EXAMENSARBETE INOM TEKNIK,
GRUNDNIVÅ, 15 HP

STOCKHOLM, *SVERIGE 2019*

GUI performance metrics framework

Monitoring performance of web clients to improve user experience

ANTON ATTERLÖNN

BENJAMIN HEDBERG

KTH
SKOLAN FÖR ELEKTROTEKNIK OCH DATAVETENSKAP

Abstract

When using graphical user interfaces (GUIs), the main problems that frustrates users are long response times and delays. These problems create a bad impression of the GUI, as well as of the company that created it.

When providing a GUI to users it is important to provide intuition, ease of use and simplicity while still delivering good performance. However, some factors that play a major role regarding the performance aspect is outside the developers' hands, namely the client's internet connection and hardware. Since every client has a different combination of internet connection and hardware, it can be a hassle to satisfy everyone while still providing an intuitive and responsive GUI.

The aim of this study is to find a way to monitor performance of a web GUI, where performance comprises response times and render times, and in doing so, enable the improvement of response times and render times by collecting data that can be analyzed.

A framework that monitors the performance of a web GUI was developed as a proof of concept. The framework collects relevant data regarding performance of the web GUI and stores the data in a database. The stored data can then be manually analyzed by developers to find weak points in the system regarding performance. This is achieved without interfering with the GUI or impacting the user experience negatively.

Keywords - performance; delay; response times; graphical user interface; improvement

Sammanfattning

När man använder grafiska gränssnitt upplevs lång responstid och fördröjning som de främsta problemen. Dessa problem är frustrerande och ger användare en negativ syn på både det grafiska gränssnittet och företaget som skapat det.

Det är viktigt att grafiska gränssnitt är intuitiva, lättanvända och lättfattliga samtidigt som de levererar hög prestanda. Det finns faktorer som påverkar dessa egenskaper som är utanför programmerarnas händer, t.ex. användarens internetuppkoppling och hårdvara. Eftersom varje användare har olika kombinationer av internetuppkoppling och hårdvara är det svårt att tillfredsställa alla och samtidigt tillhandahålla ett intuitivt och responsivt gränssnitt.

Målet med denna studie är att hitta ett sätt att övervaka prestandan av ett grafiskt gränssnitt där begreppet prestanda omfattar responsiviteten och hastigheten av den grafiska renderingen, och genom detta möjliggöra förbättring av responstider och renderingstider.

Ett ramverk som övervakar prestandan av ett grafiskt gränssnitt utvecklades. Ramverket samlar in relevant prestandamässig data om det grafiska gränssnittet och sparar datan i en databas. Datan som sparats kan sedan bli manuellt analyserad av utvecklare för att hitta svagheter i systemets prestanda. Detta uppnås utan att störa det grafiska gränssnittet och utan att ha någon negativ påverkan på användarupplevelsen.

Nyckelord - prestanda; fördröjning; responstider, grafiskt gränssnitt; förbättring

Acknowledgements

We would like to thank our advisors at KTH Royal Institute of Technology; Leif Lindbäck and Fadil Galjic. They have provided valuable feedback throughout the entire development process of the thesis project. We would also like to thank our supervisor at Ericsson, Johan Nyblom, for all the help along the way. Lastly, we would like to thank Ericsson as a whole for giving us the opportunity to do our thesis work with them and for providing a great workplace.

Table of contents

1 Introduction

Websites have to be aesthetically pleasing, responsive and informative to keep visitors interested. Responsiveness play an important role when it comes to user experience. Just a fraction of a second delay between an action and the presentation of the result can be the difference between a user staying on the website or leaving. This sheds light on the importance of taking responsiveness into account when monitoring the performance of a website.

As users are getting accustomed to extremely fast loading speeds of web pages and applications, a delay of less than a second can be noticed by more or less any user. This decreases engagement and increases frustration [1]. As a result, bad responsiveness can be damaging for a brand and undermine the overall health of the brand. Even more so if it already has a fragile consumer affinity. Jakob Nielsen, a leading figure in research-based user experience, talks about human aspirations in one of his articles. Nielsen claims that human beings like to feel in control of their destiny and not be subjugated to a computer's whims [2]. When companies have slow products that make users wait, instead of providing a responsive service, they may seem incompetent. Nielsen also mentions human limitations, such as short term memory. Delay greatly affects overall performance due to this limitation, as there might be too much information that has to be remembered. This type of issue establishes the need for performance metrics, to identify areas where improvements can be made for a better user experience.

1.1 Background

At Ericsson, large amounts of radio data is processed to produce dynamic reports for their end users. The tool that is used for this purpose has a web GUI, which visualizes data for analytical purposes. Ericsson estimates that the size of the data will continue to grow. They have a large number of data points, and the client execution environment is out of their control. Hence, they require a way of monitoring response time of the client to be implemented.

Furthermore, Ericsson wishes that the metrics from the performance monitoring can be used to dynamically adjust the load on the client. For example, if a page takes too long to load its content, the page and its content could be adjusted to decrease the load time. This would result in a GUI that is able to visualize considerable amounts of data without negatively affecting responsiveness.

1.2 Problem

An excellent user experience is something that every company should strive for in their GUIs, in order to achieve an increased customer satisfaction level. This thesis will explore specific areas of performance in GUIs, namely response times and render times. Response times refer to the delay between an action and the response from the system. Render times refer to the time it takes the GUI to draw graphics on the screen.

The studied application will consist of significant amounts of data. If all of this data was to be fetched for every user whenever the application was launched, the user would be presented with massive page load times. Additionally, this would lead to bad performance due to the slow response times.

People enjoy aesthetic websites, but loathe delays of any kind. Hence, a problem arises: how can we improve the user experience in the best way, while still keeping the user interface both responsive and informative? Companies do not always have time or fundings to perform specific and accurate studies to find out their users' experiences. Is there a way to monitor the performance of the website for all users?

The problems that have been discussed in this section of the thesis can be summarized as the following question:

How can performance of a GUI client be monitored?

