Solent University

Faculty of Business, Law and Digital Technologies

BSc (Hons) **Software Engineering**

Academic Year 2020-2021

João Leite

Web Monitoring for Performance Regressions

Supervisor: **Craig Gallen**

Date of Submission: **May 2021**

Acknowledgements

First, I would like to thank my parents for the faith they have shown in me and for giving me the opportunity to move to the United Kingdom in pursuit of a better future. My friends and house mates, João Silva and Rui Pinto for being there for me through this journey and relentlessly trying to help me during the worst times. I would also like to thank my lecturer Craig Gallen who, besides recommending this research topic, also provided full support, and showed the best interest during the development of this dissertation. Finally, I am very grateful to have been given this opportunity to live and study abroad, where I came through all the adversities for the last three years, growing as a person in ways that I had not even imagined.

Abstract

With the increasing reach of web technologies, websites are fundamental to anyone with an online presence. This paper does an analysis on the ever-growing customer demands and how those needs can be fulfilled, focusing on problems encountered by users when navigating the web and how those can be detected with the use of web monitoring tools. A web monitoring artefact was built with the main purpose of detecting the root causes of the problems real users experience when navigating a page. The report describes the choices and approaches for the methods used and supports them with existing literature and current web standards. Finally, an analysis of the monitoring results was done to ensure its monitoring accuracy and validity for problem detection. This was tested by using the artefact to monitor different pages and compare the results with similar tools. The outcomes showed that the artefact can be used for user problem detection on web pages, as the tool was able to correctly identify the problems artificially inserted in the page.

Contents

List of Tables

List of Figures

1. 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 of today's, their platform should be optimized for ease of use and consistent performance across devices. The rapid changes in technologies and all the parts of complex web services, make it so companies now have the need to monitor and analyse their platform behaviour to improve user Quality of Experience (QoE).

QoE is defined as the overall acceptability of an application or service, as perceived subjectively by the end-user (International Telecommunication Union 2007), this means it is never the same for everyone. Regular web standards like quick loading, good design, and ease of use are often the metrics that positively influence user experience. First impressions also have a huge impact here, as people often determine if they are choosing a service based on it. Analysis of user data shows that most do not return to a website where they had a bad experience, being it because of performance or aesthetics. This was proven with a research conducted in 2017 that discovered that the user bounce rate for pages that load between 1 and 5 seconds is 90% (An 2017), this reinforces the idea that website performance is a key aspect for users. 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, Real-User Monitoring (RUM) and Synthetic Transaction Monitoring (STM) (Cito *et al.* 2015 p.418-419). STM systems simulate end-user experience and record the requests made in the browser, because tests are run from a controlled environment the system can gather data consistently. Tests can be manipulated or expanded as needed, and the results are good for measuring availability and performance as there is less noise in the data. RUM is done by monitoring real user behaviour

directly from the page, it gathers different types of user-tailored data. This data gives RUM a huge potential in the marketing sector as it can be used to measure important metrics about the users such as bounce rate or unique visitors.

This paper is a study of how STM could be used to detect problems end-users encounter when navigating a page by looking into performance degradations in web services. Its aim is to build a test harness to perform synthetic monitoring with focus on the individual requests of a webpage, analysing them to find the root cause of degradations found by users and classifying webpage performance.

The proposed system uses open-source tools to gather data through simulations of end-user experience. Simulations were done using Selenium WebDriver[1], which can then be pre-processed and saved to ElasticSearch[2]. The data can then be seen via graphs and charts in Kibana[3]. The Selenium WebDriver is a tool to create browser-based tests, in this context, a test case is a set of instructions that is executed in a browser to replicate real user behaviour (P. Ramya et al. 2017), which gives us the freedom to expand or modify tests as necessary. ElasticSearch is part of the Elastic Stack[4], a distributed, scalable, real-time search and analytics engine, as well as the database, that holds the data for visualization (Gormley and Tong 2015). Kibana is also a tool from the Elastic Stack so it was designed and optimized to work seamlessly with ElasticSearch, this tool provides a dynamic interface for real-time data analytics and visualization with customizable dashboards.

The system's efficacy was verified by performing web monitoring for extended periods of time against publicly available websites, which can be expected to have a decent amount of user traffic. Furthermore, monitoring was

[1] https://www.selenium.dev/documentation/en/webdriver/
[2] https://www.elastic.co/elasticsearch/
[3] https://www.elastic.co/kibana
[4] https://www.elastic.co/elastic-stack

done against a testbed website, that got updated during monitoring with known performance degradations to measure its ability to find root causes of web services slowdowns.

Background research was done about several topics around web monitoring. The main topics being, how others attempted to create monitoring tools or solve similar problems, what technologies are they using and what methods were used to analise these types of results; as well as user QoE, to get further insight into what metrics matter the most when navigating the web, what components affect QoE the most, what are the most common root causes of component slowdowns and what are the recommended actions to optimize those problems.

Finally, the artefact produced was evaluated, as well as its monitoring accuracy and usefulness for problem detection. Monitoring of a real website and a test website were done using both the built artefact and other monitoring tools to see how it would compare with the current industry standards, as well as verify its usability in real life scenarios. The results gathered from the different tools were compared with the artefact to accurately evaluate the research success.

2. Literature Review

With the fast development of internet technologies and their increasing reach, there is a huge demand for existing businesses to move/expand online as well as a wave of new companies going mostly digital. Like clients entering a physical store, first impressions produce a heavy impact on the business reputation, customers normally expect a clean and tidy store with good customer service. The same applies to websites, users still expect a good-looking page with an easy-to-use design and consistently fast performance across devices. Websites that are up to their users' standards, get better turnover rates, meaning users are more likely to visit for longer, come back and/or go through with an order. Some companies can even see a direct correlation between their website performance and their profitability as a business, which highlights the importance of having a good online presence (Matam and Jain 2018 p.1).

With most companies trying to differentiate themselves, their websites are often backed by a large web application with lots of layers and different components which together with a browser build the pages showed to users. But the size of web services makes them more prone to performance issues as they are consequently harder to maintain.

In a report from ContentSquare digital experience in 2021, they monitored performance and quality of experience metrics of over 900 global websites. They have found that a delay of just 100 milliseconds in a page load time can drop page conversion rates by 7%, and a delay of 2 seconds can double a website bounce rate, these results emphasize the importance of a good performance for business websites.

The report does a deeper analysis of websites performance by looking into core web vitals[5]. Time to First Byte (TTFB)[6] which indicates the time between a

[5] https://web.dev/vitals/
[6] https://web.dev/time-to-first-byte/

user accessing a page and the first data received was analysed and results show websites median load time to be more than double of Google recommendation (200ms). They also show a big discrepancy between the results for mobile and desktop visits, with the former averaging more than double the latter. Findings of other web vitals show that the average website performance is still way below recommended levels. A similar analysis, using some of Google's core vitals was used to assess this project validity according to the known standards.

To detect problems and find components that are negatively affecting a website, one can perform web monitoring and analysis. Web monitoring consists of testing a website across time to verify end-users are interacting with it as expected, the content showing as intended and the page loading quickly (CyberSafe 2021). To get an idea of how a monitoring solution could be built, research papers about the types of monitoring, development of similar applications, and how are they being analysed was taken into consideration.

Web monitoring can be done in two different ways, through STM or RUM, also known as active and passive monitoring respectively. Active monitoring works by simulating an end-user experience and measuring the website performance through the results collected; tests are normally done in uniformly distributed time intervals allowing for the gathering of data even when no users are accessing the website, the result is a complete view of availability and performance. Contrary to active monitoring, passive monitoring only gathers data when the system is being used, measuring end users experience, directly from the browser, which coincidently can be used to gather different types of user-specific data like engagement, behavioural or personal information like browser used, geographic location, IP address, browser cookies, etc. (Cito et al. 2015 p.418-419)

One of the main concerns with RUM is how businesses can abuse it to gather user's personal data, which can be sold to outside businesses for profit, this is a big opportunity for any company with large amounts of users to make

extra income with the data they were gathering for themselves. The businesses who buy it mostly use it to improve their own customer experience or redefine their marketing strategy to better fit their targets. In recent years countries have stepped in with legislations, like the Data Protection Act 2018[7] by the UK, to control the usage of data in an adequate and legitimate way, by restricting the possibilities to gather some sensitive data, providing transparency to users about what data is being gathered, and what it will be used for.

For a more complete analysis, it would be possible to implement both types but to implement passive monitoring, direct access to the web service is required. Code can be placed on the page that gathers data from user sessions. As not everybody has access to their web service, active monitoring can still be done at any time by anyone; some of the advantages that it offers over RUM are, the availability to monitor consistently, and a controlled environment to test the product, even before launch or updates, which generates more accurate results to benchmark the website performance (2 Steps Team 2019).

In a study on quantifying QoE for web applications by Lycett and Radwan 2018, they measure it by developing a model which uses a mixture of Key Quality Indicators (KQI) and Key Performance Indicators (KPI). KQI are external indicators that reflect subjective user experience and derive from KPIs, that generate a more objective user experience of measurements like performance, availability, reliability, and usability, by measuring the network resources.

The model they have built runs the data in two different tests, (1) uses an Actual-Versus-Target approach (Kan et al. 2001) to determine the accuracy between results and expected results (good QoE), (2) uses the results of the first test and applies supervised machine learning to it, several supervised algorithms were used during the second iteration to dynamically predict and evaluate KQIs.

[7] https://www.gov.uk/data-protection

The authors stated that the evaluation on the first test was not conclusive but provided a great way to do comparison studies between actual and preferred KPI and KQI values, this can be used by the QoE assessor to better define problems priority and distribute resources accordingly. The second test, used to predict QoE, observed increased accuracy when using certain algorithms. These results also provide a better way to understand the relations that bridge the service quality with the user experience. As the authors suspected, both metrics were found to show high rates of correlation throughout the experiments. Outcomes for both, objective and subjective data, produce specific results that can be grouped in subsets like, user types, service provided, etc.

This approach to web monitoring can be very useful to learn new insights about the different parts of a service. Especially for a STM system like the one developed for this project, a similar approach can be used to measure the solution monitoring accuracy by using KPIs as the main measurement metric.

In a whitepaper by Sandvine, they define the valuable metrics to get a meaningful insight into QoE. They did it by using a machine learning approach to build the anatomies of web pages.

They have focused on the page load wait time, which they state is the key measurement for website performance. They source experiments done by big companies, where adding a small delay to their pages showed a downward trend in their business profitability. They discuss that users navigate the web in a flow state, with complete focus while happily working or navigating, and when they encounter long waiting times the flow is broken, which tends to negatively influence the user perspective of the visited webpage. The visual page load time is not always linear as websites can prioritize the load of specific components to create an illusion of a complete webpage before all transactions are complete.

The contributing factors to a page load performance were separated into end-user device, web page and network characteristics. The first can be used to provide information about how different subsets of users experience the system,

the others two contain the factors that may be the root of any delays or inefficiencies, allowing a system administrator to act accordingly.

When defining which pages should be monitored, their focus was on the most important pages like portals, partners content websites and the most popular (i.e., majority of traffic, landing pages, popular products). This learning method helps isolate different factors and perform recurring analysis against them as subsets; some examples are looking for low QoE patterns with certain combinations of end-user devices/software or recurring bad performance at certain time intervals due to system inadequacies like non-optimized system processes (i.e., garbage collection). Their developed system gathers data from users, uses machine learning to learn and construct an anatomy profile of the pages, measures the load time perceived and by comparing the page anatomy against the load time formulates the report.

Although both show good methods to the web monitoring problem and present crucial information about it, the whitepaper by Sandvine focuses on one very important KPI that can be used to aggregate most of the others and takes a good approach to test it by using past performances as the main unit of measurement for current performance, as well as focuses on uncovering the root of the problem by closely analysing the page skeleton.

In another research, by Cito et al in 2015, the authors test the effectiveness of RUM and STM systems to detect web performance degradations. They set up and discuss the use of the two monitoring types and provide a statistical analysis of the results. It starts by explaining web transactions anatomy and the purposes of passive and active monitoring, discussing their advantages over another and they can be complementary systems. STM provides a more accurate representation of general website performance, but RUM can provide results in user subsets like performance by demographic groups.

