 24GB memory machine that running Mac OS Sierra. The tools were Google Chrome and the built-in DevTools Timeline in privacy mode. The cache was cleared and garbage collector was trigged between each measurement.

This tables shows which DOM event were measured and how much time recorded. We chose an linear 2D-staple diagram to present the results for each table with records to make it easier to realize and analyze.

[3] https://facebook.github.io/react/
[4] https://github.com/trending/javascript
[5] https://www.mongodb.com/

Local environment

The first measurements were performed to see the PLT in an local environment.

Table 4.1: Login page records

Login page		
DOM event	*DOMContentLoaded*	*load*
time (ms)	550 530 417 435 406	546 532 425 439 410
Average	**467,6**	**470,4**
Best	**406**	**410**
Worst	**550**	**546**
Difference	**144**	**136**

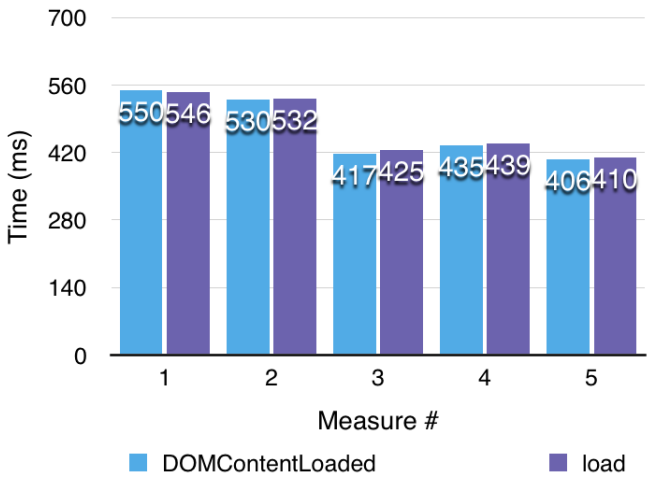


Figure 4.1: Performance measurement for the *DOMContentLoaded* and *load* events on the login page in an local environment. Five different measurements for each event was performed.

Table 4.2: Logged-in page records

Logged-in page		
DOM event	*DOMContentLoaded*	*load*
time (ms)	602 577 572 554 569	623 597 590 569 582
Average	**574,8**	**592,2**
Best	**554**	**569**
Worst	**602**	**623**
Difference	**48**	**54**

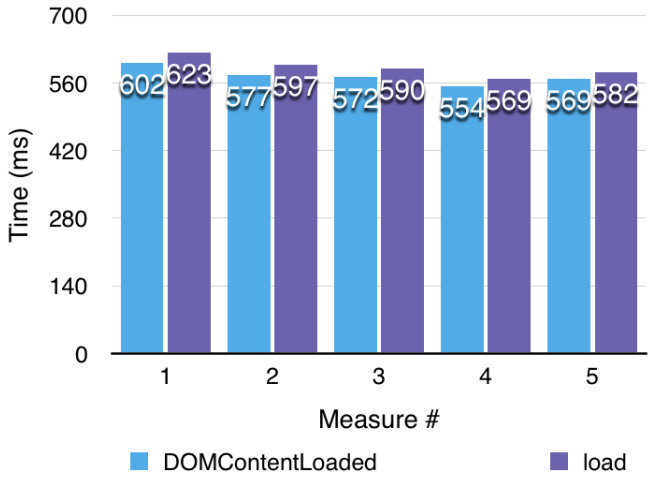


Figure 4.2: Performance measurements for the *DOMContentLoaded* and *load* events on the logged-in page in an local environment. Five different measurements for each event was performed.

The results of the difference between the average *DOMContentLoaded* event for the login page that have a smaller number of objects against the logged-in page, an increase of 23 % according to 4.1.

$$574,8/467,6 = 1,23 \tag{4.1}$$

Live environment

We also done some measurements on the application running on a live remote server at Herouku's[6] Cloud Application Platform to compare how the performance differs in an local environment against an remote environment. The measurements are collected in 4.3.

