Solent University

**Faculty of Business, Law and Digital Technologies**

**BSc (Hons) Software Engineering**

**Academic Year 2020-2021**

**João Leite**

**Synthetic Transaction Monitoring**

Progress Report

COM616 AE1

Tutor: Craig Gallen

Date of submission: 02/2021

**Table of Contents**

[1. Introduction 5](#_Toc65228807)

[2. Project Evaluation (Methods) 6](#_Toc65228808)

[2.1. Data Capture 6](#_Toc65228809)

[2.2. Storing Data 7](#_Toc65228810)

[2.3. Data Visualization 8](#_Toc65228811)

[2.4. Problem Detection 10](#_Toc65228812)

[2.5. Performance Classifier 12](#_Toc65228813)

[3. Project Progress 13](#_Toc65228814)

[4. Project Management 15](#_Toc65228815)

[5. What next? 16](#_Toc65228816)

[10. Reference list 17](#_Toc65228817)

[11. Bibliography 19](#_Toc65228818)

**List of Figures**

[Figure 1 - Data capture diagram. 7](#_Toc65228826)

[Figure 2 - Storing data diagram. 8](#_Toc65228827)

[Figure 3 - Example Bar Graph for log quantity with Kibana 9](#_Toc65228828)

[Figure 4 - Data visualization diagram 10](#_Toc65228829)

[Figure 5 - Problem detection diagram 11](#_Toc65228830)

[Figure 6 - Performance classifier diagram 12](#_Toc65228831)

[Figure 7 - Overall system diagram 14](#_Toc65228832)

**Glossary**

API : Application Programming Interface 8

FCP : First Contentful Paint 9

HAR : HTTP Archive 6

HTTP : Hypertext Transfer Protocol 6

JRE : Java Runtime Environment 13

JSON : JavaScript Object Notation 14

KQI : Key Quality Indicators 6

NDJson : Newline Delimited JSON 8

RUM : Real User Monitoring 5

STM : Synthethic Transaction Monitoring 5

TBT : Total Blocking Time 9

TTI : Time to Interactive 9

# Introduction

With the internet now reaching more people than ever, businesses have a growing demand to expand online, and in such a competitive market as we are in today, their platform should be optimized for ease of use and consistent performance across devices. The rapid changes in technologies and all the moving parts of complex websites, make it so companies now have the need to monitor and analyse their platform behaviour to improve user experience.

In web terms, a good user experience is a quick loading, easy to use website with a good design. First impressions also have a huge impact here, as people often determine if they are choosing a company based on it. Analysis of user data shows that most will not return to a website where they had a bad experience, being it because of performance or aesthetics. Research conducted in 2017 discovered that the user bounce rate for pages that load between 1 and 5 seconds is 90% (An 2018), this means that, for a lot of businesses, there is a direct correlation between website load time and profitability. A better-optimized website increases business reach, improves customer satisfaction, minimizes the cost of system failures, and allows you to prevent them by identifying inefficient code (Matam and Jain 2018).

There are two main ways to perform monitoring on websites, RUM (passive) and STM (active). Although both can be used for performance analysis, the former focuses on analysing consumer behaviour with the page, giving us a large range of data from real users (E.g., DNS resolution, page render time, location, etc.). The latter is more performance-oriented, allowing for more flexibility by creating synthetic users to collect data, which can be done at any time at the desired scale. I will be focusing on using STM to conduct experiments on how it can be a viable tool to detect problems real users would face when using a website.

# Project Evaluation (Methods)

In order to understand how a project like this can be developed, and assess its feasibility, I am gathering a range of resources from papers, journals, patents and websites on web monitoring techniques and analysis as well as the official documentation of the technologies used.

I am proposing the creation of a test harness for web monitoring with the goal of accessing how we can detect problems real users would encounter using STM. The test harness will be running against websites with a variety of failure scenarios and results will be compared with known KQI for user experience.