Their artefact was tested by monitoring a web service injected with different performance degrading components and analysing the results to try and

find their root causes. The authors have defined three main types of web delays, (1) Global Delay, a significant increase in overall page response time, can be due to software releases, server components bottlenecks or server misconfigurations; (2) Periodic Delay, not continuous but repeating, cause can be timed background processes like system backups or software garbage collection; (3) and Partial Delays, only occur to a subset of requests, can be caused by geographic location due to inefficient Content Delivery Networks (CDN) or load balancing routing.

All tests were then analysed with changepoint analysis to identify the points in time where statistical properties change which helps understand when request load time changes radically. Detecting these changes can also be useful to understand the web service trends, or life cycle of certain tasks. By visually interpreting the results, global and periodic delays were clearly seen, as for partial delays the change was almost too subtle and it was harder to detect the points where the degradations were inserted or removed, which could lead to a misinterpretation of results. When compared with the results of the statistical analysis, the findings were closely related to the observations from the graphs, with the global delay being the most accurate despite an error, and the other two showing several inaccuracies in its detections.

While applying changepoint analysis to the RUM results did not complement the results for more accurate performance so they were concluded to be insufficient for performance monitoring.

The authors conclude that the two systems should be used together, with active monitoring giving an advantage in availability and performance. The changepoint analysis was a good way to detect changes but setting the threshold manually could have been done instead by looking at previous performance and extracting the data level from there. An evaluation inspired by changepoint analysis was be used to test this research artefact usefulness for problem detection.

A study by Ahmed et al., 2016, takes a different approach by using Application Performance Management (APM) tools to diagnose performance regressions and reviewing their effectiveness. APM tools are used to monitor the performance and availability of web-based applications. Can also gather several hardware and web performance metrics such as CPU utilization and response time. During analysis, the tools periodically collect data and determine whether transactions are slower than usual. It can also notify the user of the detected error in a categorized way, pointing to a possible source of the problem such as excessive memory usage or slow database queries.

The tests were conducted by running load tests on a test website, redeploying it with a performance regression and finally analysing the results. The authors decided to test several types of regression problems using a range of APM tools. The case study results showed very varied outcomes across tools, with some doing very well and other very poorly. Still, the tools averaged a 75% accuracy rate when detecting anomalies, except for RESTful web services, with an accuracy of 20%, which are good results considering the tools' purpose. The authors agreed these tools lack the flexibility to try different approaches when testing websites and still require too much manual work to detect regressions.

Similar testing was applied in this research using other web monitoring tools to compare results. This generates a more accurate measurement of the tool usefulness and can also be used to get a better understanding of current industry standards.

3. Project Specification

The project artefact was built in Java and uses several tools and libraries to complete the use cases necessary to simulate end-user experience and analyse the results gathered. The project scope was defined with a set of functional and non-functional use cases that would be part of the complete monitoring system. The first goal was to achieve a Minimum Viable Product (MVP) by completing all functional requirements and expand accordingly from there.

Additionally, a website for testing purposes was created and used with performance regressions to test the artefact. A discussion of the use cases achieved can be found in chapter 6.1.

Functional	Non-Functional
Simulating end-user interaction with a website	Package application as containers
Running tests programmatically on a recurring basis	Deployment and hosting of application in the web
Saving simulation transactions to database	Programmatically detecting fundamental changes in page load times
Creating time-series visualisations for web monitoring	Programmatically detecting components causing page slowdowns

Table 1 - Artifact Requirements

4. Methodology

To understand how a project like this could be developed and assess its monitoring accuracy, a range of resources from papers, journals, patents and the official documentation of the technologies used were gathered. The main research consists of how others have created and evaluated their web monitoring tool, how can said systems find the root cause of regression problems, the advantages of certain types of systems over others, and how those can be used to accurately evaluate QoE for website users.

The literature reviewed served as a good foundation to develop the proposed monitoring solution and inspired the creation of the test harness used. Some of the methods of those works were modified and adapted to this project needs.

4.1 Data Gathering

Because the focus is the perceived performance of web pages to end-users, the system collects data in a way that simulates user experience and behaviour. Different software tools take different approaches for data collection but to simulate user experience the project is using Selenium WebDriver. This tool is being used on a timed basis to reach out to a target website, when run, it uses the Firefox browser (other browsers supported) to replicate, in real-time, the behaviour of an end-user on the page. The tool also allows changes in test cases in ways that simulates desired user actions, like clicking a button or filling a form, which can be used to perform functional testing.

When a user enters a page, there are often hundreds of web transaction happening in the background that build it. The transactions happen between the client browser and the hosting server through the HTTP protocol. This is the application layer protocol where HTTP transactions happen, when a client issues a request for HTTP content to the server a new transaction is started. Each transaction follows a set of steps from the time of request to receiving of the

response, and all details are registered in the transaction (Cito et al. 2015), the different fields of a transaction are often very good metrics for monitoring. Because Selenium does not save the data, the gap is filled with a library called BrowserMob Proxy[8] which records and saves the transactions between the WebDriver and target web service to a HAR[9] file. A HAR file is a JSON[10] based format, created to allow flexibility to pre-process its HTTP archives, which are composed of general data about the browser, some user details and all the HTTP transactions (Odvarko 2021).

This method, paired with the rest of the solution, allow the constant collection of transactions happening from the simulated browser. When properly pre-processed, the results can already be used to monitor a web page availability and performance without major efforts.

One limitation of this approach is that it is only performing synthetic transactions, so it lacks the real-user side of monitoring, on the other hand, this model was chosen with the goal of getting the most accurate measures possible and doing it in a controlled environment, with not many external variables, produces the best results.

4.2 Data Visualisation

Graphs are crucial to the task of web monitoring and problem detection as they reduce the manual work needed to manage the data which can instead be quickly analysed visually. To make the best out of the results, additional research of the data collected is needed. From the analysis of individual tests/HAR files and by looking into individual transactions, several important metrics can be extracted. Fields such as the timings of the transaction steps, data type, size of the request and target URL are the most important for plotting information. They can be used in Kibana to represent website availability, overall

[8] https://github.com/lightbody/browsermob-proxy
[9] http://www.softwareishard.com/blog/har-12-spec/
[10] https://www.json.org/

performance and transaction-specific performance through a combination of time-series plots and statistics (useful for detection of degradation root causes).

When it comes to determining availability and performance there are already several key metrics defined by businesses. These metrics were gathered by looking into Google's documentation for the performance audits of Lighthouse[11], as well as other web monitoring solutions (i.e., Dareboost[12], Uptrends[13]). Using several sources is very important to get a better understanding of the current standards for web monitoring applications and help create a more complete solution by merging features/visualisations from the different tools.

These tools are using a range of techniques to display their test results dynamically, but page load time was the key metric present in all of them. As the free version of the tools only performs a singular test on the target website, all transactions are unique. When organized and put together, result in the Gannt chart like the one in Figure 1, which can be seen in most browsers developer tools under the Network tab. A visualisation like this is very useful when it comes to taking a deeper look into webpage resources, making the time to load of each request very distinctive, meaning they can be easily identified.

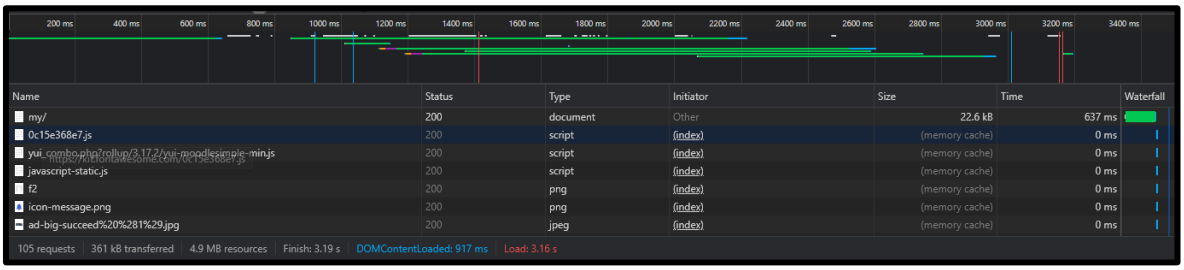

Figure 1 - Google DevTools Network (https://learn.solent.ac.uk/my/)

[11] https://web.dev/performance-scoring/
[12] https://www.dareboost.com/en
[13] https://www.uptrends.com/tools/website-speed-test

4.3 Calculating Accuracy

Calculating accuracy using different monitoring tools was inspired by the analysis done by Ahmed et al. in 2016. To measure good monitoring accuracy, two tests were done using free monitoring tools and the results gathered were compared with the built artefact. Accuracy was calculated through a balance of two KPI, the total time to load and page weight. The base metrics were collected from the following tools:

- Uptrends
- Dareboost
- Firefox Network Tab

Both Solent's website and the test website were monitored, and the results were compared using relative standard deviation.

Having in mind that the different tools could also yield different results, these were compared separately. The tests settings available in the tools were modified to be the closest as possible to the developed artefact and the data was gathered within a 5-minutes interval to minimize possible influence of external variables.

The results were classified as shown in *Table 2*, from 1 to 4, being 4 the best possible outcome, according to the relative standard deviation between the KPIs of the free tools and the artefact produced.

Relative Standard Deviation	Classification
<15%	4
<30%	3
<50%	2
>50%	1

Table 2 - Classification by Relative Standard Deviation

4.4 Problem Detection

When testing the artefact for problem detection, it was first used to see what valuable data it could gather in a long time period, and then, taking an approach similar to the testing done by Cito et al in 2015, it was tested by monitoring a test page inserted with performance regressions.

The first one consisted of analysing changes in already existing components of a live web application, which were tested by monitoring Solent's website[14] for 3 days. Gathering data for long periods of time allows you to gain further insights into possible recurring problems, which can appear as patterns and often originate from standard procedures in the hosting server like garbage collection, backups, or log generation; or individual instances like performance bottlenecks due to bad code/configurations or amounts of users that exceed the capacity of the service.

The second one is the overall website performance when there are new updates/releases. Testing took a similar approach to the study by Cito et al (2015) by using a test web service that was monitored before and after regressions were added or modified. This was tested against a website with no visitors where two different updates were tested. The different versions of the website were monitored for one hour each and results were compared at the end by using the visualisations in Kibana. Two performance regressions were used which consist of changes to the chosen website template. The first one aimed to detect the specific requests that slow down the system by adding non-optimized stylesheets to the page, while the second one involved a similar test where three of the website images were substituted with non-compressed high-quality ones.

Since the regressions were injected manually, the root cause for the delays is already known so it only needs to be identified in the monitoring tool results.

[14] https://www.solent.ac.uk/

The effectiveness for problem detection was classified according to how easy it was to detect and distinguish the problems root cause.

5. Implementation

The solution built is composed of two distinct applications, the STM system used to test the research question and a website to validate its effectiveness.

5.1 Monitoring Tool

The monitoring tool can also be separated into four main components, capturing, storing, and visualising data and application deployment. It runs Selenium tests on a timed basis, generating a flow of data that is saved to ElasticSearch, and viewed through metrics and graphs in Kibana's interface. The project has been packaged into Docker[15] images and deployed to a cloud hosting environment to facilitate testing and gathering of results.

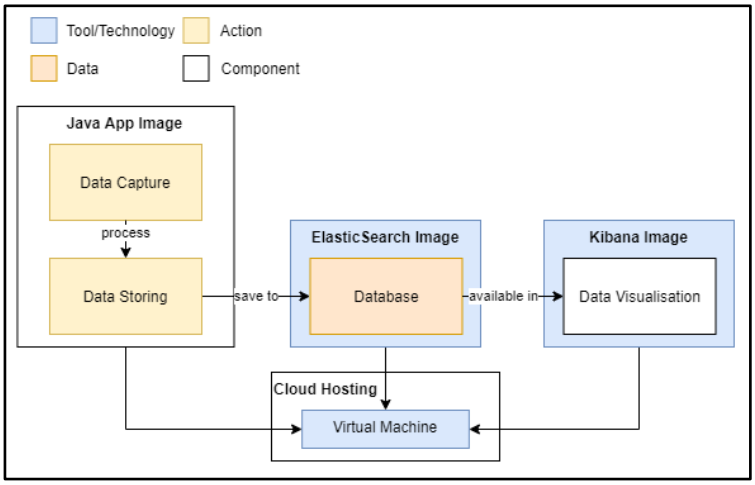

Figure 2 - System Diagram

5.1.1. Data Capture

The data capture happens by using Selenium WebDriver to replicate a user interaction with a page. It works by executing a Selenium test case, using Firefox (other browsers supported). Different test cases can be added to perform various tests, this is useful for functional testing of websites as it can be adapted to individual test cases like submitting a form or navigating through pages. But for

[15] https://www.docker.com/

gathering performance data, a test that gathers all transactions at website load is enough. But Selenium does not capture the data, so a proxy was used to capture the contents of all requests and responses between the simulated browser and the target website.