1.3 Purpose

This thesis aims to find a suitable way of creating a framework that enhances user experience on websites by collecting performance metrics from clients such as render speed, which is the time it takes the website to generate and display graphic objects. This metric, in addition to multiple additional metrics that will be mentioned in this report, can then be used to recognize parts of the application that can be improved.

The experiences from this study could benefit companies by proving, or disproving, the usefulness and effectiveness of using a performance metrics framework to find weak points in their websites, so that they in turn can improve the user experience of their websites.

Furthermore, companies that are in a similar situation as Ericsson, i.e. processing large amounts of data and visualizing it in a web GUI, and want to improve their user experience, can review this thesis to determine if a performance metrics framework is a suitable solution for their company.

Although the data that has to be collected from clients is not by any means personal and private data, this could potentially cause some ethical problems. For example, some users might not be comfortable with the company collecting data during website visits.

1.4 Delimitations

Generally, any framework can be built upon and extended almost infinitely. Since the goal of this project is to find a suitable method to monitor performance of a web GUI, it was decided that this framework would only contain functionality to collect and store relevant data for this purpose.

The project does not include analysis of collected data, nor implementation of the functionality to dynamically update the website in real-time based on the collected performance related data. The result of this project should however lay a foundation and enable the system to be built upon further to include the analyzing and dynamical updating parts as well as other future functionalities.

1.5 Thesis outline

The thesis report is structured as follows:

- Chapter 2 contains necessary theoretical background information as well as technical background needed to understand the rest of the report.
- Chapter 3 contains the research strategies and methods used during this project.
- Chapter 4 presents the result of the project.
- Chapter 5 discusses positive and negative effects of the framework developed as well as possible future improvements.

2 Theoretical Background

To fully understand the content of this thesis, some basic understanding of how websites work behind the scenes is needed. This chapter presents the reader with information that is required to understand the work and research of this thesis. Consequently it contains information about concepts that are fundamental when it comes to developing a functional website, as well as information about certain performance metrics and why they are relevant.

2.1 Performance of web interfaces

This section contains information about measuring points as well as explanations of what they mean and why it is important to monitor these points.

2.1.1 Measuring points

When a user tries to visit a website, a lot of things happen before the user is presented with the webpage. A connection has to be established, the content of the webpage has to load and be painted to the user's screen, etc. The Performance Timing interface from the Navigation Timing API, which is developed by Mozilla, was used to calculate and collect execution times. The chart presented in illustration 1 shows the different timestamps that the API stores.

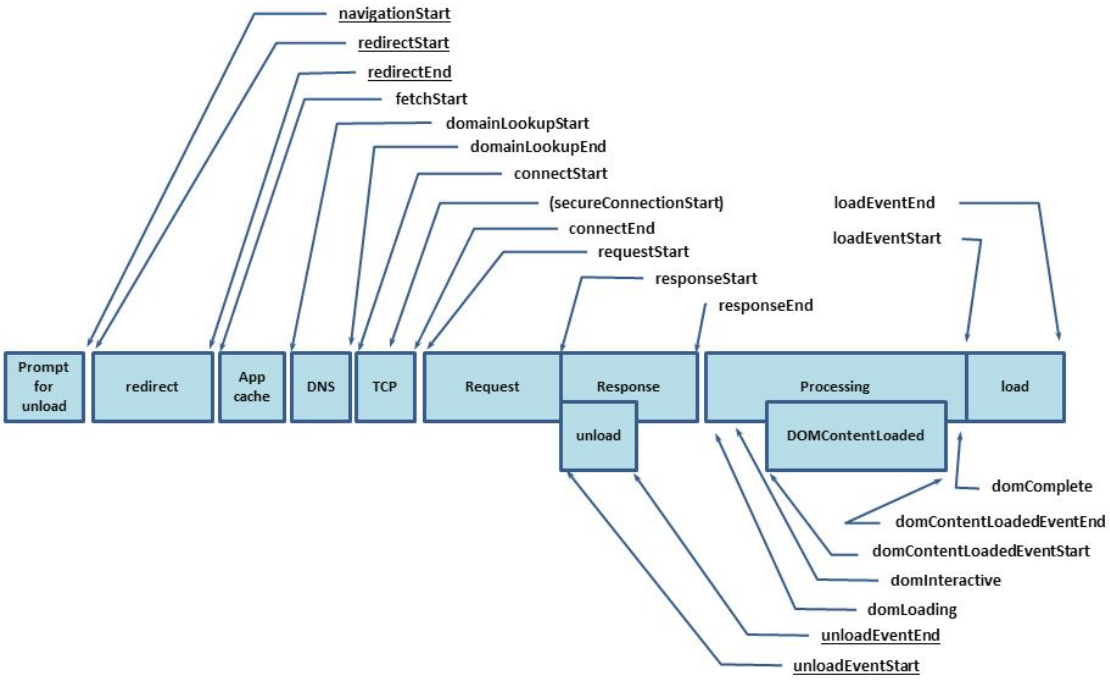

Illustration 1: Timeline and available measure points of the Performance interface.

With these timestamps it is possible to calculate more or less any time taken for the launching of the application. One example of this is to simply calculate the difference between *domainLookupEnd* and *domainLookupStart* to get the time it took for the DNS lookup to complete. This calculation can be seen in illustration 2.

```
dns = (performance.timing.domainLookupEnd - performance.timing.domainLookupStart);
```

Illustration 2: How to calculate DNS lookup time.

Additionally, it is also possible to measure whatever is desired by inserting measure points in the code, and then calculate the difference between these points. For this thesis, calculations of elapsed time between the different timestamps illustrated by the chart in this section was the main approach for retrieving measurements. This is done in the same way as the DNS example which was previously mentioned.