The project can be divided into 5 parts that work collaboratively with the goal of achieving the final solution.

## Data Capture

To replicate user interaction with a page I will be using Selenium, which is a testing framework with three main components, the main one I will be using is its web driver which can be utilized to run test cases on a browser. In this context, a test case is a set of instructions that will be executed in a browser to replicate real user behaviour (P. Ramya et al. 2017). A very popular alternative would be JMeter, which is heavily focused on performance testing of applications, networks, and protocols. The main reason to not use it was that it does not recreate the browser experience but instead it creates and sends requests to a server, limiting the range of data that could be compared to real user results.

But Selenium does not save the data, for that purpose I will be using a library called BrowserMob Proxy which allows me to capture and export HTTP content from the test case to a HAR file (Lightbody 2021). A HAR file is a JSON based format, created to allow flexibility to pre-process its HTTP archives, they are composed of general data about the browser and the user as well as all the information from requests and responses made to the page (E.g., cookies, headers, response status, content, etc.). (Odvarko 2021)

Gathering this type of file means I can only perform STM, to implement RUM we would need access to the hosting server, where we could collect data from real users to get a further insight into performance and user statistics. A couple of RUM limitations that can be addressed with the use of an STM system instead are, the lack of benchmarking against competitors nor the possibility to test the website in a production environment (before launch or update).

Graphical user interface, application, PowerPoint

Description automatically generated

Figure 1 - Data capture diagram.

## Storing Data

For the remaining of the project, the main technologies used are two tools from the Elastic Stack, developed by Elastic, which are ElasticSearch and Kibana. ElasticSearch is a distributed, scalable, real-time search and analytics engine, used to search, analyse, and explore data (Gormley and Tong 2015). It was designed to work seamlessly with Kibana, a tool that provides an easy-to-use interface for real-time data analysis and visualization, it provides a lot of extra functionalities and a friendly way to navigate the data stored in ElasticSearch (Gupta 2015).

Due to the amount of information that is retained in a HAR file, there is a need to pre-process the data before sending it to ElasticSearch. To achieve this, I will be using a JSON processor called JSTL, it can be used to query values, as well as filter data and transform it between JSON formats (Schibsted Media Group 2021). One of the formats best supported by Elastic is NDJSON, so I will be using JSTL to break down the HAR into smaller components and remove useless information.

Having the pre-processed data, I need to be able to send it programmatically to ElasticSearch, this is normally done using search APIs. There are some options, including the several search APIs provided by Elastic (Elastic 2021) but for this project, I have been looking at a Java HTTP Rest client called Jest, its goal is to make the communication between an application and Elastic simpler by providing a fluent API with easy to work interfaces (Pratt 2019).

Graphical user interface, application, PowerPoint

Description automatically generated

Figure 2 - Storing data diagram.

## Data Visualization

As mentioned previously, Kibana provides an easy way to interact and visualize the data in ElasticSearch, it allows you to create a dashboard where you can have charts, diagrams and graphs representing groups of data that can be used to get a further understanding of the website performance in a chronological view.