Selenium needs a driver for the desired browser, so an instance of GeckoDriver[16] (driver for Firefox) is downloaded when building the Docker image and made available to the application as a system property by using arguments. Selenium also allows you to specify other capabilities, such as the time to accept a request before losing connection, accepting insecure requests, and running in headless mode. This mode is very important when using Selenium for performance tests as it gets rid of the user interface. This means tests will run with increased speed and allows for testing of servers, containers, APIs, etc (Magsby 2019).

As mentioned, the library used as the proxy was BrowserMob Proxy which can be used to manipulate HTTP requests and responses as well as capture and export the data as a HAR file. Using a Java StringWriter, the content of the HAR are saved in memory before being processed and saved to the database, this process was done to reduce the writing and reading the application needed to access and manipulate the data.

All this logic is then put in a task, which runs through a scheduled thread pool with a one-minute interval. The repetition of tests is crucial for visualisation through time-series graphs as it provides data through time.

[16] https://github.com/mozilla/geckodriver

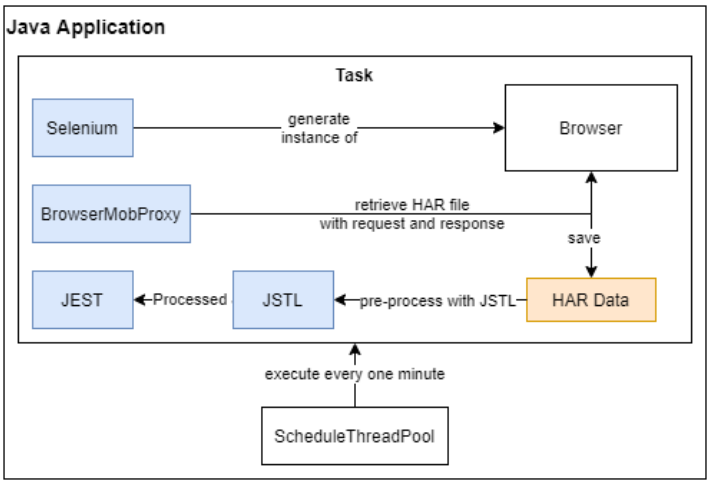

Figure 3 - Data Capture Diagram

5.1.2. Storing Data

Due to the large amounts of resources present in most web pages, the HAR files retrieved often consist of dozens of transactions, making the file fairly big. This means they need to be broken down before being sent to the database. This was done with JSLT[17], a query and transformation language for JSON. In this project the tool is used to transform the HAR files by breaking them down into individual transactions, so they can be saved separately. This transforms the data into smaller fragments that are easier to send to the database, as well as to analyse individually. This tool was also used to process the already existing data by removing the redundant information from the requests and responses (content text) and adding useful metadata (transaction start time for time-series visualisations and ID to group the broken-down transactions into simulated test). After processing, the data is outputted in NDJSON which is one of the best formats to work with ElasticSearch API.

When it comes to sending newly formatted data to the database, ElasticSearch provides a handful of RESTful APIs[18] (specifically Search and Index API) which can be used directly from the console of the hosting machine or within

[17] https://github.com/schibsted/jslt
[18] https://www.elastic.co/guide/en/elasticsearch/reference/current/rest-apis.html

Kibana's interface. A way to interact with the database programmatically was to use an external HTTP RESTful client called Jest [19], created to make the communication between a Java application and Elastic simpler by providing a fluent API with easy to work interfaces (Pratt 2019). Jest can be connected to ElasticSearch through its URL and port, and has a set of methods to manipulate its clusters, indexes and data, that can be used to push and index the web monitoring results.

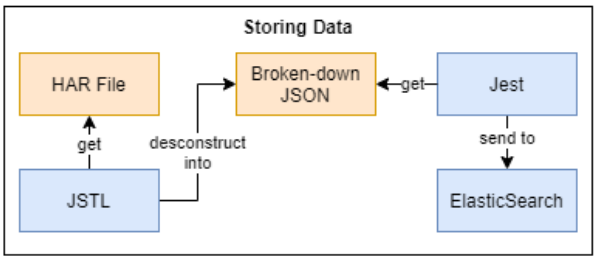

Figure 4 - Storing Data Diagram

5.1.3. Data Visualisation

The visualisations were made for the purpose of showing the website availability, performance and give several insights into the data gathered. It was also crucial that they could be used to detect slowdowns in components to validate the application usefulness. The 9 graphs created will now be shown and described according to their purposes.

Availability

The website availability can be seen in this vertical bar graph (Figure 5) that counts the number of requests, grouped by their metadata IDs, received in a determined time interval. If white spaces appeared between the bars, it would mean tests failed and the website was unavailable.

[19] https://github.com/searchbox-io/Jest

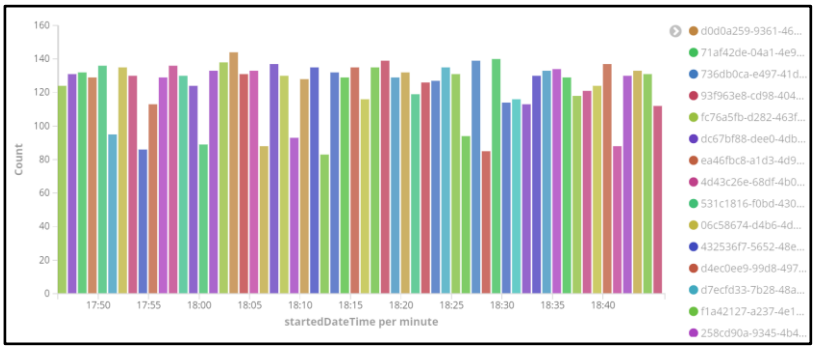

Figure 5 - Number of Request by Simulated Test Over Time (1-hour range)

Performance

The main time-series panel for performance (Figure 6) shows the sum of the timings of transactions in a determined time interval. When viewed in a one-hour range, shows the timings of individual tests however on larger intervals, it groups transactions, which results in false values for the sum by test but still accurately represents the website performance over time. The dark purple line is always at the top as it represents the total time, while the others represent the specific timings of the transaction steps, like time to connect, receive or wait.

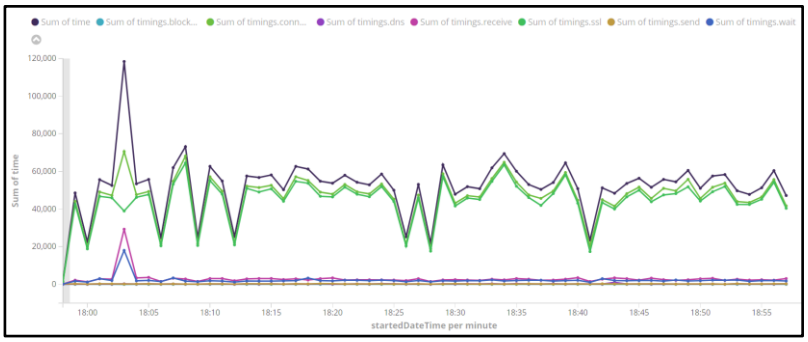

Figure 6 - Sum of Time by Requests Over Time (1-hour range)

The second panel (Figure 7) shows the average load time, the most important perspective from a user point-of-view and the sum of the response body sizes. These are also the metrics that were used to calculate the systems' accuracy. By using a ratio of 10 to 1 the average time represents milliseconds,

which in Figure 7 would be 5.125 seconds. The number for response body size is in bytes or 5.92 megabytes.

Figure 7 - Average Load Time and Average Response Body Size

The third one (Figure 8) can be used to gain further insight into the status of responses, it shows the number of transactions by their responses' status in a determined time interval. Larger intervals also show the incorrect number of requests but show the distribution of response status correctly.

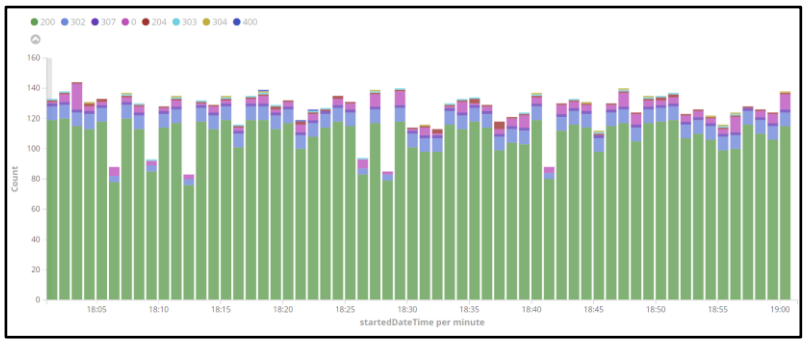

Figure 8 - Request Count by Response Status Over Time (1-hour range)

Problem Detection and Others

The main visualisation for problem detection (Figure 9) shows, in a time-series line graph, the 3 longer transactions found per test. Although the results can look messy when there are no components behaving in weird ways, the graphs should clearly show any request with distinguishable time to load by either a dot or a line peak.

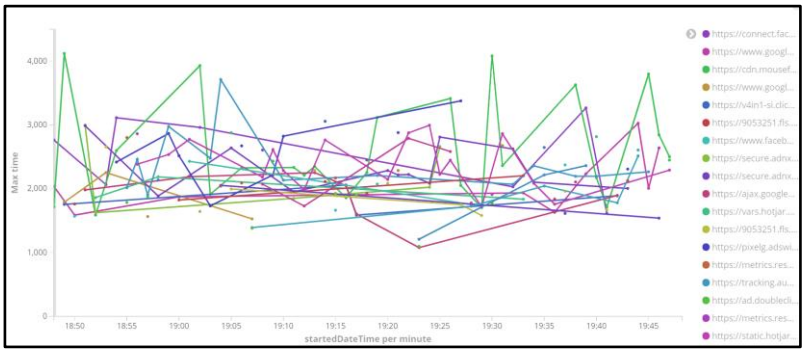

Figure 9 - Top 3 Transactions by Load Time Over Time (1-hour range)

The following pie charts (Figure 10) represent response types by amount, by sum of time and by sum of response body size respectively. These represent further the distribution of data according to the response data types. From these pies, it can be seen that the results show little or no correlation between the subsets.

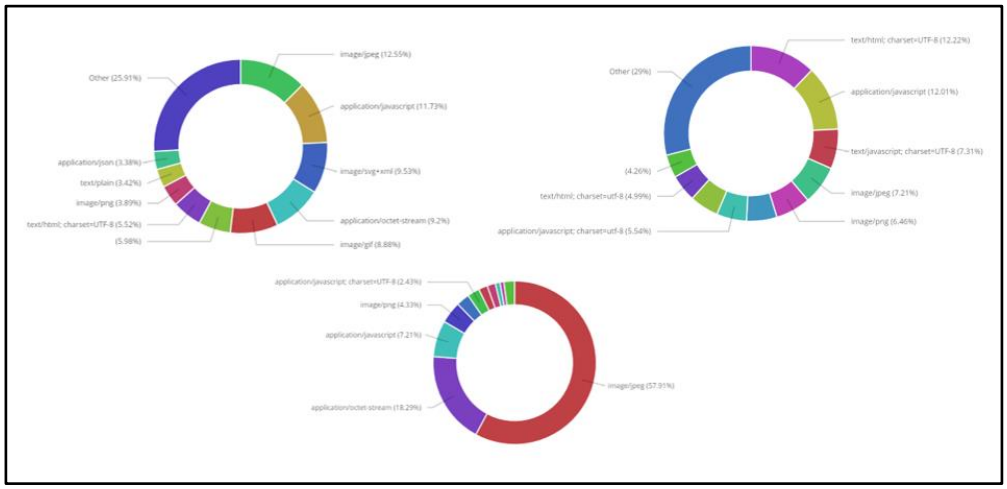

Figure 10 - Request Percentage by Data Type for Count, Sum of Time, and Sum of Response Size

The final bar graph (Figure 11) provide details of the timing's percentage according to response data type. Can be used to learn how long transaction steps take for different types of data.