Though it is not part of the Performance Timing interface, render times are also an important aspect of performance of web GUIs. Render times refer to the time it takes the GUI to draw graphics on the screen. To measure render times, the performance.now() [17] method (which returns a time stamp) from the Performance interface [20] is used. One call to the performance.now() is placed at the start of a graphical render and another call to the performance.now() method is placed at the end of a graphical render. The start time is then subtracted from the end time and the result is the total time it took to render the graphics.

Research was done to find suitable execution times to measure and their usefulness regarding performance analysis and improvement. KeyCDN, a leading European CDN provider [24], published a list of what they consider are 14 important website performance metrics one should always be analyzing [3]. The information provided by KeyCDN and its relevance to performance was analyzed. The six execution times that were chosen as a result of this research are presented below.

- **DNS lookup time**
  The duration it takes for the DNS (Domain Name System) provider to translate the domain name into an IP address is called DNS lookup time. [3]

- **Connection time**
  The time between a request and when a connection is established between the user's browser and the origin server of the website is called connection time. [3]

- **Time to first byte**
  The time between the establishment of a connection and when the very first byte of information reaches the user's browser is called time to first byte. The order in which users receive information is important, and some slight alterations in the code can boost this website performance metric. For example, changing some synchronous operations to asynchronous could potentially shorten the time to first byte. [3]

- **Time to start render**
  The time between a user's request and the moment content first starts to appear in their browser is called time to start render. This metric is important to analyze because the sooner a visitor sees content appear, the more likely they are to stay for the rest of the page to load. [3]

- **Time to interact**
  The time between a request and the moment the user is able to click on links, type in text fields or scroll the page is called time to interact. Some elements such as scripts and trackers may continue to load during this period. [3]

- **Time to last byte**
  When the user's browser finally receives each and every byte (including all graphics, e.g. CSS and images) of your website, the last byte time is recorded. The quality of the code and database queries play a big role in this metric. If a large amount of data has to be fetched and displayed, it should be done with great caution as to not slow down the website. [3]

All of the times that were mentioned in this list will henceforth be grouped together, and described by the term "response times". An exception to this is when a specific measurement is discussed.

2.1.2 Levels of responsiveness

As mentioned earlier in the introduction, having a responsive website is critical for a business. It has been proven that a responsive website can boost customers' interest and the business's sales [2].

The Nielsen Norman Group, a UX research firm trusted by leading organizations world-wide, have through elaborate studies found that there are three response-time limits that developers should know about and take into consideration when developing a website [2].

The first limit is around 0.1 seconds response time. A response time of 0.1 seconds gives the user the feeling och an instantaneous response. The user feels that the outcome was caused by him-/herself and not the computer. The user feels in control and this level of responsiveness is essential to give the user the feeling of direct manipulation.

The second limit is around 1 second response time. A response time of 1 second gives the user a sense of delay, they can tell that the computer is generating the outcome. However, they still feel in control of the overall experience and they can still move around freely rather than waiting on the computer. To achieve good navigation on the website, this level of responsiveness is needed.

The third limit is around 10 seconds response time. A response time of 10 seconds still keeps the user's attention. From 1-10 seconds, the user feels at the mercy of the computer and wish it was faster. Though the user can handle it, after 10 seconds they start to think about other things which makes it harder to get their brains back on track once the computer finally responds.

According to Nielsen, A delay of 10 seconds will often make the user leave the site immediately, and even if they stay it will be harder for them to understand what is going on, which makes it less likely that they will succeed in any difficult task.

Nielsen also mentions that just a few seconds delay is enough to give an unpleasant user experience, the user is no longer in control and they are continuously annoyed by having to wait for the computer. If a website presents the user with short delays repeatedly, the user will most likely give up and leave the site, unless they are extremely committed to completing the task. A business with this level of responsiveness could potentially lose a lot of the sales, simply because their site takes a few seconds to load each page.

This again emphasizes the importance of monitoring performance of web GUIs. It is of high importance for companies to know what the flaws in their products are, and where they are, so that they can be addressed and taken care of.

2.1.3 Impact of performance on users

According to a study conducted by Radware[21], a global leader of application security solutions and delivery, slow performance has a negative impact on user's engagement (8% decrease) and level of frustration (26% increase). Moreover, it is damaging for the brand, mainly concerning purchase intent. [1]

This study found that slow pages are the number one issue website users complain about. It was also found that two-thirds of website users expect sites to load in 4 seconds or less[1]. A visual representation of this is shown in illustration 3 below. This is further backed up by Nielsen, who also mentions that users care a lot about speed in interaction design, and that they hate slow websites. [2]

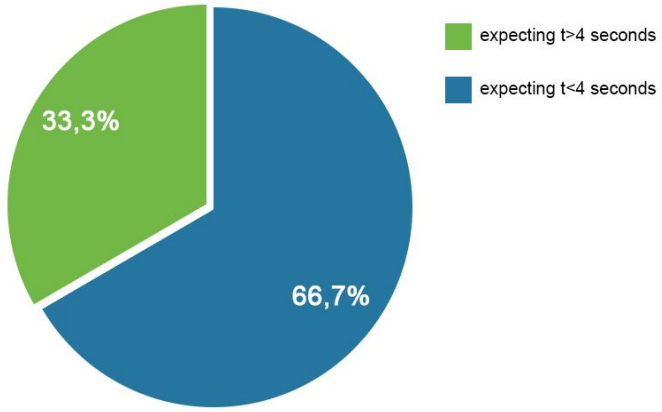

Illustration 3: The distribution of users expecting web pages to load within four seconds and users who don't, where "t" is the load time.

Radware's study also mentions that there are different factors that influence the scale of performance's impact on user experience. This report focuses on two of the factors mentioned in the report. These are the importance of speed regarding relaxed browsing (e.g. social media) versus time dependent browsing (e.g. video games), and the expected performance of different websites. [1]

2.2 Client-side web technologies

This section contains general information about websites and the technologies used on the client-side.