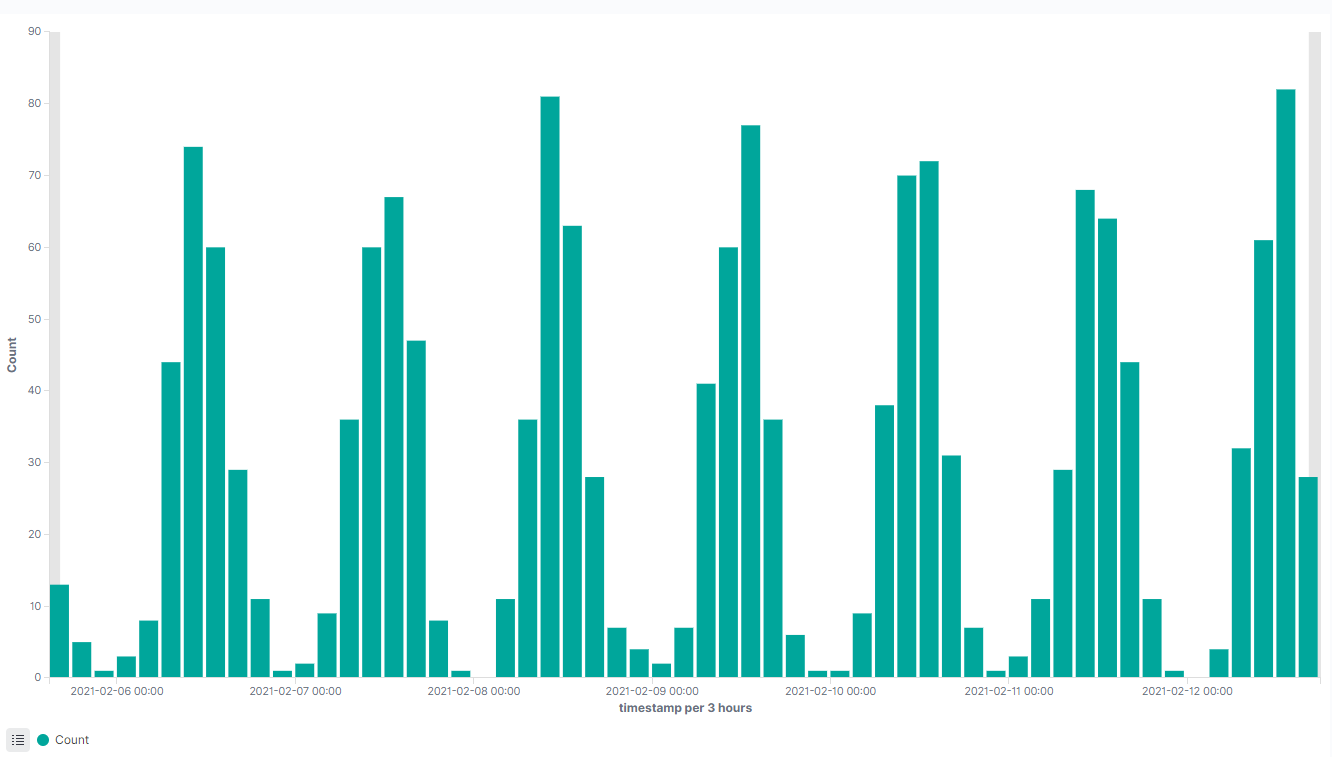


Figure 3 - Example Bar Graph for log quantity with Kibana

From analysing existing web monitoring software (I.e., Google Lighthouse, SemaText, MachMetrics and BlazeMeter) and the sample weblogs project on Kibana I am inclined to get the following fields to the dashboard:

* FCP – how long until the first content is rendered (text or image).
* TTI – how long until the user is able to fully interact with the page, starts after FCP and stopping when there is an uninterrupted period without long tasks (long tasks are defined to be Javascript that takes longer than 50ms to execute) (W3C 2017)
* TBT – the sum of the time of long tasks, blocking time will be the remaining time of tasks that take over 50ms (WebDev 2019).
* Total Page Load Time – how long until all resources are loaded.
* Error Rate – the percentage of failed requests to successful ones.
* Statistics of Responses Time – min, max, average, etc, of times of requests.
* Uptime – the percentage of the time where the website is loaded correctly (Uptrends 2021).

With the use of Kibana, we can represent chronological data with real-time updates, in which you can easily change the time intervals for the dashboard contents. For our case, it would be more beneficial to be looking for both individual and monthly results in order to try finding patterns of broken components or constant bad performance.