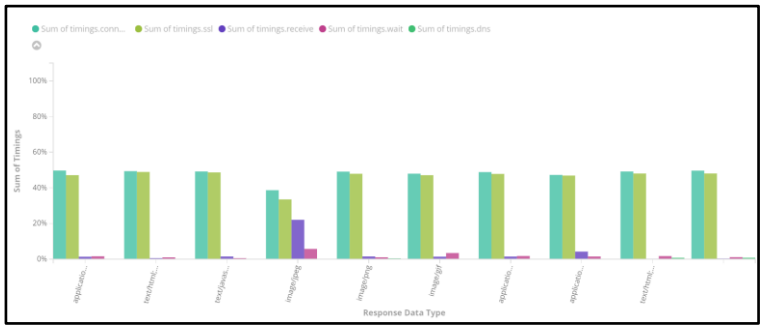

Figure 11 - Percentage of Timings by Data Type

5.1.4. Deployment

To facilitate testing, the application was bundled with Docker and shipped to a hosted virtual machine. The application is packaged with Apache Maven[20] so it is easier to reuse, this process runs the application Junit test and saves its source code into a JAR file[21], but is still missing all its dependencies. This issue was addressed by using a library called One-JAR[22] that packages everything into a single executable JAR file. When packaging with Maven commands the application is compiled, tested, and packaged to a One-JAR. This file can be directly executed with Java and can work assuming the hosting machine has all the necessary tools to do so, in this case, JDK 11, Firefox and its respective GeckoDriver.

Although it could already work on other devices as it made with Java, it is likely to cause compatibility problems or fail due to missing components, this was tackled by dockerizing the application. By creating a Docker image using a publicly available image of CentOS (Linux based operating system) with JDK 11, installing the necessary tools, and copying the JAR file into it, the application can now be run in any machine with Docker without causing problems as the code runs within a pre-defined environment. The Java application was structured as represented in Figure 12. Using configurations in the POM files, the project is

[20] https://maven.apache.org/
[21] https://docs.oracle.com/javase/8/docs/technotes/guides/jar/jarGuide.html
[22] http://one-jar.sourceforge.net/

built programmatically through the project parent directory. Starting by packaging the application logic to a JAR file in *Service*, adding the necessary dependencies in *Assembly* with One-JAR and finally, copying the One-JAR and building the image in *Docker*.

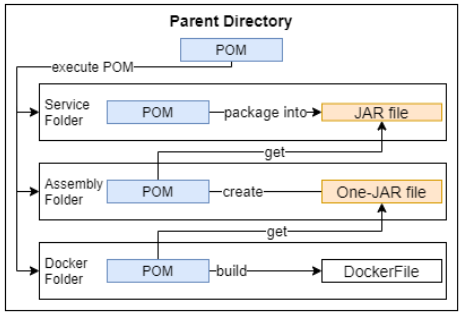

Figure 12 - Application Packaging Structure

Using a Docker Compose[23] file, the application gets the official images for ElasticSearch and Kibana, as well as the latest image packaged with Maven. It also opens the required ports for those tools to bridge communication and performs a health test that return the container status (red, yellow, green).

The application was then pushed to a GitHub[24] repository[25] so it could be easily accessed in other systems. With a virtual machine running a Linux based operating system, and the necessary technologies, (JDK, Maven, Docker and Docker Compose) hosted in Microsoft Azure[26], the repository could be cloned and executed. The normal steps to build and run the application locally were used to launch it in the virtual machine. The virtual machine specifications need at least 2Gb of RAM to avoid performance bottlenecks which can result in failed test simulations, as well as a decent size disk so it can support at least a week of results.

[23] https://docs.docker.com/compose/
[24] https://github.com/
[25] https://github.com/LEITE5/WebMonitoring/
[26] https://azure.microsoft.com/

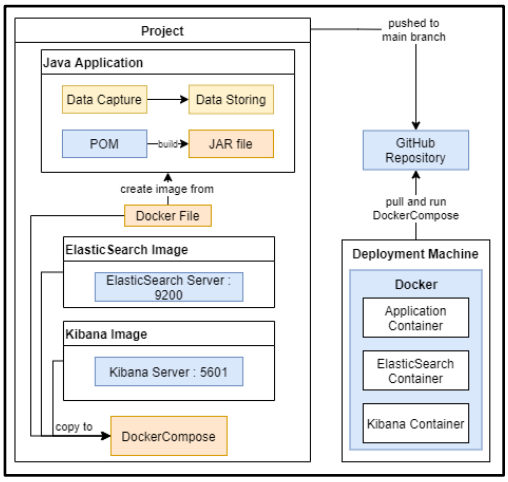

Figure 13 - Deployment Diagram

5.2 Website

The test website was made with a free bootstrap responsive template called Radiance[27]. A template was chosen to increase the complexity of the page so the injections would not be so clear due to a lack of components. The template was downloaded, and its Bootstrap stylesheets were removed to use as the original version (Fig. 14). The page was then hosted for free on GitHub Pages.

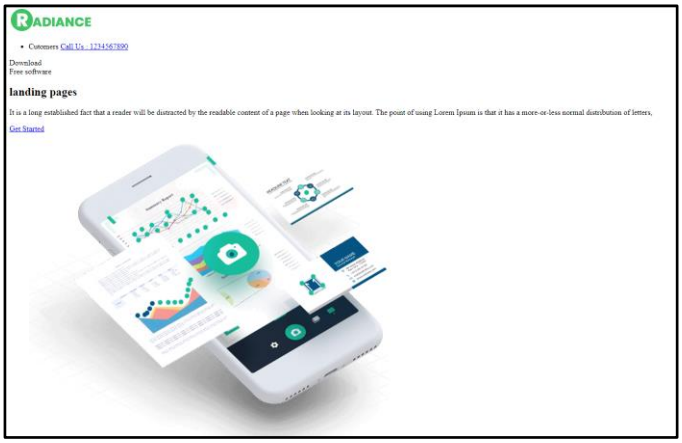

Figure 14 - Test Website Layout (No Bootstrap)

[27] https://www.free-css.com/free-css-templates/page266/radiance

The first injection was made by adding the non-optimized stylesheets to the page which increased both the number of requests and the page size, this aims to test the effect of resources CDNs.

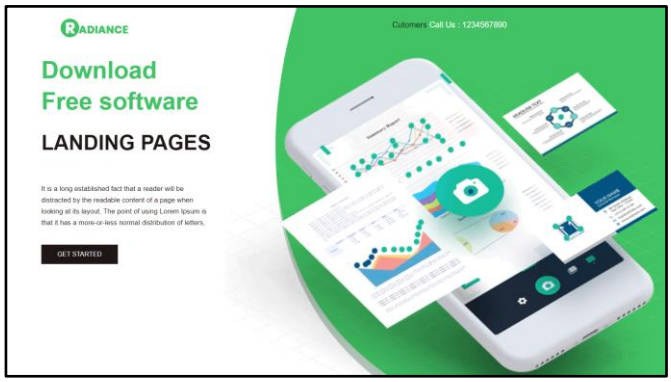

Figure 15 - Test Website Layout (With Bootstrap)

The second injection substituted three of the page images with high-quality non-compressed ones (over 500Kb) and aims to test its overall impact on the page performance.

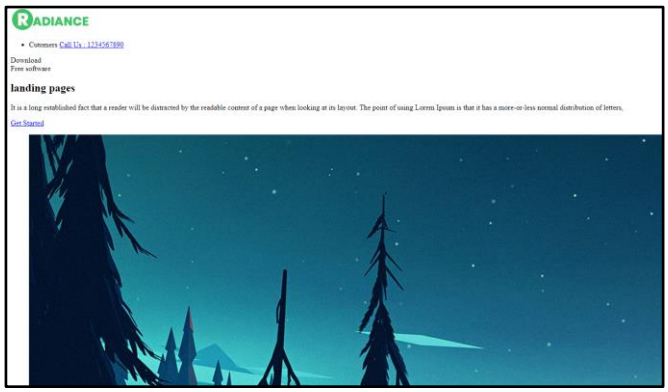

Figure 16 - Test Website Layout (with High-Quality Images)

The different pages were saved as separate GitHub commits to facilitate changes between versions. The injections were applied by changing the GitHub repository to the commit desired.

6. Results & Discussion

The discussion of results will be split into what was achieved for the final artefact having the requirements in mind, an analysis of its ability to monitor accurately and its effectiveness for problem detection.

6.1. Artefact

The proposed minimum viable product was achieved as all functional requirements were completed, and excess time was used to expand the application further with a focus on facilitating data gathering and availability for testing.

Use Case	Functional/Non-Functional	Achieved
Simulating end-user's interaction with website	Functional	Yes
Running tests programmatically on a recurring basis	Functional	Yes
Saving interaction transaction to database	Functional	Yes
Creating time-series visualisations for web monitoring	Functional	Yes
Package application as containers	Non-Functional	Yes
Deployment and hosting of application in the web	Non-Functional	Yes
Programmatically detecting fundamental changes in page load times	Non-Functional	No
Programmatically detecting components causing page slowdowns	Non-Functional	No

Table 3 - Achieved Use Cases Review

As described in chapter 5.1.1, the system can simulate user behaviour with a page using Selenium, the simulations happen programmatically, with a timer launching new tests separately. In 5.1.2, the process of sending the results to the chosen database, ElasticSearch, is explained, as well as how external tools were used to pre-process and send it, respectively. The following chapter (5.1.3) represents the process that went in the creation of the visualisations for availability and performance monitoring and what are their uses. Together this

achieved the MVP that could already be used for testing but was not so practical to use from different environments, as it still requires outside resources installed in the development environment. The artefact was extended a bit further by being packaged into Docker images and deployed to a hosted virtual machine (Chapter 5.1.4). All the monitoring was done using the application through its hosted version.

As seen in Figure 17, the artefact can already be used for monitoring a web page availability, and performance depending on the results achieved for its accuracy. By now it shows a lot of information about the page which can be used to learn about its transactions and possible patterns that can be found in components.

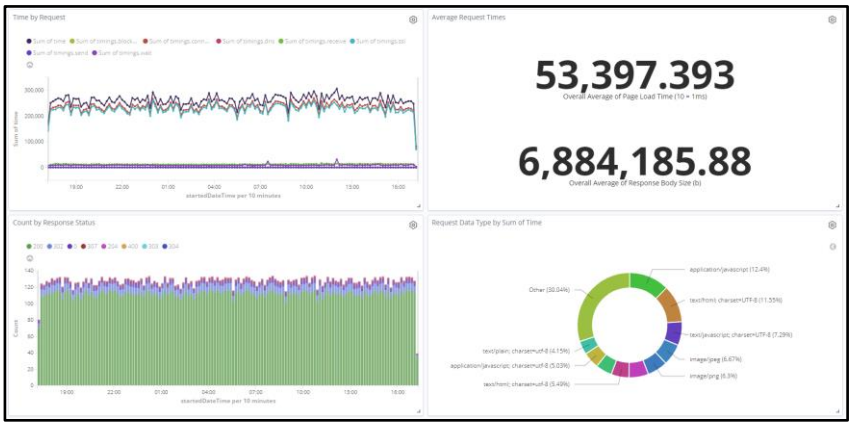

Figure 17 - Monitoring Solent's Website (1-day)

6.2. Calculating Accuracy

The results from the 3 tools will now be individually showed in tables, against Solent's and the test website respectively, and the relative standard deviation will be calculated. Since the values were already in a smaller scale than their more common measurement metrics (seconds and megabytes), the values were rounded to integers. The sources of the data can be found in Appendix A.

Table 4 represents the results from the tests done on Solent's website.