2.2.1 HTML

HTML is an abbreviation of Hyper Text Markup Language and describes the structures of web pages using markup. HTML elements are the building blocks of HTML pages and are represented by tags. These tags are not displayed in the browsers, instead the browsers use these tags to render the content of the page. For example, a "heading" tag tells the browser that the content of this tag should be rendered as a heading. [4]

HTML code is client-side code, which means that the code is ran on the client's device. Therefore the speed of which the HTML code will be ran is determined by the quality of the hardware in the client's device. [4]

As development of the interface is not a part of the project, HTML will not be written by the thesis workers. Any HTML code that is included in the project is provided by Ericsson and is a part of the interface that the framework is being developed for.

2.2.2 CSS

CSS is an abbreviation of Cascading Style Sheets and is used to describe how HTML elements should be displayed. CSS allows adaption of the presentation to different types of devices regardless of screen size. Although CSS is often associated with HTML, it is not dependent of HTML. CSS can be used with any XML-based markup language. [5]

CSS saves a lot of work and time for the developers since it can control multiple web pages all at once and is easily reusable, since the stylesheets can be stored in separate CSS files and imported to any HTML page. This in turn also makes the sites easier to maintain. [5]

As was previously mentioned, development of the interface is not a part of the project, therefore no CSS will be written by the thesis workers. Any CSS code that is included in the project is provided by Ericsson and is a part of the interface that the framework is being developed for.

2.2.3 JavaScript

JavaScript is a cross-platform, object oriented scripting language. It allows for implementation of dynamic things on web pages, mainly to make the web pages interactive with for example complex animations, clickable buttons, popup menus and dynamically updated content. [6]

Whenever a web page does more than just display static information without any animations, you can almost be certain that JavaScript is involved. JavaScript is the third layer of the "layer cake" of standard web technologies, where the other two are HTML and CSS. [6]

Since the framework is being developed in JavaScript, the project will consist predominantly of code written in JavaScript.

2.2.4 React

The framework that is being developed is primarily aimed at measuring performance of interfaces built in React, but can easily be applied to other websites. React is a JavaScript library for building user interfaces and mainly aims to provide speed, simplicity and scalability. [7]

React is maintained by Facebook, Instagram (product of Facebook) and a community of individual developers and corporations. React is open source and under a standard MIT license. [8]

2.2.5 JSX

JSX, or JavaScript XML, is an extension to the JavaScript language syntax. JSX allows developers to write JavaScript with a syntax similar to HTML. React components are typically written using JSX instead of JavaScript. [9]

2.3 Server-side web technologies

This section contains general information about websites and the technologies used on the server side to manage data.

2.3.1 SQL

SQL is an abbreviation of Structured Query Language and let's you access and manipulate databases. SQL became an ANSI (American National Standards Institute) and ISO (International Organization for Standardization) standard as early as 1986 and 1987 respectively. [10]

To build a website that displays data from a database, you need a RDBMS (Relational Database Management System), a server-side scripting language like PHP or ASP, SQL to get the data you want and HTML and CSS to style the page. [10]

2.3.2 PHP

PHP is a widely used open source server-side scripting language that is especially suited for web development and can be embedded into HTML. The main thing that sets PHP apart from a client-side language like JavaScript is that the code is executed on the server rather than on the client. This means that the code is ran on the server and the generated result is sent back to the client. The client would never know what the underlying code was that generated the result. PHP can be used to handle everything that needs to be handled on the server. [11]

In this thesis, PHP is mainly used to execute SQL commands to retrieve data from the database and update the database.

2.4 Related works

No studies covering the specific subject of this thesis project were found. However, some products that were similar to the framework that has been developed for this thesis were identified. One of these products was developed by Google, and is called PageSpeed Insights. [31] This product measures certain performance related metrics, that are also measured by the framework that was developed for this thesis, e.g. *time to interact*. Another similar product is WebPagetest [32], a tool that was developed by AOL and has been open-source since 2008. [30] In addition to the products that have been listed, it is worth mentioning the built in tools in most browsers that are available today, e.g. Mozilla Firefox and Google Chrome.

3 Methods

This chapter presents the research strategies and methods that were used for this study to answer the research question as well as describing the retrieval and storing of data. Additionally, the evaluation methods that were used are explained.

3.1 Impact of delay on users

This section presents the research strategy and literature study that was carried out to gather knowledge needed to answer the problem statement.

3.1.1 Literature study

In order to answer the research questions, some knowledge about how users are affected by the delays of a web client had to be acquired. To acquire this knowledge, a literature study was carried out. Scientific studies on the subject were analyzed and conclusions were drawn based on the results of these studies. The results of the literature study shows that slow performance has a negative impact on user engagement and level of frustration and can be damaging for the brand. Furthermore, the results show that slow pages are the number one issue website users complain about.

3.1.2 Research strategy

A systematic review [26] was the main research strategy for this thesis. There are several advantages of systematic reviews. For example, the risk of the literature being biased is diminished, since a large amount of material is studied when performing this sort of review. [27] Systematic reviews can also lead to strong evidence, given that the results of the studied literature are consistent. On the other hand, it is possible to examine causes of variation if the results are inconsistent. [27]

A systematic review can be summarized in five different steps. [28] These steps, as well as how they have been applied in this specific thesis project, are described below.

The first step is to define a research question. Prior to the formulation of the research question itself, it was decided that performance would be the main area and focus of research. Originally, two additional research questions were included in the thesis project. Since they were evaluated to be unrelated to the main area of research, it was decided to exclude these questions. The final research question for this thesis is *“How can performance of a GUI client be measured?”*.

The second step is to perform a search for data that is relevant to the answer of the research question. For this thesis, articles and reports were collected from sources such as Google Scholar[29]. These publications covered not only possible methods of measuring performance, but also the importance of this field of analysis.

The third step is to extract relevant data from the previous step. In this thesis, the relevance of data was mainly established by answering questions such as *"what is the value of this information?"* or *"how does this information contribute to answering our research question?"*.