Graphical user interface, application, PowerPoint

Description automatically generated

Figure 4 - Data visualization diagram

## Problem Detection

In the chosen metrics we can often find drastic changes in values that affect the website performance negatively. Each metric will be compared to the findings of the open-source project HTTP Archive (<https://httparchive.org/>), as well as other sources (I.e., An 2018, Calvano 2018), which focus on storing historical web performance information from websites. In the table below I have selected values that will help define the success metrics of the project.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Metric** | **Type** | **From** | **To** | **Median(s)** |
| **FCP** | Desktop | 15 December 2016 | 01 January 2021 | 2.50 |
| **FCP** | Mobile | 15 December 2016 | 01 January 2021 | 5.20 |
| **TTI** | Mobile | 01 July 2017 | 01 May 2019 | 9.30 |
| **DOMContentLoaded[[1]](#footnote-1)** | Desktop | 01 August 2013 | 01 January 2021 | 3.10 |
| **DOMContentLoaded** | Mobile | 15 May 2016 | 01 January 2021 | 7.20 |
| **onLoad[[2]](#footnote-2)** | Desktop | 15 November 2010 | 01 December 2020 | 6.80 |
| **onLoad** | Mobile | 16 May 2011 | 01 December 2020 | 17.60 |

Table 1 – Median Values metrics (HTTP Archive 2021)

Problems should be easily recognized while monitoring with Kibana. If I manage to create the dashboard with the metrics proposed in the previous chapter, broken or non-efficient components will most of the time be identifiable through peaks in charts. I will be focusing on the time to load of different components and the size of the website which has a direct correlation with the website loading time (Munyaradzi et al. 2013).

Graphical user interface, application, PowerPoint

Description automatically generated

Figure 5 - Problem detection diagram

## Performance Classifier

Finally, I would like to build a test that would retrieve all the data gathered for a specific website and classify its performance according to the current internet standards. They could be classified as good, average, or bad by comparing statistics from recent studies to the statistics generated by the saved data.

To perform the classification, I have been looking at how I can build tests using Apache Groovy, which is a dynamic scripting language that gets compiled into Java bytecode, meaning it can run on any platform with a JRE. Besides that, the scrips can be sent to ElasticSearch, giving you more flexibility when it comes to querying the datastore.

Graphical user interface, application, PowerPoint

Description automatically generated

Figure 6 - Performance classifier diagram

# Project Progress

By looking into the tools proposed, I was able to get a better feel for the project scope, which also cleared a lot of miss conceptions I had regarding web monitoring and how websites communicate with the different stakeholders.

I have been trying to use the tools, in the order of the life cycle described in the methods chapter, beginning with the development of a test for gathering the data from a website. By using the Selenium toolset and a respective WebDriver in Java I am able to start a browser instance, on a specified URL, this allows us to control the browser and run different tests as if a real user were using it. Using BrowserMob Proxy I can capture and export HTTP content and export it as a HAR file.

My next step was to get a feel of the Elastic Stack tools and understanding how and which data can be used. I now know that Elastic works well with NDJSON data, but it requires mappings for the objects we want to use. I have managed to insert sample JSON scripts into ElasticSearch with the use of the console (Developer Tools) inside the Kibana interface and using CURL commands through the Windows console. Of course, this process will have to be automatically done every time we gather new data, this will be done with JEST that allows requests to the ElasticSearch datastore.

As we will be dealing with large data files and neither ElasticSearch nor Kibana support the direct insertion of complex JSON, I will need to deconstruct them and make several calls sending different parts of the archive. There is also a lot of unnecessary data in the files, so I will need to filter with JSTL for useful data like the request content types, status code, load time, etc.

With some data on ElasticSearch, I have used Kibana to create a dashboard and visualize and compare the data chronologically.

For the performance classifier and problem detection, I am researching how we can run Groovy scripts in ElasticSearch programmatically. Both would work by looking for radical changes in values, or the detection of patterns for the metrics researched.