Source	Uptrends	Monitoring Artefact	Relative Standard Deviation	Score
Total Time	5741 ms	4899 ms	11.19%	4
Total Size	5200 kB	6786 kB	18.71%	3
Source	Dareboost	Monitoring Artefact	**Relative Standard Deviation**	**Score**
Total Time	5210 ms	4899 ms	4.35 %	4
Total Size	5550 kB	6786 kB	14.17%	4
Source	Firefox Network Tab	Monitoring Artefact	**Relative Standard Deviation**	**Score**
Total Time	5330 ms	4899 ms	5.96%	4
Total Size	5416 kB	6786 kB	15.88%	3

Table 4 - Page Load Time and Size of Solent's Website

From the analysis of the different tools results against each other's, a reasonable interval, for both time and size, can be seen. Their values do not show a higher relative standard deviation than 6% which means these tools are pretty accurate between themselves.

When analysing against the monitoring artefact the results fell a bit short of the comparison above but still yield excellent results, achieving maximum score for the monitored load time and being just 2 points off for page size. It can be concluded that the monitoring artefact accuracy in Solent's website is 3.66 out of 4 which are good results.

To understand how consistent these results can be against other web pages, these were compared with the monitoring of the test website by looking at the next table (Table 5).

Source	Uptrends	Monitoring Artefact	Relative Standard Deviation	Score
Total Time	835 ms	772 ms	5.54%	4
Total Size	628 kB	1228 kB	45.72%	2
Source	Dareboost	Monitoring Artefact	**Relative Standard Deviation**	**Score**
Total Time	945 ms	772 ms	14.25%	4
Total Size	579 kB	1228 kB	50.79%	1
Source	Firefox Network Tab	Monitoring Artefact	**Relative Standard Deviation**	**Score**
Total Time	533 ms	772 ms	25.90%	3
Total Size	579 kB	1228 kB	50.79%	1

Table 5 - Page Load Time and Size of Test Website

When now comparing the results from the different tools, they are not so close, with the Network tab detecting considerably less time than the other two, with 30% deviation, but very similar for page size with about 5% deviation.

Defining the accuracy for the test website yields some interesting results. As it can be seen, the values from the monitoring artefact are not far from the tools tested when it comes to the time to load, missing the maximum score by 1. But for the page size, the artefact seems to have detected about double the actual size, giving almost the worst possible score of 1.66. This could mean the artefact produced has trouble calculating the page weight when it comes to small pages with very few resources, as it is the case. Although this does not seem to have affected monitoring time, it is interesting to see that the performance indicators show no correlation between them.

This test score is 2.5, which is slightly above the acceptable results (halfway point - 2). The results ended up counterbalancing the accurate time monitoring with the poor size calculation.

The actual final score for the monitoring accuracy is 3.08 (or 77%) and was calculated by averaging the results of both tests. Despite the bad results for page size in the second test, the tool can be said to detect accurately most of the time and could already be used for real-world performance monitoring despite needing improvements. Still, way further testing would be needed against a more varied set of websites to ensure its monitoring accuracy.

6.3. Problem Detection

6.3.1. Solent Monitoring

For an overall test of the artefact for performance monitoring, Solent's website was monitored for a period of 3 days. From looking at the graphs in Figure 18, it already shows various details of page requests.

The pie charts (Figure 19) show that Solent has several types of requests taking about the same time to load and that no specific type takes way more time than others. That variety seems to be relatively consistent between the amount of request and their time to load but the same cannot be seen for the request size pie, where images take almost 50% of all the page size.

Figure 18 - Monitoring Data for Solent's' Website (3-day range)

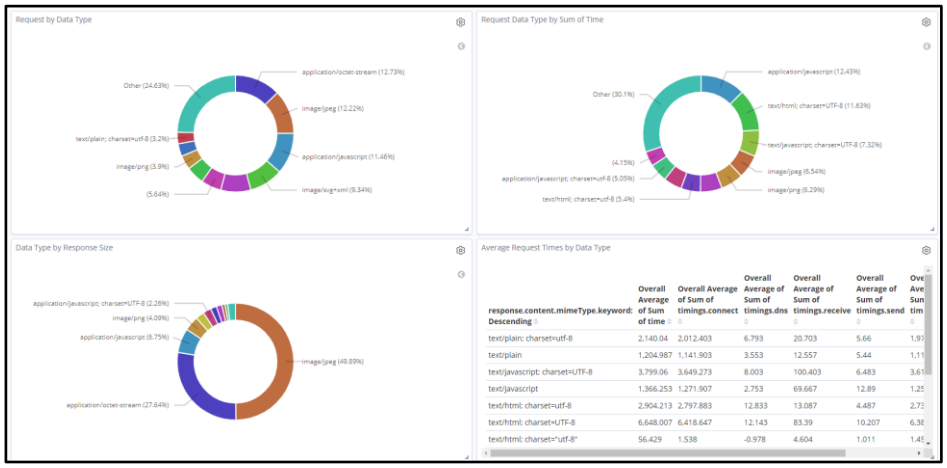

Figure 19 - Solent's Website Data Type Distribution (3-day range)

By viewing the bar graph in Figure 20, some concerns emerged about its usefulness for availability. As the tests are programmed to start every minute, there should be an even distribution of requests across time. Upon selecting a smaller time interval for comparison, the graph (Figure 21) updates accordingly and shows instead just singular requests that fail a couple of times a day.

The reason for this problem appears to be due to the way the graph was built, because, as the application gathered large amounts of individual tests, the application started to lag. Because the failed tests are relatively frequent, and there is no way to confirm the website availability at those times, it cannot be concluded if these are originated from the artefact or if the web page was in fact unavailable. Further analysis was done to see if the failed requests were presented in any pattern, but as no evident results were found, no conclusion was drawn.

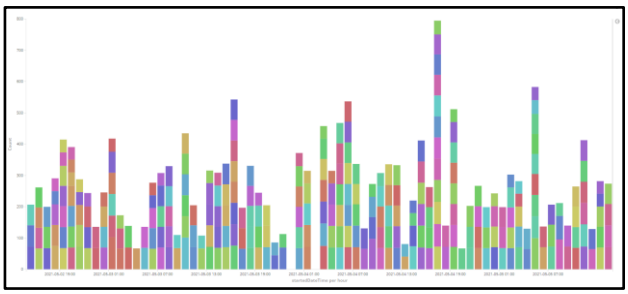

Figure 20 - Requests Over Time (3-day range)

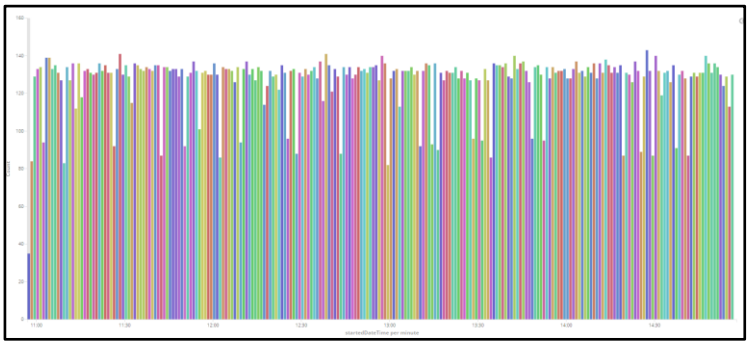

Figure 21 - Request Over Time (4-hour range)

Still, the application can gather large amounts of data which can be used to correctly monitor the website overall performance. Grouping these results for different time periods can also be used to learn about troubling requests or patterns for recurring performance slowdowns.

During the 3 days, the graph for problem detection seemed to have detected a few components that behaved weirdly as can be seen in Figure 22. When analysing the selected component, it showed an increase of about 6 times the average values for the top 3 requests.

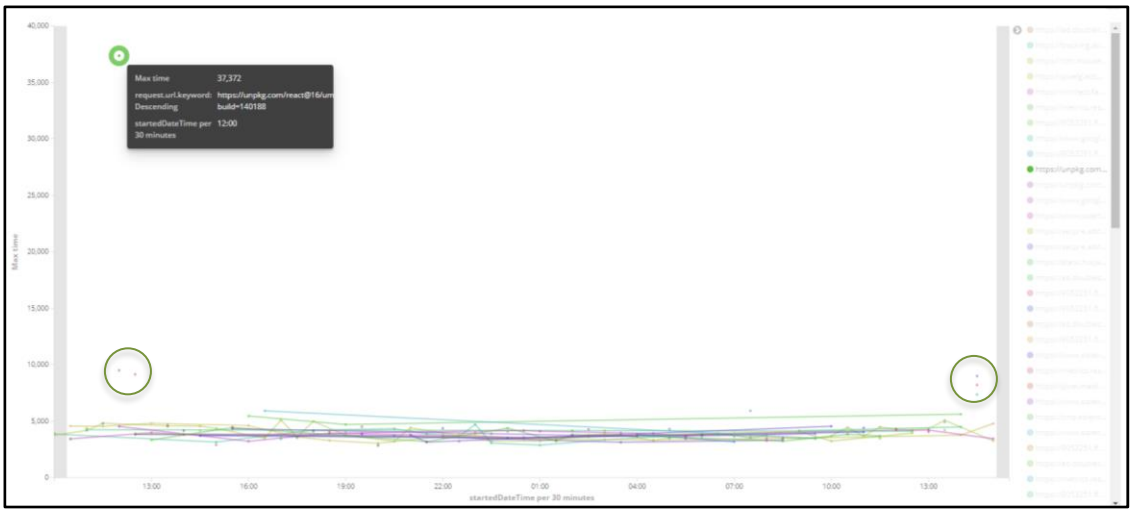

Figure 22 - Top 3 Request by Load Time with Highlighted Regression (5-hour range)

When looking at the performance graph (Figure 23) to see if the component affected the page load time, a clear increase can be seen in the wait time of request which might mean the problem could be originating from a CDN that might have had temporary delayed performance. Being able to easily

identify the request in question is a useful feature as system administrators can then act accordingly. In this case, the problem does not seem recurring, so it probably does not require changes but having the option to identify slow components like this gives a huge advantage for problem detection.

Figure 23 - Performance Graph VS Top 3 Requests by Load Time (1-hour range)

6.3.2. Performance Injections

Starting by looking at the three-hour interval where the injections were applied, the changes can be perceived using the performance time-series graph (Figure 24). The tests were realised between 10:30 and 13:30, with the first regression (non-optimized stylesheets) being inserted at 11:30, and the second (substitution of images for high quality, non-compressed ones), at 12:30.

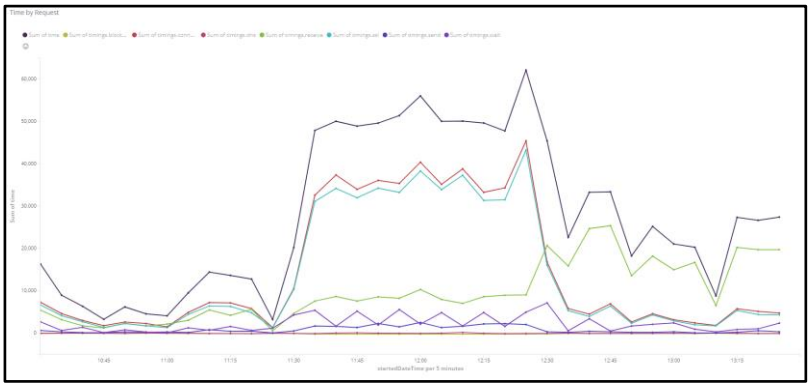

Figure 24 - 3 Hour Performance Test Graph

If the websites regressions did not affect the page so heavily, some of the other visualisation can help detect the presence of anomalies. In this test, the regressions were evident from the performance graphs, but the results can be complemented by looking at others like Figure 25, which clearly show different amounts of requests as well as the status of their responses. However, this does not help with finding the root cause of the regression, for that the graph in Figure 26 is being used which indicates the top 3 individual transactions by their time to load. From analysing the interval for this visualisation, it is clear the components taking up the page load time are changing when regressions were inserted.