The fourth step of the review is to assess the quality of data. Quality assurance of information was primarily carried out by examining who published the information, i.e. if they are established within fields that are relevant to the thesis. An example of a source, that was evaluated to be trustworthy, is Jakob Nielsen from the Nielsen Norman Group.

The fifth and final step of a systematic review is to analyze and combine data. In this thesis different methods of performing performance measurements were compared, and were combined when possible. Data regarding the effect of performance on users was combined to present the value of the thesis.

3.2 Data management

This section explains the methods used to gather, store and retrieve data, and how these methods were applied.

3.2.1 Literature study

A literature study was carried out to figure out the best and most suitable methods to gather, store and retrieve data. Since the framework was only supposed to be a proof of concept, the demands for these methods were that they had to be simple and easy to implement. No demands were made for security.

The literature study resulted in the usage of three languages to monitor the website in terms of responsiveness. The languages chosen and the reasoning behind it are listed below.

JavaScript was chosen as the client-side language to gather data because it is supported on every major browser. Another reason for using JavaScript is because it is so popular and widely used that the majority of web developers have knowledge and experience working with this language. [12]

PHP was chosen as the server side language to handle the connection to the database mainly because it is by far the most popular server side scripting language. Just like with JavaScript, this means that the majority of all web developers have knowledge and experience working with this language. [13]

As both of the thesis writers had previously worked in several projects with SQL, and since it was widely used at Ericsson, the database research was focused on this language. SQL is a powerful language that is optimised for large amounts of table rows, while still maintaining performance [14]. Additionally, SQL is fast for searching and querying data over a single table [14], which is a desirable feature in this project as there is only one table. As SQL has been around for a very long time, developers are used to working with this language. This would mean that the framework that has been developed is easier to use for developers at Ericsson.

3.2.2 Collection of data

Collection of data is mainly handled by the PerformanceTiming [15] interface from the Navigation Timing API [16] developed by Mozilla. The Navigation Timing API provides data that can be used to measure the performance of a web site. Mozilla claims that this API can be much more accurate and reliable than JavaScript-based libraries that have historically been used to collect similar information. The PerformanceTiming interface from the Navigation Timing API contains properties that supply performance timing information for different events that occur during the use and loading of the current page.

The framework also makes use of manual timing to get render times by using the performance.now() [17] method from the Performance interface [20]. This interface was also developed by Mozilla.

To restate and answer the research question for this thesis, the performance of the GUI can be monitored and analyzed by continuously measuring critical time periods that are expected to affect the responsiveness of a GUI. Improvement or deterioration of the responsiveness of the GUI can easily be detected by comparing response times before and after updates to the GUI or backend. The Navigation Timing API was chosen as it uses data that is always collected when browsing the web, which means that the data is always easily accessible. This approach won't have a significant enough effect on the overall performance for the user to take notice of it, since the stored data is made up of timestamps which doesn’t contain large amounts of data.

3.3 Design and implementation of a prototype

This section explains the methods used to design and implement the framework. This includes evaluations of the framework, as well as the necessary studies to decide the most optimal solution for this problem.

3.3.1 Literature study

A literature study was carried out to find the most suitable design methods for this framework in addition to gathering more knowledge about the programming languages and techniques which would be used.

Codecademy[25] was the main source of information regarding the necessary knowledge to work with Javascript and React. Several guides which covered these topics were followed before any design decisions were made, as to ensure that there was no "happy hacking" approach. This decision was made to prevent early implementations with poor design, which would be a huge waste of time.

3.3.2 Performance measurements' impact on user interface

Since the framework will run simultaneously while the user is using the website, the framework cannot be allowed to hurt the user experience in any way. Therefore it is important to evaluate the performance of the framework to ensure that it does not harm the user experience.

The framework uses data that is already stored automatically which means that no time is used to collect this. Although, when the framework measures render times after the initial load of the page, e.g. when a user navigates the web page, the performance.now()[17] function is used. To investigate to which degree the usage of performance.now() affected the performance of the GUI, manual testings were made. The GUI was reloaded several times and the time it took the GUI to reload was noted each time. The same test was then made with performance.now() present in the code. The notes of the reload times were then compared. The time it takes for performance.now() to get a value was found to be, in most cases, less than a millisecond. It was therefore decided that the effect the usage of performance.now() has on the performance of the GUI was negligible.

The rest of the framework largely consists of AJAX calls which are asynchronous, which means that they will run at the same time as the main code. The actual impact of the framework on the speed of the system is in the case of this thesis negligible.

Furthermore, the server-side of the framework is written in PHP and SQL and will have no impact on the client or the measurements, as the actual values are measured on the client.

3.3.3 Evaluation methods

Continuous evaluations were made throughout the project to ensure that the framework achieved its purpose. An iterative process of designing, re-designing and implementing functionality was used. Weekly meetings with the supervisor at Ericsson, where the framework was presented and the changes from the week before were explained ensured that the framework achieved its purpose of collecting, storing and retrieving data in a convenient way.

4 Result

This chapter describes the results of the project and goes in depth about solutions used.

4.1 Summary

The result of this project is a lightweight, easily adaptable framework, which is able to monitor performance for any web GUI. Due to requiring only a single import statement, using the framework is effortless. The performance related data that is collected using the Performance interface[20] in combination with the performance.now() [17] method is stored in a MySQL database and can be used for data analysis, which in turn can be used to improve the interface performance-wise as well as improve the user experience overall. Illustration 4 below shows how easy and effortless it is to run the framework on a web GUI.

```
import * as test from './benton.js';
```

Illustration 4: All that has to be done to run the framework on your GUI is to add this line.

4.2 Project structure

The project is built around three layers. The first layer handles measuring and collecting of data. The second layer handles storing of data. Finally, the third layer handles fetching of the data that was stored in the previous layer. It is worth noting that the third layer is mainly for testing purposes and will most likely not be kept in the final product when Ericsson takes over. The whole process is shown in the sequence diagram presented in illustration 5 below.