Diagram

Description automatically generated

Figure 7 - Overall system diagram

# Project Management

To make sure I am on track with the project deliveries I will be using Agile techniques to help manage my time more efficiently, although not all concepts can be applied to a single person team there are still some that I can take from the methodology. Having used it in past, highlighted the importance of following a set of rules when developing a project. The agile key concepts (Beck et al. 2001) that I intend on applying are:

* The state of the project artefact as the main measure of progress.
* Constant software delivery, in my case regular updates of the most updated version of the project to GitHub.
* And the flexibility to change the main plan by reflecting on the weekly progress done.

The project minimum viable product would involve the development of a Java application that consistently gathers data through HAR files with Selenium, using BrowserMob Proxy to pre-processes and save it to ElasticSearch. Finally, it should allow for the dynamic visualization of the data with a Kibana dashboard. Building from there I would like to add problem detection and performance classification.

Also, to stay on schedule and have a working software as soon as possible I will be using GitHub and assigning the development of the different parts of the program to release versions. During the implementation phase, I will also use Git Project as my project management software.

All notes and thoughts are being registered in a logbook and I am keeping track of the weekly meetings with the tutor and the feedback given.

# What next?

Going forward I still need to expand my knowledge about the tools in question. Then again, I will try to work my way up the 5 steps, my main goal is to first develop a minimum viable product which will consist of a system to record, save, and visualise data, even if manually. To achieve this, I need to be able to insert formatted data from the HAR files into ElasticSearch and view it dynamically in Kibana. If for some reason I get stuck with a technology I will be looking for alternative software to help me developed the solution.

My first step from now is to learn how to pre-process the data using JSTL and try saving it to the database, which will probably require an appropriate mapping. With the data and mappings ready I will be looking at how I can use the Jest library to send it to ElasticSearch. From there I need to create a dashboard in Kibana with the metrics defined.

This is my plan to achieve my minimum viable product, from here on the focus would be to automate this process by deploying the solution in Docker containers in a cloud environment like Microsoft Azure, and creating a script that would run a Selenium test every defined time interval (I.e., 5 minutes).

The last part would be to prepare scripts that would detect problems (components affecting performance) by querying ElasticSearch for the latest data inserted. The classification of website performance would work in a similar way but instead, it would look at several statistics, like average, standard deviation, median, and others, from all the data available for a specific website. For now, the use of Groovy scripts seems like the most appropriate way to try this as they can be sent to ElasticSearch as custom queries, and run programmatically (E.g., every time new data is inserted).

# Reference list

AN, D., 2018. Find Out How You Stack Up to New Industry Benchmarks for Mobile Page Speed Available from: <https://www.thinkwithgoogle.com/marketing-strategies/app-and-mobile/mobile-page-speed-new-industry-benchmarks/>

BLAZEMETER, 2021. JMeter, Load & Continuous Performance Testing Platform [viewed February 2021]. Available from: <https://www.blazemeter.com/>

CALVANO, P., 2018. Impact of Page Weight on Load Time [viewed February 2021]. Available from: <https://paulcalvano.com/2018-07-02-impact-of-page-weight-on-load-time/>

ELASTIC, 2021. Search APIs | Elasticsearch Reference [viewed February 2021]. Available from: <https://www.elastic.co/guide/en/elasticsearch/reference/current/search.html>

GOOGLE, 2019. Lighthouse - Tools for Web Developers [viewed February 2021]. Available from: <https://developers.google.com/web/tools/lighthouse>

GORMLEY, C. and Z. TONG, 2015. Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine. Google Books. O’Reilly Media, Inc.