Table 4.3: Logged-in page records from an remote server

Logged-in page (remote server)		
DOM event	*DOMContentLoaded*	*load*
time (ms)	662	2540
	683	2530
	726	2130
	929	2230
	655	1620
Average	**731**	**2210**
Best	**655**	**1620**
Worst	**929**	**2540**
Difference	**274**	**920**

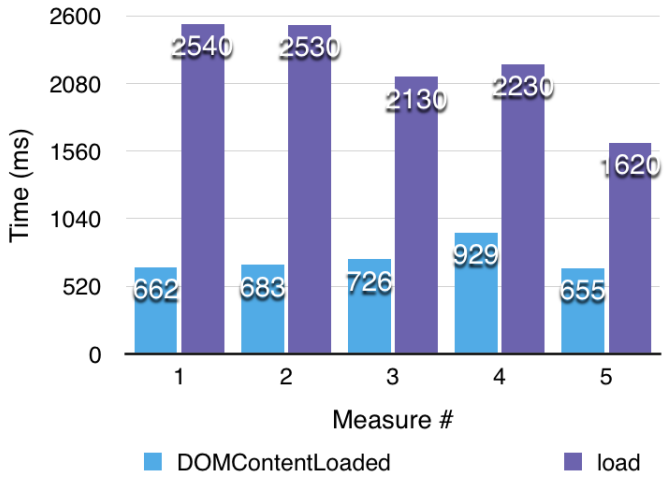


Figure 4.3: Performance measurement for the *DOMContentLoaded* and *load* events on the logged-in page in an live environment. Five different measurements for each event was performed.

The results of the difference between the average *load* time for the local environment and the live environment was that the live environment are 273 % longer according to 4.2.

$$2210/592,2 = 3,73 \tag{4.2}$$

It led us to examine deeper on what elements that affects the big difference. We could see that two of the loaded JPEG images stand for the greater part of the increase on the remote server. The obvious difference against the other images were the image size.

[6]https://www.heroku.com/

We decided to investigate the effect for the performance to optimize those images by reduce the image size by reduce the image quality to 70% by use Photoshop CC 2015 and then use ImageOptim[7] tool to compress the image-file. The purpose was to find out how much these two affected the load event.

> The JPEG image size was decreased from 2200KB to 673KB. And the other image from 1010KB to 261KB. It is a reduce of the size by 69,4% for the first one and 74,2% for the other.

The last performance measure was done after the image optimization:

Table 4.4: Logged-in page records from live server

Logged-in page after image optimization (remote server)		
DOM event	*DOMContentLoaded*	*load*
time (ms)	669 882 662 627 708	814 912 1240 910 1060
Average	**709,6**	**987,2**
Best	**627**	**814**
Worst	**882**	**1240**
Difference	**255**	**426**

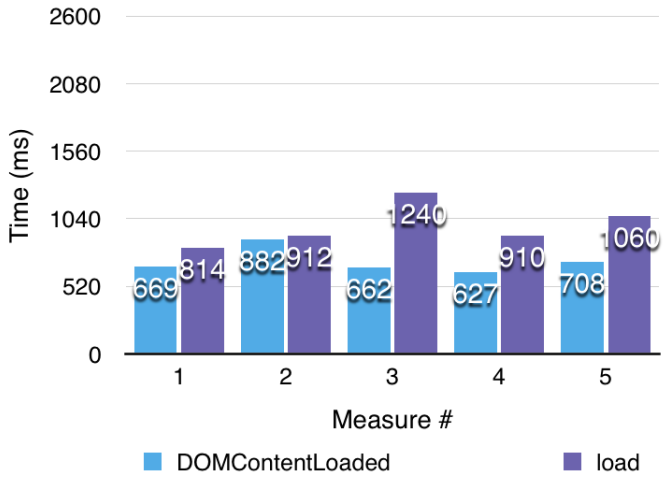


Figure 4.4: Performance measurement for the *DOMContentLoaded* and *load* events on the logged-in page in an remote environment after image optimizations. Five different measurements for each event was performed.