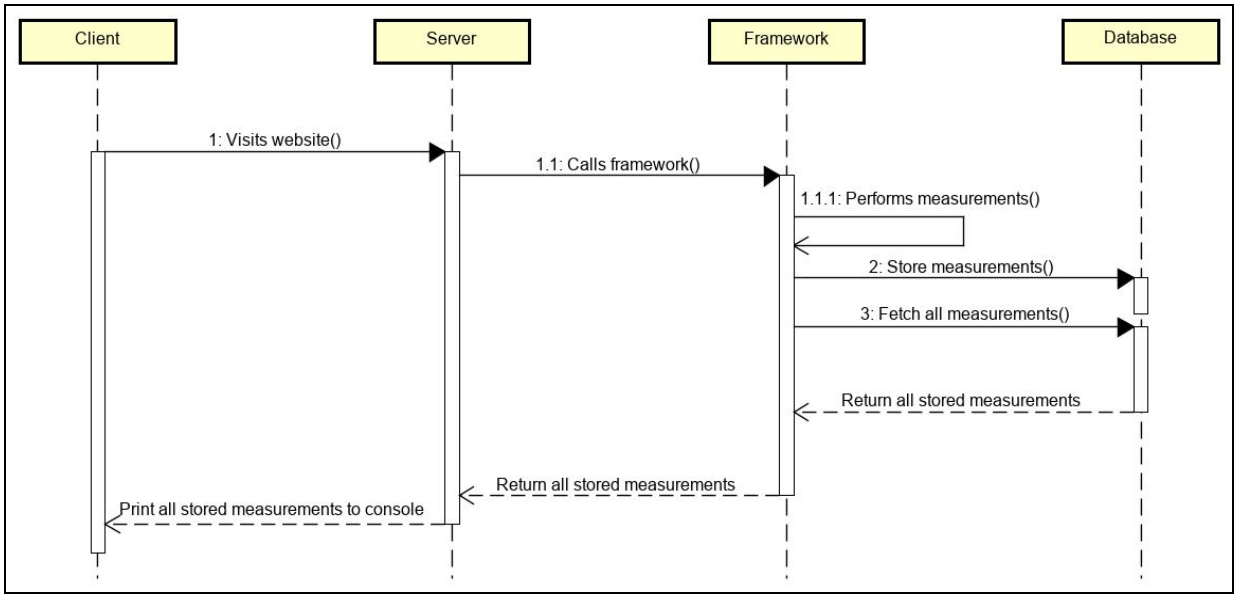

Illustration 5: Sequence diagram showing the interaction between the client, server, framework and database

The sequence diagram in illustration 5 shows the entire flow of the framework. Action 1 to 1.1.1 shows the first layer, collecting of data. In this part of the flow a Client visits a website that has implemented the framework. This triggers the collection of measurements. Secondly, action 2 shows the second layer, storing of data. This part is more or less self explanatory. The measurements that were performed in action 1.1.1 are stored in the database. Finally, action 3 and the following return statements show the third layer, fetching of data. This part of the flow retrieves the measurements that are stored in the database, and prints them to the client's console.

4.2.1 Collecting data

This is the primary part of the framework. The framework collects data regarding response times. The collected response times are: DNS lookup time, connection time, time to first byte, time to start render, time to interact, and time to last byte. The different response times that are collected were thoroughly described in subsection 2.1.1.

Response times are measured in the background by the Performance Timing interface [15] from the navigation Timing API[16]. Illustration 6 demonstrates the code that was written for this process.

```
//Store measurements in variables for later use
var setValuesPromise = new Promise(function(resolve, reject){

  ttsr = (performance.timing.domLoading - performance.timing.requestStart);
  ttfb = (performance.timing.responseStart - performance.timing.connectEnd);
  ttlb = (performance.timing.responseEnd - performance.timing.connectEnd);
  ct = (performance.timing.connectEnd - performance.timing.connectStart);
  dns = (performance.timing.domainLookupEnd - performance.timing.domainLookupStart);
  curr = window.location.href;
  jsonData = {};

  //Makes sure that domInteractive and domComplete exists before storing their values
  document.onreadystatechange = function () {
    if (document.readyState === "interactive") {
      tti = (performance.timing.domInteractive - performance.timing.requestStart);
    }
    if (document.readyState === "complete") {
      ttc = (performance.timing.domComplete - performance.timing.requestStart);
      resolve("done");
    }
  }
});
```

Illustration 6: The measurement of the response times mentioned in chapter 2.1.1

Illustration 6 shows how simple it is to collect the relevant response times. Since all measuring points are stored in the background, getting the values of the performance metrics is as simple as subtracting the starting point of a performance metric from the end time. When it comes to performance metrics regarding the DOM [23] (Document Object Model), it gets a little trickier. Since the framework is ran simultaneously as the GUI, it will try to get the measuring points values for the performance metrics *time to interact* and *time to complete* as soon as the framework is called. This is an issue because it takes the DOM some time to complete. The framework is told only to retrieve the values from the measuring points regarding the performance metrics when they are ready to be retrieved. Since all these operations are asynchronous, a Promise object [18], which represents the eventual completion (or failure) of an asynchronous operation, was used to verify that the measurements were completed before these values could be added to a JSON object. This is shown in the first line of code in both illustration 6 and 7.

```
//Store all connection related data in jsonData and send it to database
setValuesPromise.then(function(result){
  jsonData.TimeToStartRender = ttsr;
  jsonData.TimeToFirstByte = ttfb;
  jsonData.TimeToLastByte = ttlb;
  jsonData.ConnectionTime = ct;
  jsonData.DNSLookupTime = dns;
  jsonData.CurrentPage = curr;
  jsonData.TimeToInteractive = tti;
  jsonData.TimeToComplete = ttc;
  jsonData.Date = null;
  console.log(jsonData);
  storeMeasurements(jsonData);
  getMeasurements();
});
```

Illustration 7: The values shown in illustration 6 are added to a JSON object called jsonData

In addition to the measurements, some other values are added to the JSON object. The currentPage value is used to show which page of the web application the measurement was performed on. Furthermore, the Date value shows when the measurement was done. The date value is set when the measurements are added to the database with PHP, using the Date function [19]. The result from this function is seen as the date value of the JSON object in Illustration 9.