GUPTA, Y., 2015. Kibana Essentials. Google Books. Packt Publishing Ltd

HTTP ARCHIVE, 2021. Loading Speed Available from: <https://httparchive.org/reports/loading-speed?start=earliest&end=latest&view=list>

LIGHTBODY, P., 2021. Lightbody/browsermob-proxy [viewed February 2021]. Available from: <https://github.com/lightbody/browsermob-proxy>

MACHMETRICS, 2021. Website Speed Monitoring [viewed February 2021]. Available from: <https://www.machmetrics.com/>

MATAM, S. and J. JAIN, 2018. Pro Apache JMeter: Web Application Performance Testing. Apress, p. 1

MUNYARADZI, Z., G. MAXMILLAN and M.N. AMANDA, 2013. Effects of Web Page Contents on Load Time over the Internet. International Journal of Science and Research (IJSR), 2(9)

ODVARKO, J., 2021. HAR 1.2 Spec [viewed February 2021]. Available from: <http://www.softwareishard.com/blog/har-12-spec/>

P. RAMYA, V. SINDHURA and P.V. SAGAR, 2017. Testing Using Selenium Web Driver. In: 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT). IEEE, p. 1-2

PRATT, M., 2019. Jest - Elasticsearch Java Client [viewed February 2021]. Available from: <https://www.baeldung.com/elasticsearch-jest>

SCHIBSTED MEDIA GROUP, 2021. JSTL [viewed February 2021]. Available from: https://github.com/schibsted/jslt

SEMATEXT, 2021. Sematext: Metrics, Logs, Frontent, Uptime & API Monitoring [viewed February 2021]. Available from: <https://sematext.com/>

UPTRENDS, 2021. Calculation of uptime and downtime Available from: <https://www.uptrends.com/support/kb/reporting/calculation-of-uptime-and-downtime>

W3C, 2017. Long Tasks API 1 Available from: <https://www.w3.org/TR/longtasks/>

WEBDEV, 2019. Total Blocking Time Available from: <https://web.dev/lighthouse-total-blocking-time/?utm_source=lighthouse&utm_medium=devtools>

# Bibliography

ALTVATER, A., 2020. What is Real User Monitoring (RUM)? Examples and Tutorials. Available from: <https://stackify.com/what-is-real-user-monitoring/>

AN, D., 2018. Find Out How You Stack Up to New Industry Benchmarks for Mobile Page Speed. Available from: <https://www.thinkwithgoogle.com/marketing-strategies/app-and-mobile/mobile-page-speed-new-industry-benchmarks/>

APACHE, 2021. The Apache Groovy programming language [viewed February 2021]. Available from: <https://groovy-lang.org/index.html>

BECK, K. et al., 2001. The agile manifesto

BLAZEMETER, 2021. JMeter, Load & Continuous Performance Testing Platform [viewed February 2021]. Available from: <https://www.blazemeter.com/>

CALVANO, P., 2018. Impact of Page Weight on Load Time. Available from: <https://paulcalvano.com/2018-07-02-impact-of-page-weight-on-load-time/>

CURA, C.A.Z., 2013. Page load performance analysis

DOCKER, 2018. Enterprise Application Container Platform [viewed February 2021]. Available from: <https://www.docker.com/>

ELASTIC, 2021. Search APIs | Elasticsearch Reference [viewed February 2021]. Available from: <https://www.elastic.co/guide/en/elasticsearch/reference/current/search.html>

GOOGLE, 2019. Lighthouse - Tools for Web Developers [viewed February 2021]. Available from: <https://developers.google.com/web/tools/lighthouse>

GORMLEY, C. and Z. TONG, 2015. Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine. Google Books. O’Reilly Media, Inc.

GUPTA, Y., 2015. Kibana Essentials. Google Books. Packt Publishing Ltd

HTTP ARCHIVE, n.d. Loading Speed. Available from: <https://httparchive.org/reports/loading-speed?start=earliest&end=latest&view=list>

JANSEN, B.J., 2009. Understanding User - Web Interactions via Web Analytics. Available from: <http://www.iro.umontreal.ca/~nie/IFT6255/Books/User-WebInteractions.pdf>

JMETER, 2021. Apache JMeter [viewed February 2021]. Available from: <https://jmeter.apache.org/>

JOHNSON, J., 2017. What is a HAR File? Available from: <https://blog.stackpath.com/glossary-har-file/>