[7] https://imageoptim.com/mac

Comparison

To make it easier to see how big the differences are we made an diagram that include the *load* staple from 4.3 and 4.4

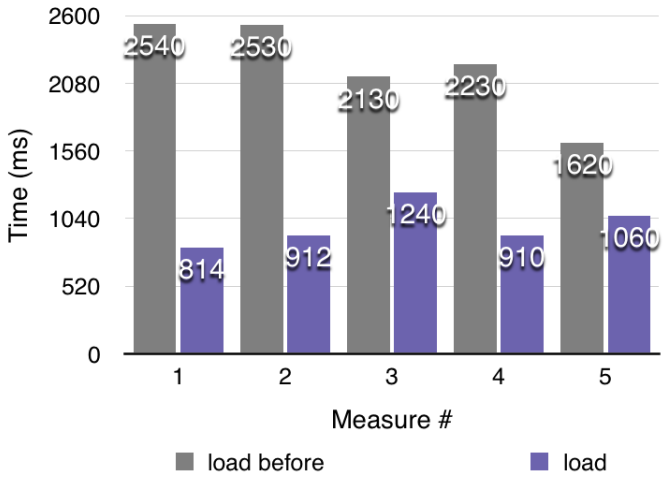


Figure 4.5: Performance measurement for the ***load* events** on the logged-in page in an remote environment before and after image optimizations.

The biggest difference between the worst and the best *load* event in 4.4 are so much as 1726ms / 68% less according to 4.3 and 4.4.

$$2540 - 814 = 1726ms \tag{4.3}$$

$$1726ms/2540ms = 0,68 \tag{4.4}$$

4.3 Responsive design

This section focuses on what approach we use to get our web application to adapts varying screen sizes.

We chose to use Bootstrap framework to get the fluid grid features as mentioned in 2.3. To be able to utilize the fluid grids we used the defined column classes in Bootstrap.

Listing 4.1: Example of HTML-code for an fluid grid example

```
1     <div class="col-sm-12 col-md-6">Content 1</div>
2     <div class="col-sm-12 col-md-6">Content 2</div>
```

The grids consists of totally 12 columns so this code will adjust the divs side-by-side if the screen-size are mid-sized. And if the screen-size are small-sized the first div will be placed before the first one.

With the Bootstrap-framework implemented, we also get responsive images by use the implemented *img-responsive* class within Bootstrap.

Listing 4.2: Example of HTML-code for an responsive image

```
1     <img src="images/pic01.jpg" class="img-responsive" />
```

That class include the CSS settings:

```
1 max-width: 100%, height: auto, and display:block
```

We also used Bootstrap to make our button adjusts after the varying screen size by the built in class *btn* according to:

```
1 <button type="button" class="btn btn-primary">Button</button>
```

Bootstrap is used because the includes classes automatically adapts to varying screen sizes and Bootstrap has great documentation that makes it easy to implement.

We used the Chrome DevTools elements mode with the inspector feature to test the implementations before it actually were implemented to save time.

Evaluation

We used the Chrome DevTools elements mode combined with the Device toolbar 3.2 to analyze the responsive design to see if the application can be used on varying screen sizes. We chose to examine the application on the iPhone 6 because it has the dimensions of 375x667 pixels which is smaller than our implemented breakpoint for mobile devices. We also chose to use examine the application on an desktop screen by have a screen width larger than 1024 pixels that is set to our upper breakpoint (1025x650 pixels).

The result of how the application adapts to varying screen sizes. First, we evaluate the application for the iPhone 6 screen (375x667 pixels), see figure 4.6 and 4.7.