Render times are measured manually using the performance.now()[17] method from the Performance API, developed by Mozilla in JavaScript, and are also stored in JSON objects along with other useful data.

4.2.2 Storing data

The project made use of a simple MySQL database to store all data that the framework produces. The collected data are stored in JSON objects, which in turn are sent to the database using AJAX post requests. The AJAX requests are handled by the server, using PHP, which sends the data to the database using SQL calls. The code that was used for this process is shown in illustration 8.

```
//Sends the measurement data collected in setValuesPromise to the database through PHP
export function storeMeasurements(jsonob){
  $.ajax({
    url: 'http://localhost/savems.php',
    type: 'post',
    data: {
      "jsonData": JSON.stringify(jsonob)
    },
    success: function(data){
      console.log(JSON.parse(data));
    }
  });
}
```

Illustration 8: The storing of the JSON object using AJAX containing all the measurements

Once the data has been added to the database the entries looks as the following illustration.

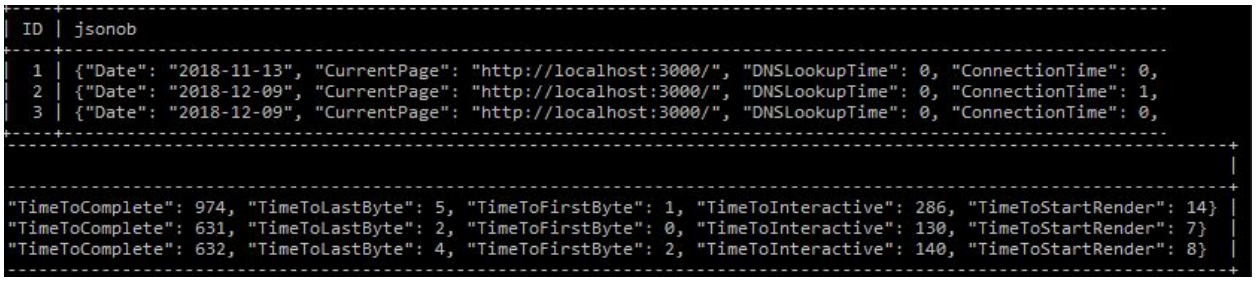

Illustration 9: Three entries from the measurement database

The entries in the database represent the response times for specific web pages at a specific time, making it easy to find weak points and differences between versions of the GUI.

4.2.3 Fetching data

Fetching of data was not part of the project description, but was added to prove that the solution enables the stored data to be easily accessible and fetched. Only a simple AJAX get request function was implemented to prove this point.

The data is fetched by sending an AJAX get request, which is received by the server (PHP). The server sends an SQL call to the MySQL server and retrieves the data. The data is then sent back to the client as the response in the AJAX get request. The code that was used to fetch the measurements from the database are shown in illustration 10.

```
//Get the measurement data stored in the database through PHP
export function getMeasurements() {
  $.ajax({
    url: 'http://localhost/getms.php',
    type: 'get',
    success: function(data){
      console.log(JSON.parse(data));
    }
  });
}
```

Illustration 10: Fetching the measurements from the database using AJAX calls

The function shown in illustration 10 prints all of the measurements that are stored in the database in the browser's console. An example of the result of this procedure is shown in illustration 11. In addition to all of the entries from the database, the measurement that was just performed is printed explicitly, as to get a clearer view of the most recent measurement.

Illustration 11: The result of the getMeasurements() function

5 Discussion

This chapter of the thesis discusses the conclusions that were drawn. There were several positive effects that were noted, and some areas that could be improved. Additionally, possible additions to the framework and thesis are discussed at the end of the chapter.

5.1 Positive effects

The framework is able to monitor response times and render times for the GUI that was developed by Ericsson. The framework is also easily adapted to other GUIs conveniently since it only requires a single import statement to implement. The automatic performance metric collection removes the necessity for manual performance measurements and analysis, since there always will be data that can be analyzed to improve the system's performance as well as the user experience. This automatic process can potentially save companies massive amount of time.

Due to requiring only a single import statement the implementation of the framework is effortless. Moreover, the framework is lightweight, making it comprehensible. This results in a pleasant experience running the framework, as well as updating it. To back up these claims, the supervisor at Ericsson was asked to evaluate the overall quality of the framework. In the supervisor's opinion, these statements are correct.

Database calls by the framework are performed asynchronously. This means that the framework has such small impact on the overall performance of the website it is ran on, that the impact is negligible. In addition, the framework uses measuring points which are collected and retrieved by the Performance Timing interface, from the navigation Timing API, in the background (since it is ran asynchronously). As was mentioned earlier in this report, the impact these measurements and calculations have on the performance of the system are also so small that they are negligible.

5.2 Drawbacks

It was not investigated whether or not the storing of data was handled in the best way. The first solution used was to store all measuring points in different columns in a table in the database. The final solution uses one table with two columns, one for each collection of measurements in its own JSON object, and one for IDs to identify the measurements which created a more clear database layout. Better solutions were not investigated since this project was supposed to be a proof of concept and not a final solution.

The system uses JSON objects to store data in its current version. It was not investigated whether or not the storing of data was handled in the best way. Different possible solutions should be examined to decide if they would improve the quality of the project.

Furthermore, the project was mainly created for testing purposes. For a live version a different database system might be preferable, with features such as greater security.

It was discussed whether or not information about specific clients should be included in the data that is collected, for example tying measurements to unique clients. With this approach it would be easier to adjust the GUI to give a more personalized experience. Although, this was decided against since it would present an issue with the integrity of users and possibly violate GDPR [22] if not handled correctly and with extreme caution.