KASHYAP, K., 2016. BrowserMob: Proxy for WebPage Load Testing Using Selenium Available from: <https://www.tothenew.com/blog/browsermob-proxy-for-webpage-load-testing-using-selenium/>

KEYCDN, 2017. What Is a HAR File? - KeyCDN Support Available from: <https://www.keycdn.com/support/what-is-a-har-file>

LIGHTBODY, P., 2021. Lightbody/browsermob-proxy [viewed February 2021]. Available from: <https://github.com/lightbody/browsermob-proxy>

MACHMETRICS, 2021. Website Speed Monitoring [viewed February 2021]. Available from: <https://www.machmetrics.com/>

MATAM, S. and J. JAIN, 2018. Pro Apache JMeter : Web Application Performance Testing. Apress

MDN, 2021. Performance Monitoring: RUM vs synthetic monitoring - Web Performance | MDN [viewed February 2021]. Available from: <https://developer.mozilla.org/en-US/docs/Web/Performance/Rum-vs-Synthetic>

MUNYARADZI, Z., G. MAXMILLAN and M.N. AMANDA, 2013. Effects of Web Page Contents on Load Time over the Internet. International Journal of Science and Research (IJSR), 2(9)

ODVARKO, J., 2021. HAR 1.2 Spec [viewed February 2021]. Available from: <http://www.softwareishard.com/blog/har-12-spec/>

P. RAMYA, V. SINDHURA and P.V. SAGAR, 2017. Testing Using Selenium Web Driver. In: 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT). IEEE, p. 2

PERFORMANCE LAB, 2021. Why Is Load Testing Important for Web Applications [viewed February 2021]. Available from: <https://performancelabus.com/load-testing-for-web-applications/>

PRATT, M., 2019. Jest - Elasticsearch Java Client [viewed February 2021]. Available from: <https://www.baeldung.com/elasticsearch-jest>

SCHAPIRA, B., 2019. TTI: Time To (consistently) Interactive, how to measure web page interactivity Available from: <https://blog.dareboost.com/en/2019/05/time-to-interactive-tti-measure-interactivity/>

SCHIBSTED MEDIA GROUP, 2021. JSTL [viewed February 2021]. Available from: <https://github.com/schibsted/jslt>

SEMATEXT, 2021. Sematext: Metrics, Logs, Frontent, Uptime & API Monitoring [viewed February 2021]. Available from: <https://sematext.com/>

SEMATEXT, 2021. Application Performance Monitoring 101: Tools, Metrics & More [viewed February 2021]. Available from: <https://sematext.com/guides/application-performance-monitoring/>

SMARTBEAR, 2021. What is Real-User Monitoring? [viewed February 2021]. Available from: <https://smartbear.com/learn/performance-monitoring/what-is-real-user-monitoring/>

UPTRENDS, 2020. Calculation of uptime and downtime Available from: <https://www.uptrends.com/support/kb/reporting/calculation-of-uptime-and-downtime>

W3C, 2017. Long Tasks API 1 Available from: https://www.w3.org/TR/longtasks/

WALTON, P., 2019. First Contentful Paint (FCP) Available from: <https://web.dev/fcp/>

WEBDEV, 2019a. Time to Interactive [viewed February 20221]. Available from: <https://web.dev/interactive/?utm_source=lighthouse&utm_medium=devtools>

WEBDEV, 2019b. Total Blocking Time [viewed February 2021]. Available from: <https://web.dev/lighthouse-total-blocking-time/?utm_source=lighthouse&utm_medium=devtools>

ZHANG, M. et al., 2012. Web page load time prediction and simulation

1. The number of seconds from the time the navigation started until the initial HTML document has been completely loaded and parsed. (HTTP Archive 2021) [↑](#footnote-ref-1)
2. The number of seconds from the time the navigation started until the document and all of its dependent resources have finished loading. (HTTP Archive 2021) [↑](#footnote-ref-2)