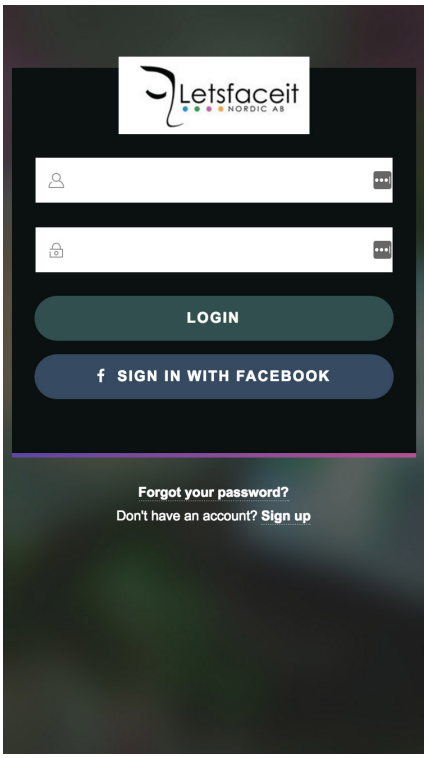


Figure 4.6: Screenshot of the login page from the application simulated on an device with screen size of 375x667 pixels.

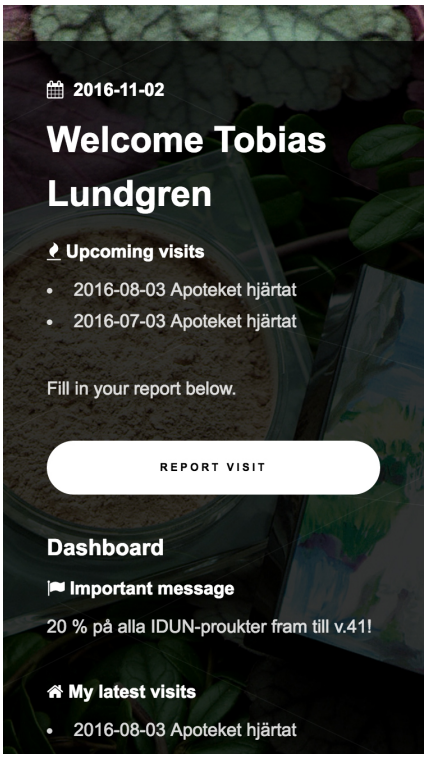


Figure 4.7: Screenshot of the logged-in page from the application simulated on an device with screen size of 375x667 pixels.

Second test evaluate the application simulated on an desktop screen (1025x650 pixels), see figure 4.8 and 4.9.