5.3 Future work

This project only includes gathering and storing of necessary data. Extending the project to include analysis of data and real time updates of the website to provide a better user experience in terms of delays and response times would be the next step to improve this framework.

The database used for this project is a simple MySQL database. This approach is suitable for testing environments since MySQL databases are simple and easy to use. When used in live environments however, a more advanced option is advisable. No thought was put into the security of the choice of database for this project. In live environments, security is an extremely important aspect of any product.

Fetching of data was mainly added to this project as a proof that the data can actually be fetched. Only the function to fetch all JSON objects in the database is implemented in this version. When implemented in a live environment, more flexibility of fetching is advisable. When analysing data it is preferable to be able to analyse specific parts of the data, e.g analysing a specific measure point in a specific time span. An example of this could be if a new patch is released for the website. Then it would most likely be desired to be able to analyse the effect the patch had on various response times compared to the values before the patch was released.

5.4 Problem statement answered

The main purpose of this project is to answer the problem statement presented in the problem section of the report. This section of the report covers the conclusions and answers regarding the problem statement.

5.4.1 How can performance of a GUI client be monitored?

The approach that was used to monitor performance in terms of responsiveness of the GUI is to measure response times. Finding and eliminating bottlenecks on GUIs is important as this leads to a significantly better user experience (see ch 2.1.2 for detailed explanations on how delays affect users). Finding bottlenecks manually can be a hassle and in bigger projects almost impossible to do. By continuously measuring response times, bottlenecks caused by slow

internet connection, a slow server or slow hardware on the client side can be found. This information can be analyzed and the GUI can be updated accordingly to accommodate bad hardware.

5.5 Value of the project

Ericsson's website contains large amounts of data for analysis purposes, this information has to be readily at hand for the users working with the provided data. Since the website's primary users are not relaxed browsers nor heavily time-dependent browsers, it is likely that performance in speed has a important impact on the overall user experience. Additionally, it is worth mentioning that Ericsson is an IT-company. Therefore, the expectations regarding their website's performance related to speed are considerably high.

Seeing that performance has an influence on user's levels of frustration and engagement, in addition to potentially being harmful for the brand, it is a vital aspect of an optimal user experience. Thus, making this thesis project valuable and meaningful when attempting to optimize user experience.

References

[1] Mobile stress: Slower web pages lead to increased user frustration and lower engagement [Cited February 17, 2019]
https://www.immagic.com/eLibrary/ARCHIVES/GENERAL/RDWR_IL/Radware_Mobile_Web_Stress_Report_2013.pdf

[2] Website Response Times [Cited April 9, 2018]
https://www.nngroup.com/articles/website-response-times/

[3] 14 website performance metrics you should be analyzing [Cited April 20, 2018]
https://www.keycdn.com/blog/website-performance-metrics

[4] HTML information [Cited April 23, 2018]
https://www.w3.org/html/

[5] HTML & CSS [Cited April 23, 2018]
https://www.w3.org/standards/webdesign/htmlcss

[6] What is javascript? [Cited April 23, 2018]
https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

[7] React (JavaScript library) [Cited April 23, 2018]
https://en.wikipedia.org/wiki/React_(JavaScript_library)

[8] What is React? [Cited May 5, 2018]
https://reactjs.org/tutorial/tutorial.html#what-is-react

[9] JSX [Cited May 5, 2018]
https://en.wikipedia.org/wiki/React_(JavaScript_library)#JSX

[10] Introduction to SQL [Cited May 10, 2018]
https://www.w3schools.com/sql/sql_intro.asp

[11] What is PHP? [Cited May 15, 2018]
http://php.net/manual/en/intro-whatis.php

[12] JavaScript version browser support [Cited January 20, 2019]
https://www.w3schools.com/js/js_versions.asp

[13] Best programming language for server-side development [Cited January 20, 2019]
https://twm.me/best-programming-languages-and-frameworks-for-server-side-web-development/

[14] To SQL or not to SQL [Cited January 20, 2019]
https://capgemini.github.io/design/sql-vs-nosql/

[15] PerformanceTiming [Cited June 15 , 2018]
https://developer.mozilla.org/en-US/docs/Web/API/PerformanceTiming

[16] Navigation Timing API [Cited June 15, 2018]
https://developer.mozilla.org/en-US/docs/Web/API/Navigation_timing_API

[17] performance.now() [Cited June 15, 2018]
https://developer.mozilla.org/en-US/docs/Web/API/Performance/now

[18] JavaScript Promise [Cited December 30, 2018]
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise#Description

[19] PHP Date function [Cited December 30, 2018]
http://php.net/manual/en/function.date.php

[20] Performance Interface [Cited June 15, 2018]
https://developer.mozilla.org/en-US/docs/Web/API/Performance

[21] Radware [Cited February 15, 2019]
https://www.radware.com/

[22] GDPR [Cited February 17, 2019]
https://www.datainspektionen.se/lagar--regler/dataskyddsforordningen/

[23] DOM [Cited February 15, 2019]
https://www.w3.org/DOM/

[24] KeyCDN [Cited February 15, 2019]
https://www.keycdn.com/about

[25] Codecademy [Cited February 15, 2019]
https://www.codecademy.com/about

[26] Kitchenham, B. and Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. [Online] p.3. [Cited February 28, 2019]
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.471

[27] Kitchenham, B. and Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. [Online] p.4. [Cited February 28, 2019]
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.471

[28] Systematic Review [Cited February 28, 2019]
https://en.wikipedia.org/wiki/Systematic_review#Stages

[29] Google Scholar [Cited February 28, 2019]
https://scholar.google.se/

[30] WebpageTest about [Cited February 28, 2019]
https://www.webpagetest.org/about

[31] PageSpeed Insights [Cited February 28, 2019]
https://developers.google.com/speed/pagespeed/insights/

[32] WebpageTest [Cited February 28, 2019]
https://www.webpagetest.org

Illustrations:

[1] Performance interface [Cited December 30, 2018]
https://www.w3.org/TR/navigation-timing/#processing-model