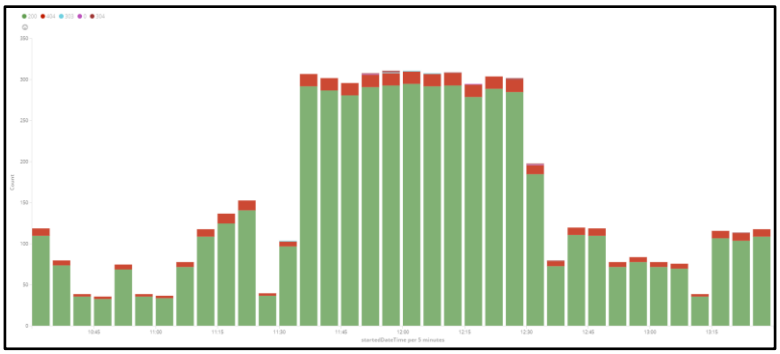

Figure 25 - 3 Hour Request Status Count Test Graph

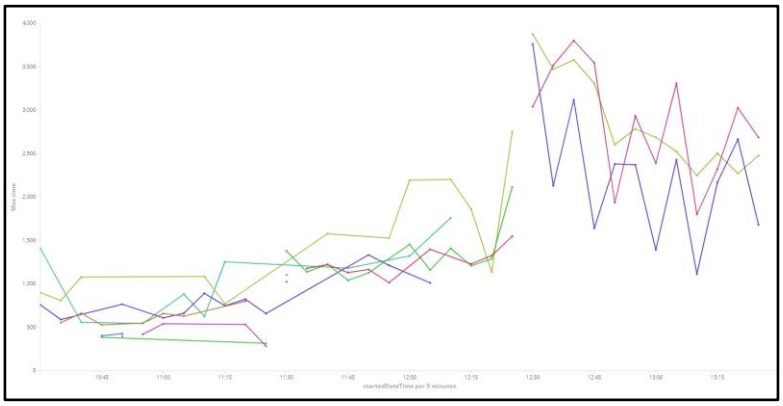

Figure 26 - 3 Hour Max Transaction Times Test Graph

An in-depth individual analysis can reinforce these findings. The sources of the data for the following conclusions can be seen in Appendix B.

A side-by-side comparison of the *Original* page versus *Injection 1* and *Injection 2*, clearly shows the differences between the page load time and the response body size. As *Table 6* tells us, there was an increase of about 0.6 seconds for the regressions, but as expected, the page size did not grow proportionally to the time. This is due to the differences in the injections, although it shows the size increase when the high-quality images were added (*Injection 2*), the size remained roughly the same as the *Original* when adding the stylesheets (*Injection 1*).

The time increase for *Injection 1* can instead be seen in the time to connect of the transactions to the stylesheets source, contrary to *Injection 2* where most of the load time is spent receiving requests.

	Total Time (ms)	Total Size (kB)
Original	395	2381
Injection 1	1024	2565
Injection 2	1032	7951

Table 6 - Injection Test Results for Time and Size

Comparing the pie charts for each version reinforces this as it clearly shows in *Injection 1*, that the longest resource to load by data type is now *text/css*, which does not take a share of the main resources for the *Original* page.

And for *Injection 2*, the most significant resources by both load time and size are the *image/jpeg*, which corresponds to the format of the images used.

Top Request By:	**Amount**	**Time**	**Size**
Original	application/octet-stream (41.59%)	image/png (44.09%)	application/octet-stream (75.03%)
Injection 1	application/octet-stream (27.38%)	text/css; charset=utf-8 (41.96%)	application/octet-stream (72.17%)
Injection 2	application/octet-stream (41.15%)	image/jpeg (59.92%)	image/jpeg (69.19%)

Table 7 - Injection Test Results for Data Type by Amount, Time, and Size

Finally, when conducting research on the specific components, instead of just the data type, the top requests at each time interval can be compared to see if, those too, can be detected directly from the visualisations. In the *Top 3 Transactions by Load Time Over Time* Graphs, it consistently shows the expected requests, confirming the previous findings that the tool can be used to identify the individual transactions affecting the system.

7. Conclusions

Web pages request can be used to gain lots of insights into the transactions between the browser and the hosting server. When transactions are grouped, they can clearly show performance for a determined page. The same results were not found for monitoring availability, as some tests have failed, and it was not possible to verify if the website was in fact down or if the problem originated from the artifact. Using the approach in this research, the results show that STM can very well be done by simulating user behaviour using the Selenium tool and gathering its transactions, the same way any user can directly in a browser.

Although results from the MVP (not packaged) were not recorded, their monitoring outcomes seemed to fluctuate more, as the development environment was still being used during monitoring. When done in a controlled environment the tests were more accurate, as there were fewer external variables affecting the system, this reduced the noise in the data making the results more accurate.

This was achieved by packaging and hosting the application which ended up being one of the most crucial and time-consuming steps in this project development. This severely helped gathering results as the application can just be deployed to online hosting and left running until its disk runs out of space. Monitoring can be done at any time since the application is always available and Kibana's interface can be accessed through a public URL.

The development of the application was an overall success and the analysis done on its accuracy reinforces this idea. Especially for monitoring the page load time, the outcomes were great, with almost a perfect score during analysis against the chosen tools.

There are still some worries regarding other aspects of monitoring like calculating the page size or showing its availability. Although it worked very well for the test against a real website, the results were not so trustworthy when it

came to the test page. Both the failed tests for a page with very few components and the incorrect calculation of page size, present some concerns about the validity of those parts of the application. These can of course be addressed by further testing and the expansion of the application.

For the analysis on problem detection, it was concluded that the tool can be used to identify individual components taking a strain to the system. These can be individually picked out from looking at the performance graphs and are normally sourced by one transaction that took way longer time than usual. Grouping other fields from requests can also be very advantageous as it can be used to display most metrics of websites, like what data types occupy most requests, their longest time to load, or how much page size they utilize.

Of course, there are some threats to the quality of this analysis as manually visualising the data can cause user induced errors to happen. Besides, the manual labour needed to point out the root causes of degradations can be dull and should be automated. The tool is by no means perfect as while gathering the results from the hosted application, Kibana and ElasticSearch data kept disappearing after 4 days, probably due to initial misconfigurations in Microsoft Azure. This was the reason for a shorter than preferred monitoring analysis of Solent's website.

There were also some problems regarding time management during development, as the proposed solution should be able to programmatically detect root causes of problems as well as classify its performance according to Google's metrics. There was already a big learning curve for the technologies used so the project was not expanded further than deployment.

The methodology to evaluate the web monitoring tool was inspired by the works present in the literature review with some adaptions to this project. The use of KPIs as the main performance metrics definitively set a good base for the results achieved. The evaluation methods, like the injection of performance

regressions and collection of results from different tools, were extremely valuable to the final outcomes.

Finally, the project can be concluded as a success as the monitoring tool MVP was built and expanded and can indeed be used to identify performance issues that can be encountered when entering a web page. The approach of user simulated tests was one of the key decisions to the results as it replicates real user experiences, but testing is still needed to assure its accuracy against different types of web services.

8. Recommendations

The artifact produced is still in early stages of development so it should certainly be expanded, also further testing is needed for other aspects of monitoring. Upon continuing development, the recommendations are as follows.

Starting with its current problems the application has a malfunction where Selenium tests sometimes fail on the test website; this could be due to the page being too small, but no conclusions were drawn at this point.

To gather more varied results, such as the ones produced by RUM, the hosting server location could be changed so it could gather user experiences from different locations. Another option is using a different browser with Selenium, so that it would generate similar results that can observe and compare performance for different user settings. The system could also be adapted to calculate web performance metrics like time to first byte, first contentful paint or total blocking time, which, besides being good metrics for page performance, could also be used for comparison against other tools.

The tests used to determine the application usability could also be expanded. As seen in the literature review, performance regressions can be caused by a variety of factors and their root cause can affect different parts of the web server. Further study on degrading changes could be done by adding new injections for testing different parts of the system, for example, the testing of user variables (i.e., browser, location, etc.), or network capacity (using load or stress tests).

Further analysis on other points of the system can still be done to ensure its performance monitoring accuracy. Although the outcomes show a very high efficacy rate for time prediction, it needs to be tested against a larger variety of websites, preferably with different systems backing them up. Performance testing should also last at least a week, so the visualisation can be used to find patterns or noticeable changes in the gathered data.

Another good addition would be to add a Selenium framework for mobile testing like Selendroid[28]. As most of web traffic comes from mobile phones, the system lacks the ability to test pages performances on those devices and would certainly increase the application use cases.

One last recommendation would be implementing the remaining use cases, automating problem detection and performance classifier. The technology initially researched to do this was the Apache Groovy programming language, which can dynamically query the data in ElasticSearch through its APIs and logically process it (like classifying results). The script could then be triggered at a desired time, like when new data is pushed to ElasticSearch or on a timed basis.

[28] http://selendroid.io/

9. Reference List

2 STEPS TEAM, 2019. *Synthetic Monitoring vs Real User Monitoring* Available from: https://blog.2steps.io/synthetic-monitoring-versus-real-user-monitoring

AHMED, T.M. *et al.*, 2016. *Studying the Effectiveness of Application Performance Management (APM) Tools for Detecting Performance Regressions for Web Applications: An Experience Report* Available from: https://ieeexplore.ieee.org/document/7832882

AN, D., 2017. *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/

CITO, J. *et al.*, 2015. Identifying Web Performance Degradations Through Synthetic And Real-User Monitoring. *Journal of Web Engineering*, 14(5-6), 414–442

CONTENTSQUARE, 2021. *Digital Experience Benchmark*. ContentSquare Available from: https://contentsquare.com/insights/digital-analytics-benchmarks/

CYBERSAFE, 2021. *Website Monitoring – CyberSafe* [viewed 8 April 2021]. Available from: https://cybersafe.ae/website-monitoring/

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

KAN, S.H., J. PARRISH and D. MANLOVE, 2001. In-process metrics for software testing. *IBM Systems Journal*, 40(1), 220–241

MAGSBY, D., 2019. *Why Should I Run My Selenium Tests in Headless?* Available from: https://smartbear.com/blog/selenium-tests-headless/

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

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

PRATT, M., 2019. *Jest - Elasticsearch Java Client* Available from: https://www.baeldung.com/elasticsearch-jest

SANDVINE, n.d. *Industry Whitepaper - Measuring Web Browsing Quality of Experience: Requirements for Gaining Meaningful Insight* Available from: https://www.sandvine.com/hubfs/downloads/archive/whitepaper-web-browsing-qoe.pdf

10. Bibliography

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

BECK, K. *et al.*, 2001. The Agile Manifesto

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/

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

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

HOBFELD, T., F. METZGER and D. ROSSI, 2018. Speed Index: Relating the Industrial Standard for User Perceived Web Performance to Web QoE. *2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX)*

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

HUSTON, T., n.d. *What Is Real-User Monitoring?* Available from: https://smartbear.com/learn/performance-monitoring/what-is-real-user-monitoring/

INTERNATIONAL TELECOMMUNICATION UNION, 2007. *Definition of Quality of Experience (QoE)* Available from: https://www.itu.int/md/T05-FG.IPTV-IL-0050/en

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

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?* Available from: https://www.keycdn.com/support/what-is-a-har-file

LADAN, M.I., 2011. Web Services Metrics: a Survey and a Classification. *International Conference on Network and Electronics Engineering*, 11, 97

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

MANHAS, J., 2013. A Study of Factors Affecting Websites Page Loading Speed for Efficient Web Performance. *International Journal of Computer Sciences and Engineering*, 1(3)

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, 2013a. Effects of Web Page Contents on Load Time over the Internet. *International Journal of Science and Research (IJSR)*, 2(9)

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

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

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/

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

SMARTBEAR, n.d. *What Is Synthetic Monitoring* Available from: https://smartbear.com/learn/performance-monitoring/what-is-synthetic-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, 2019. *Total Blocking Time* [viewed February 2021]. Available from: https://web.dev/lighthouse-total-blocking-time/?utm_source=lighthouse&utm_medium=devtools

11. Appendices

11.1 Appendix A: Calculating Accuracy Sources

Below are the screenshots taken from the different tools used to test the artefact monitoring accuracy. To be noted that values for Uptime were gathered from the sum of the transaction steps present in figures 29 & 35, Dareboost from their Time to (Consistently) Interactive in figures 31 & 37 and the Network tab from Finish time in figure 32 & 38.