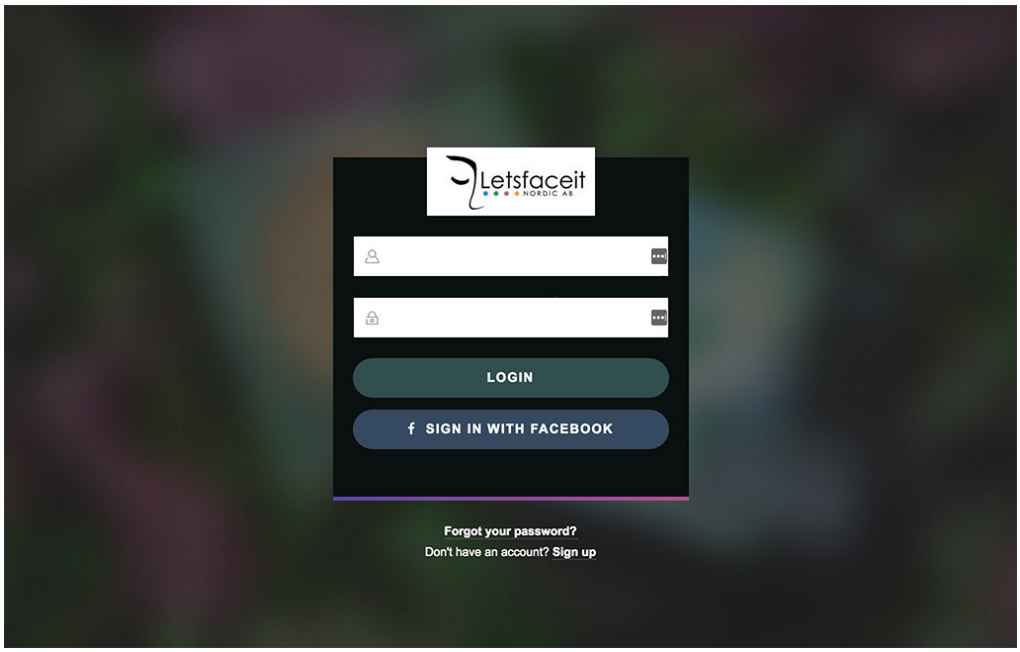


Figure 4.8: Screenshot of the login page from the application simulated on an device with screen size of 1025x650 pixels.

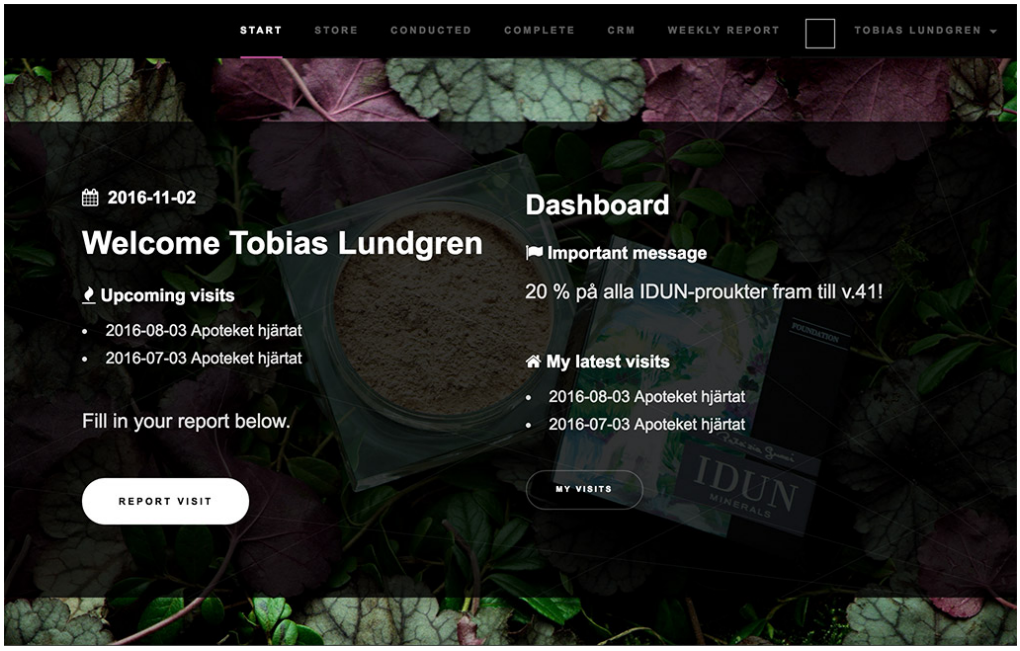


Figure 4.9: Screenshot of the logged-in page from the application simulated on an device with screen size of 1025x650 pixels.

We can see how the interface of the custom-made minimal CRM web application adjusts after the simulated device varying screen sizes.

5 Discussion

This chapter discuss the results, and the method used to get our results. What shortcomings it has and how have they could affected our work?

5.1 Method

Our approaches focuses on the W3C DOM *DOMContentLoad* and *load* events to done the measurements for the PLT.

We chose to both measure the *login page* and the *logged-in page* too see how much the PLT differs on page with different amount of DOM objects in or application. This method was very important in this study to see if our theory was correct.

The importance in measure the application in an optimal environment because the measures can differ a lot.

To measure the application performance both running on an local environment and a live environment on a remote server was also very important to see what factors that impacts the performance most. An drawback with our method where to measure on an local environment so the *load* event almost was fired at the same time as the *DOMContentLoaded* event.

In our case we would like to evaluate how we design our web application to achieve good performance. Therefore is it relevant to see how the application works in both an local and a live remote environment. The propose with the local environment is too see the PLT's without the delay of downloading content before the *load* event was fired. The live remote environment is a way to see how the PLT differs when the content is stored on a remote server instead of the local machine.