The following images show the data gathered from Solent's website.

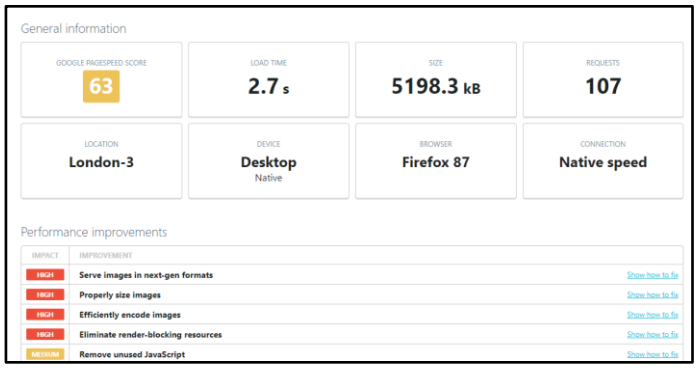

Figure 27 - Uptrends Solent Test Results 1

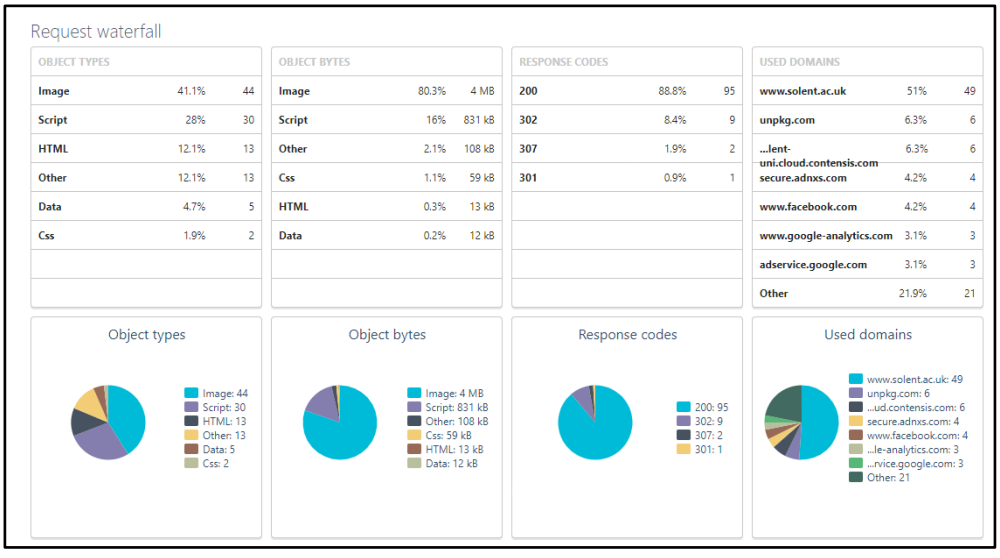

Figure 28 - Uptrends Solent Test Results 2

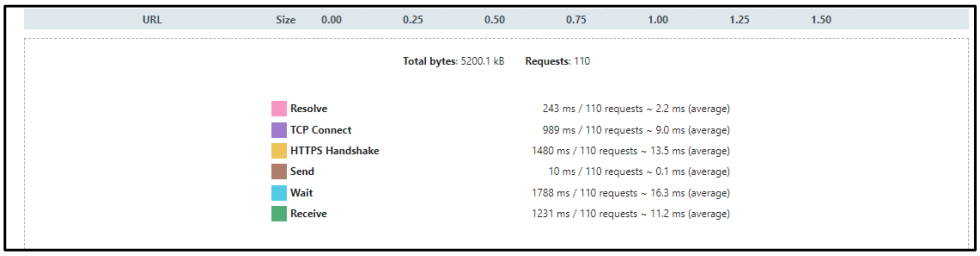

Figure 29 - Uptrends Solent Test Results 3

Figure 30 - DareBoost Solent Test Results 1

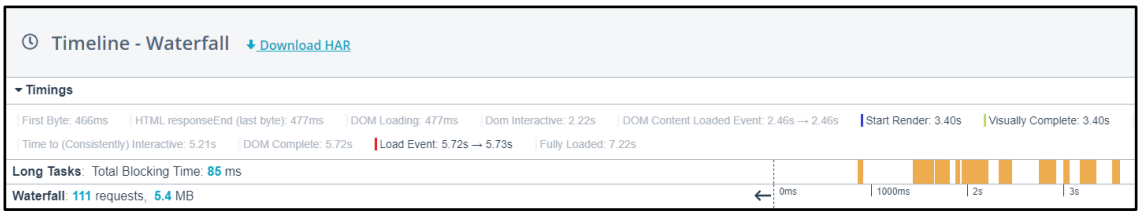

Figure 31 - DareBoost Solent Test Results 2

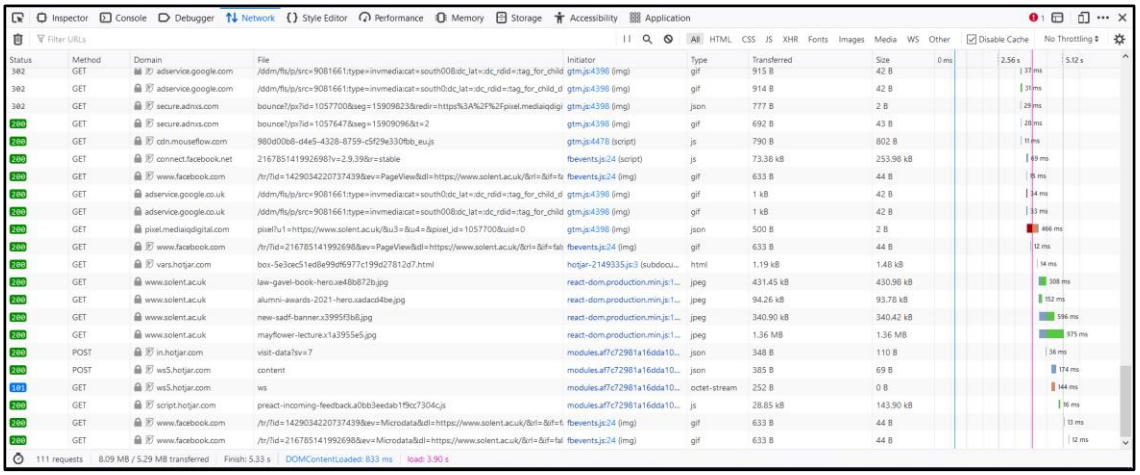

Figure 32 - Firefox Network Tab Solent Test Results

The following images represent the data gathered from the test website.

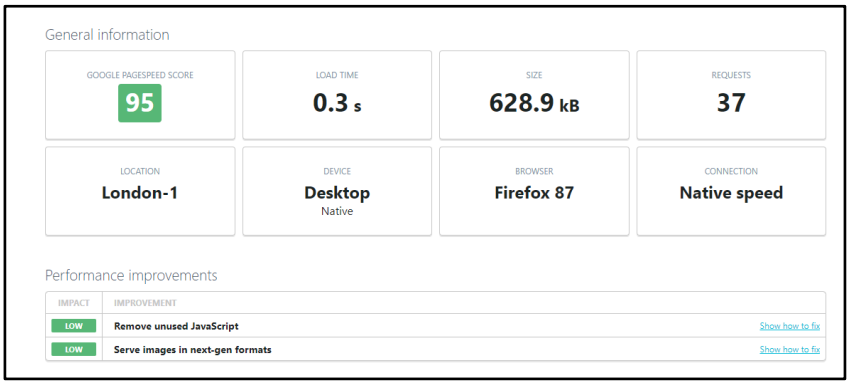

Figure 33 - Uptrend Test Website Results 1

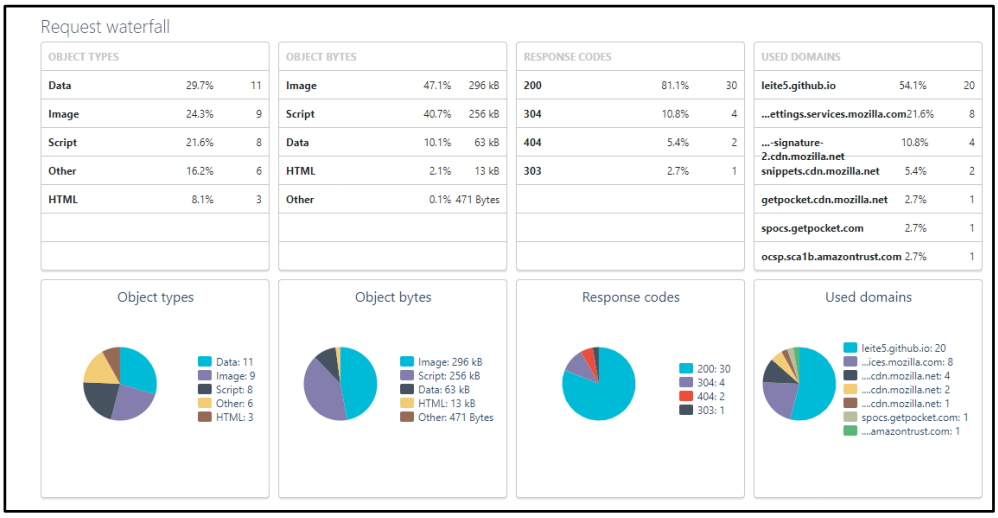

Figure 34 - Uptrend Test Website Results 2

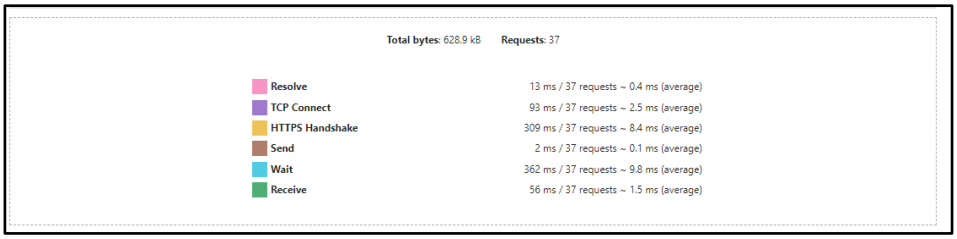

Figure 35 - Uptrend Test Website Results 3

Figure 36 - DareBoost Test Website Results 1

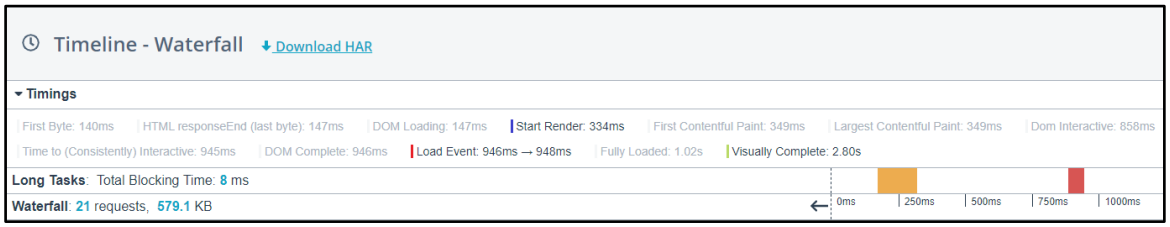

Figure 37 - DareBoost Test Website Results 2

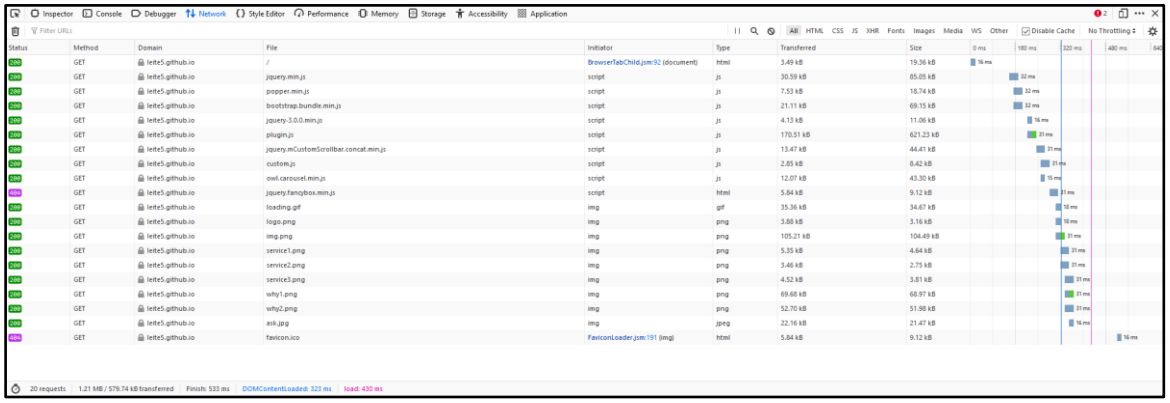

Figure 38 - Firefox Network Tab Test Website Results

11.2 Appendix B: Problem Detection Sources

Below are the screenshots taken from the different tools used to test the artefact monitoring accuracy. The first set of images show the monitoring results for the *Original* version of test page.