In our case I think the DevTools Network would be a better idea to just measure the *DOMContentLoaded* and *load* events because it's much easier to find the results and understand. In addition it also has an *disable cache* function so you don't have to done it manually between each measurement to be sure that some content are stored locally. This mode will also provides us with an clearly view over the fetching objects that facilitates to find the object as stands out. The timeline with JS profiler activated is better to investigate each function in a deeper level but we did not use it in our method because it was not the purpose according to the limitations.

We compared our *DOMContentLoaded* results of the PLT with the *DOMContentLoaded* event from Google[1] start page because we assume that they have good PLT on their site.

In our case it was unnecessary to use *JS profiles* or the *CPU profiles* in the DevTools Timeline mode because it is not necessary to measure the PLT which was the purpose of our study but are great for improve the performance more.

Responsive web design

Bootstrap was very easy to implement and use because it only needed to include a JavaScript-file and an style sheet including the classes. After that, it was very easy to apply in the web application.

The most usable were the fluid grid system that gave us the possibility to get almost all content adjustable according to the varying screen sizes.

5.2 Results

The first we noticed was how the correlation between the number of objects impacts PLT by an increase of 23 % with a larger number of objects between the login and the logged-in page according to 4.1 that conforming to the theory 2.4. Therefore, we chose to measure only the logged-in page because if it has good performance so the other side will also have it.

The second part that stood out was the main part of the notations from the results of the performance measurements 4.2. How much time it differ in time between the measurements of the *load* event for the application running in the local environment against the live environment on the remote server. That means it takes 273% longer *load* time in average to render the page and download all the content according to 4.2. The object type that was the slowest in our application was clearly the images because larger file size spending more time to download, especially from remote servers.

The improvements after the reduction in image size gave us total difference of 1726ms which are an total reduce of 68% from the PLT.

This resulted in an increase in both performance and user experience according to the theory 2.4 as proving that the PLT affects how satisfied the users are.

The results gave us the importance of analyze what factors that impacts the performance in our custom-made minimal CRM system in order to answer the question: "*How do we design the minimal CRM web application to achieve good performance?*" 1.2

Responsive web design

We could conclude that the application works on the iPhone 6 screen and also an 13" laptop. We also could increase the screen width to an large TV and the content will still be side by side and we use an max-width on an container around all content in the application so the content can be wider than 1980px because it is enough in our purpose.

[1] https://www.google.se

6 Conclusion

This chapter contains a summarization of the dissertations contribution for the purpose and the research questions.

We have showed that our method to make measurements of page load time work to analyze our web application to check if our system design keeps good performance. And how we can using the results to improve the applications page load time. We have shown that our method can be used to see which factors that influence our page load time.

The problem is that the measurements in the application gives different results depending on what environment our application is running on. I.e. it is a very small difference when the *DOMContentLoaded* and the *load* events are fired in a local environment; That is because objects can be retrieved locally without delay. But if we instead looks at the difference in a live environment on a remote server, results differs significantly. Therefore, the measurements must be made primarily in the environment in which the application will be used in order to make them as relevant as possible. We have also been shown that our method for designing our app so that it adapts to the screen size by using a modern framework works. We have also shown that our methods to evaluate and analyze responsively works.

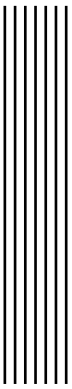
So what are the conclusions? For the first question: *DOMContentLoaded* is best to use for measure our applications performance because it is fired directly after the application are loaded and parsed. This is better without waiting for stylesheets, images and subframes because it may differ depending on how much objects(content) we have. But we have also shown that the *load* event can be good to measure for improve the performance in an effectively way by seeing each object that affecting the total page load time so you can see what items you should focus on to improve or remove.

For the second question: The front-end framework Bootstrap can use to design our app so it gets responsive web design by applying Bootstrap's built-in classes in our divs and images. This makes our minimal CRM web application to be designed with a method that adapts the user interface to varying screen sizes. It also works on all devices that have a modern web browser.

Both our conclusions are generalizable for all web applications and websites.

6.1 Future work

Possible future work based on our this thesis can be to investigate how we could get a lower PLT measured by *DOMContentLoaded* event. One way to do this would be to use the *JS profiles* to evaluate what JS function and what factor in the function that are consuming PLT. This would be good way to improve the performance further. Another way to improve the performance would be to use the *CPU profiles* to see what parameters that consuming most CPU in the web application. It could be quite difficult to improve the PLT significantly because the PLT already is relatively low.



Bibliography

[1] Alan F. Blackwell. "What is programming". In: *14th workshop of the Psychology of Programming Interest Group* (2002).

[2] Michael Butkiewicz, Harsha V. Madhyastha, and Vyas Sekar. "Understanding Website Complexity: Measurements, Metrics, and Implications". In: *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*. IMC '11. Berlin, Germany: ACM, 2011, pp. 313–328. ISBN: 978-1-4503-1013-0. DOI: 10.1145/2068816.2068846. URL: http://doi.acm.org/10.1145/2068816.2068846.

[3] Ben Callahan. *Responsive design. [Elektronic resource].* 2012.

[4] Sven Casteleyn, Florian Daniel, Peter Dolog, and Maristella Matera. *Engineering web applications*. Vol. 30. Springer, 2009.

[5] Andre Charland and Brian Leroux. "Mobile Application Development: Web vs. Native". In: *Commun. ACM* 54.5 (May 2011), pp. 49–53. ISSN: 0001-0782. DOI: 10.1145/1941487.1941504. URL: http://doi.acm.org/10.1145/1941487.1941504.

[6] Brett S Gardner. "Responsive web design: Enriching the user experience". In: *Sigma Journal: Inside the Digital Ecosystem* 11.1 (2011), pp. 13–19.

[7] Henning Heitkötter, Sebastian Hanschke, and Tim A. Majchrzak. *Evaluating Cross-Platform Development Approaches for Mobile Applications*. Ed. by José Cordeiro and Karl-Heinz Krempels. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 120–138.

[8] Adam Kolawa, Wendell Hicken, and Cynthia Dunlop. *Bulletproofing web applications*. M & T Books, 2002.

[9] Tom Leighton. "Improving Performance on the Internet". In: *Commun. ACM* 52.2 (Feb. 2009), pp. 44–51. ISSN: 0001-0782. DOI: 10.1145/1461928.1461944. URL: http://doi.acm.org/10.1145/1461928.1461944.

[10] Pattie Maes et al. "Agents that reduce work and information overload". In: *Communications of the ACM* 37.7 (1994), pp. 30–40.

[11] Alexandre Magno. *Mobile first Bootstrap. [Elektronisk resurs].* Birmingham : Packt Publishing, 2013., 2013. ISBN: 9781783285808. URL: https://login.e.bibl.liu.se/login?url=https://search-ebscohost-com.e.bibl.liu.se/login.aspx?direct=true&db=cat00115a&AN=lkp.817046&lang=sv&site=eds-live.

[12] Jonathan W Palmer. "Web site usability, design, and performance metrics". In: *Information systems research* 13.2 (2002), pp. 151–167.

[13] Petri Vuorimaa, Markku Laine, Evgenia Litvinova, and Denis Shestakov. "Leveraging declarative languages in web application development". In: *World Wide Web* 19.4 (2016), pp. 519–543. ISSN: 1573-1413. DOI: 10.1007/s11280-015-0339-z. URL: http://dx.doi.org/10.1007/s11280-015-0339-z.

[14] Xiao Wang. "Characterizing and Improving Web Page Load Times". PhD thesis. 2015.

[15] Spyros Xanthopoulos and Stelios Xinogalos. "A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications". In: BCI '13. Thessaloniki, Greece: ACM, 2013, pp. 213–220. ISBN: 978-1-4503-1851-8.