Figure 39 - Monitoring Results for Original Version - 1

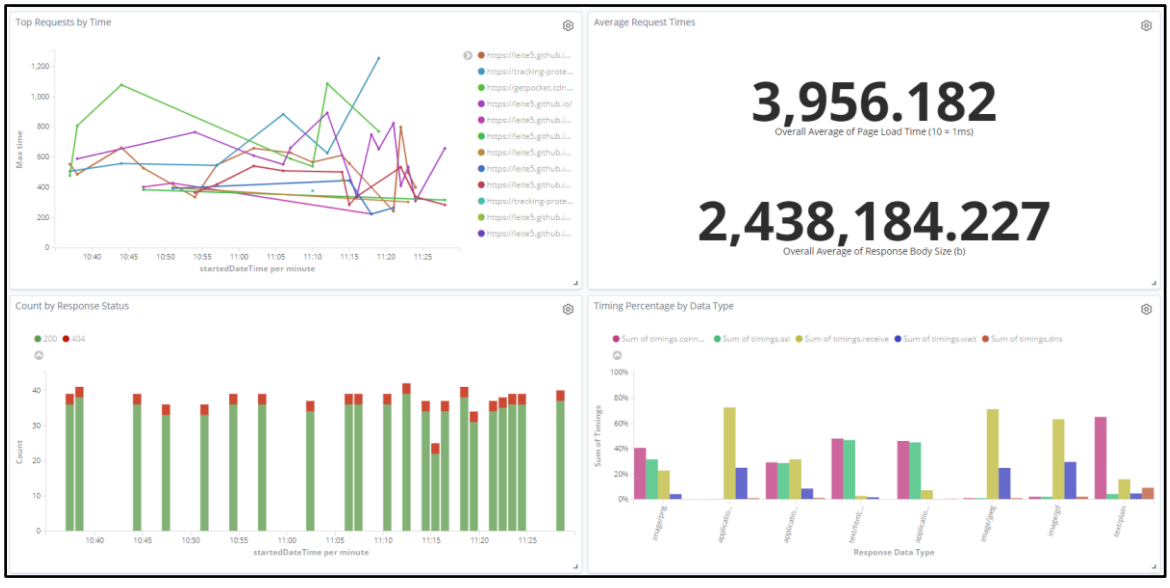

Figure 40 - Monitoring Results for Original Version - 2

The second set shows the results for the page with *Injection 1*.

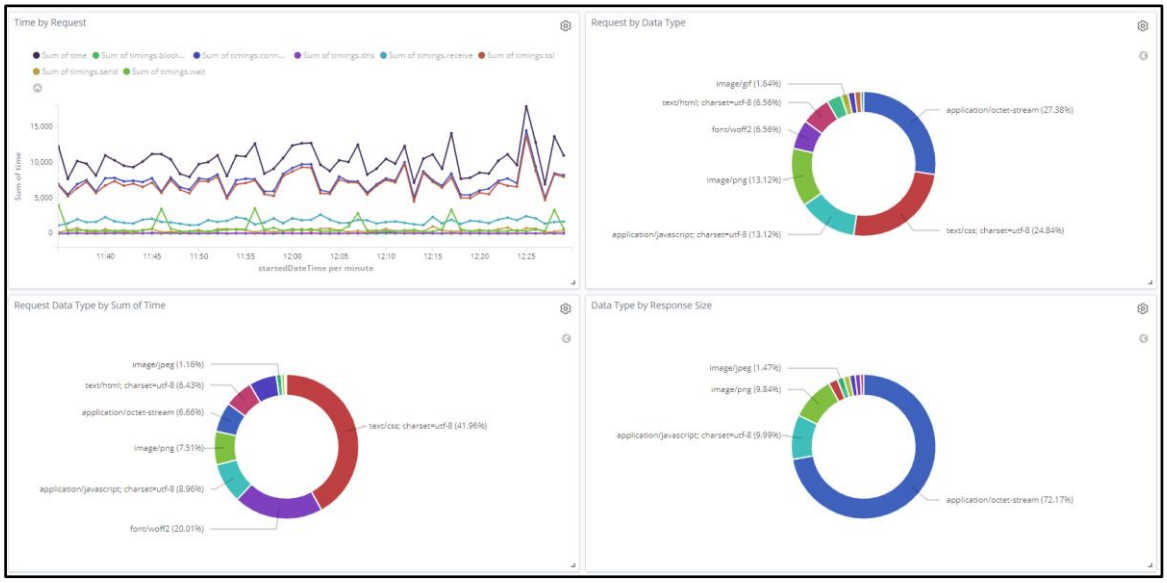

Figure 41 - Monitoring Results for Injection 1 Version - 1

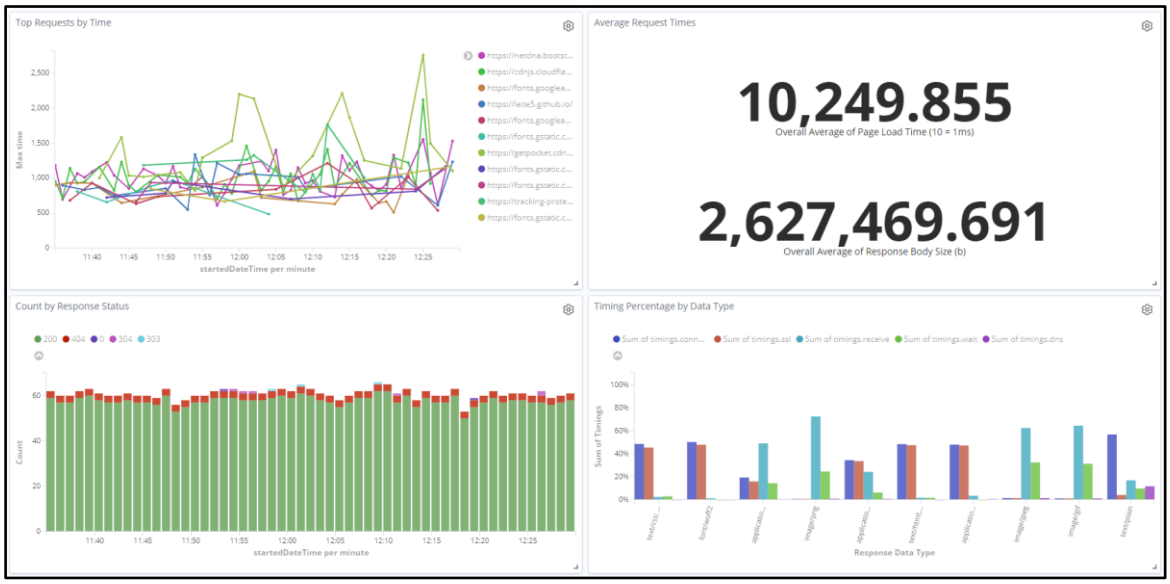

Figure 42 - Monitoring Results for Injection 1 Version - 2

And finally, the third set shows the results for *Injection 2* monitoring.

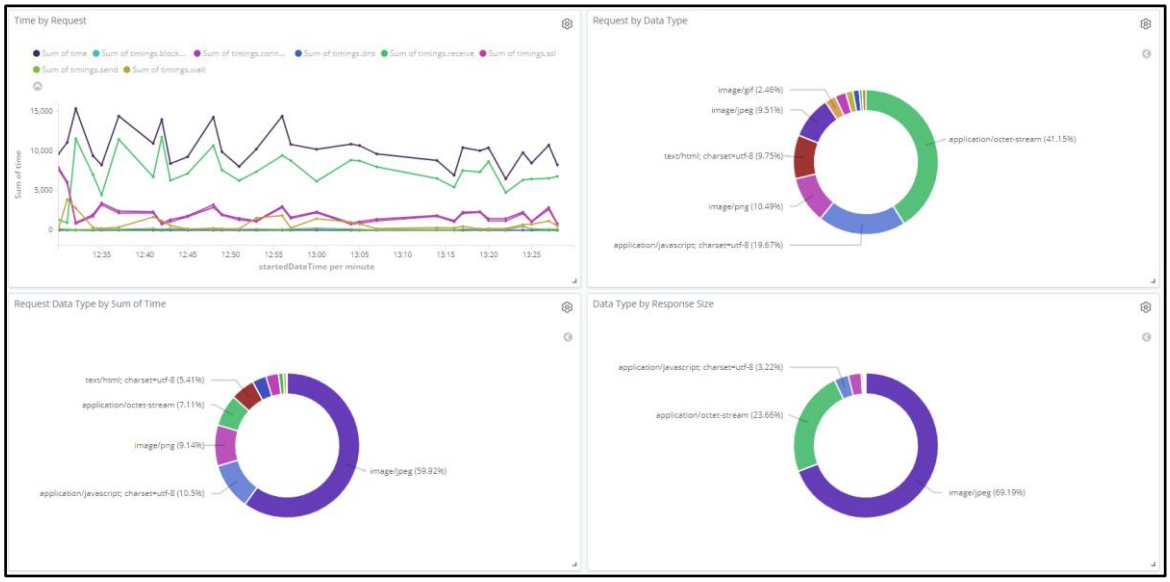

Figure 43 - Monitoring Results for Injection 2 Version - 1

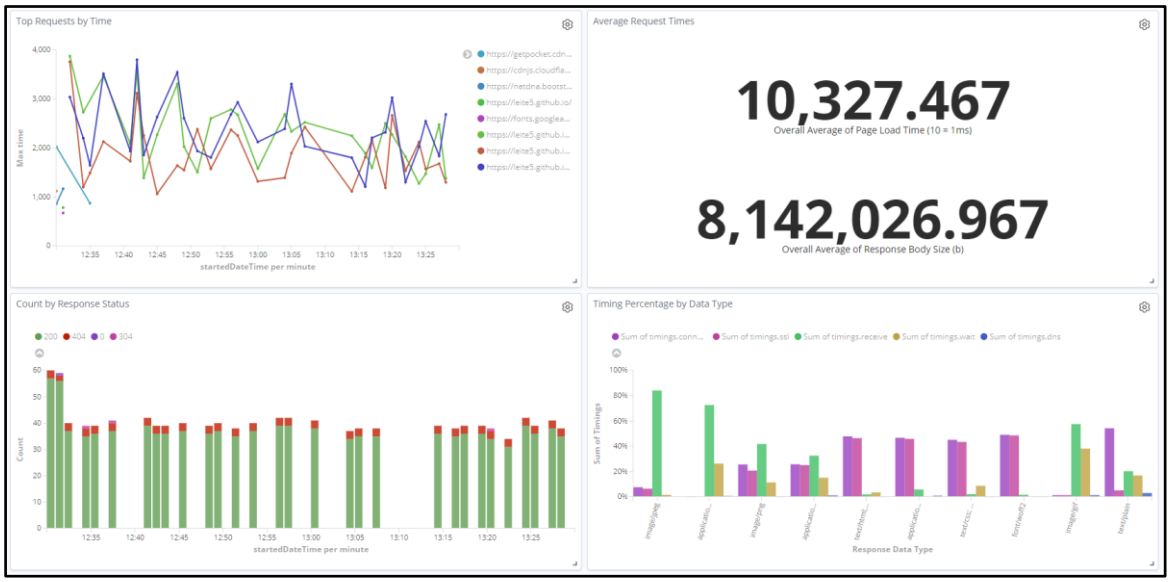

Figure 44 - Monitoring Results for Injection 2 Version - 2