An Empirical Study to Identify Frontend Performance Bottlenecks Using Developer Tools

Khavin Krishnan Kalpana
Computer Science & Applications
Virginia Tech
Blacksburg, USA
khavin@vt.edu

Krithika Gunasekaran Krishnaswamy
Computer Science & Applications
Virginia Tech
Blacksburg, USA
krithikagk@vt.edu

***Abstract*— In the fast-growing digital world, the websites we use are expected to possess good performance and smooth user experience for user satisfaction and maximum user retention. This paper studies the frontend performance of 27 popular websites, aiming to identify the major performance bottlenecks of those sites during the initial loading phase. We conducted our testing in a simulated slow 3G connection since 3G is still the most common network in many developing countries. In this study, we focused on assessing the performance of the sites containing heavy multimedia content like images, videos, CSS, and Javascript, since these sites usually take a longer time to load on slow internet connections. We use Chrome's developer tools to identify the requests that might be impacting the initial loading time of a site. We believe that every individual irrespective of where they are deserves a smooth and fast user experience. This research contributes to the field of web development by providing actionable insights that a developer can take to enhance the loading performance of their site.**

***Index Terms*—Web Performance, Frontend Bottlenecks, Developer Tools, Chrome Browser, Website Optimization, User Experience.**

I. Introduction

As the digital technology landscape keeps evolving, more business is carried out online where the performance of a site is now more important than ever before in terms of keeping users engaged on your site, and achieving business goals. While the internet is becoming more ubiquitous, especially in places that were previously disconnected, there remains a significant gap in the connection speeds around the world like the rural areas of several developing countries that still face connectivity issues due to slow 3G connections. The purpose of this research is to investigate the frontend performance of some of the popular news websites and highlight areas of problems, or bottlenecks, that could affect the user experience of people using low-bandwidth connections.

The evaluation metrics selected for this analysis are as follows: First Paint Time, First Meaningful Paint Time, Largest Contentful Paint Time, total requests, cached requests, late requests, important requests, priority changed requests, and render blocking requests.

1. First Paint Time (FP): The first paint time is the time between when the user loaded the page and when the first pixel is rendered on the screen.

2. First Meaningful Paint Time (FMP): FMP is the instant when a large layout change has happened on the screen area visible to the user. First Meaningful Paint time is often considered inaccurate.

3. Largest Contentful Paint Time (LCP): Largest Contentful Paint Time is the moment the main content of a site is rendered on the screen and this is usually when a user starts to interact with the website. This is the metric widely used to measure the initial loading performance of a site.

4. Total Requests: These are the total number of HTTP requests initiated by the browser to load all the resources like HTML, CSS, JavaScript, images, and more that are needed to render the complete webpage.

5. Cached Requests: The requests loaded from the browser's cache storage instead of the server are called cached requests. Browsers usually cache content like images, CSS, JS, etc.,

6. Late Requests: This is a term created for the purpose of this project. Late requests refer to requests that were initiated late after the browser received the response for the initial HTML document.

7. Important Requests: This is a term created for the purpose of this project. Important requests are those requests that have priorities ranging between Very High, High, and Medium priority.

8. Priority Changed Requests: These are the requests where the priorities are changed after the request is initiated. Sometimes the browser changes the priority of a request, as more information becomes available. For example, if a browser finds that an image is going to be rendered above the fold (user-viewable area), the browser changes the priority of the request to High, so that the image can be loaded faster.

9. Render Blocking Requests: Render blocking requests are resources (typically CSS and JavaScript) that prevent the browser from rendering the page until they are fully loaded and executed. An optimized website should have zero render-blocking requests.

We are using the performance trace data provided by Chrome's developer tool for our analysis. In this project, we are specifically focusing on the initial loading time of a site. The initial loading time is very important for user retention. Using the Largest Contentful Paint time or LCP we can identify when the largest block of an image or a text block is rendered on the screen. This is when a user will start interacting with the site. So we are analyzing the events that happened before the Largest Contentful Paint time to identify the requests that might be causing the slowdown. We have two main goals in this project: 1) Study the distribution of resource types that are causing major performance bottlenecks in popular websites. 2) Develop a tool that offers developers actionable insights to enhance the loading performance of a website.

II. Related Work

In 2019, Jake Archibald (an engineer at Google 2019) analyzed the performance of different Formula 1 team sites to find which Formula 1 team has the fastest site. He published the findings in his blog[1] - https://jakearchibald.com/. He found that major sites like renaultsport.com, and mclaren.com took more than 30 seconds to load. This kind of delay will have a negative effect on user experience and the majority of the users will not wait for the site to load and these users might move on to other sites. So, it is paramount to increase the performance of the sites to increase the number of page visits.

III. Approach

In this study, we aim to analyze the top 25 most popular news websites to understand and evaluate their frontend performance. We got the top 25 list from an article published by pressgazette.co.uk. In addition to the top 25 sites we also used the sites of popular Formula 1 teams like McLaren and Ferrari. So in total, we analyzed 27 sites. We chose these websites because they are rich in multimedia content like images, videos, etc., Initially, we planned to perform the analysis manually by analyzing the performance tab of Chrome's developer tools. We quickly found that manual analysis is a tiring and time-consuming process. The manual process is also very imprecise. So, we devised a way to automate the analysis and report creation.

A. Data Collection

We started the analysis by opening the Chrome dev tools. Here we adjusted the network throttling settings. We kept the CPU throttling as "None" which is the default value and we changed the network throttling to "Slow 3G". We chose the screen dimensions of Pixel 7. This is because the majority of users in developing countries access the internet via their mobile phones. Then we used the refresh button present inside the performance tab to reload the page. This will trigger performance data collection. The performance data is called Performance Trace data.

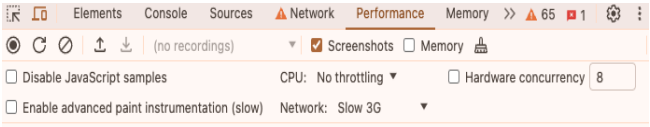

Fig. 1. Configuration settings of the performance tab.

B. Manual Analysis

In the manual analysis, we were specifically looking for late requests. We manually noted down the request start time and end time to identify the late requests. However, the manual analysis was tiresome and error-prone. The following are some of the issues we faced during the manual analysis:

1. We had to zoom in on the developer tools to inspect a request. Because of the zoom-in action, we lost context around where the other events were on the screen. For example, we were not able to identify when the request for index.html resource ended. This is crucial for identifying requests that were initiated late.

2. It became difficult to note down the request initiated time for each request manually. As the number of requests grew, the errors increased as well.

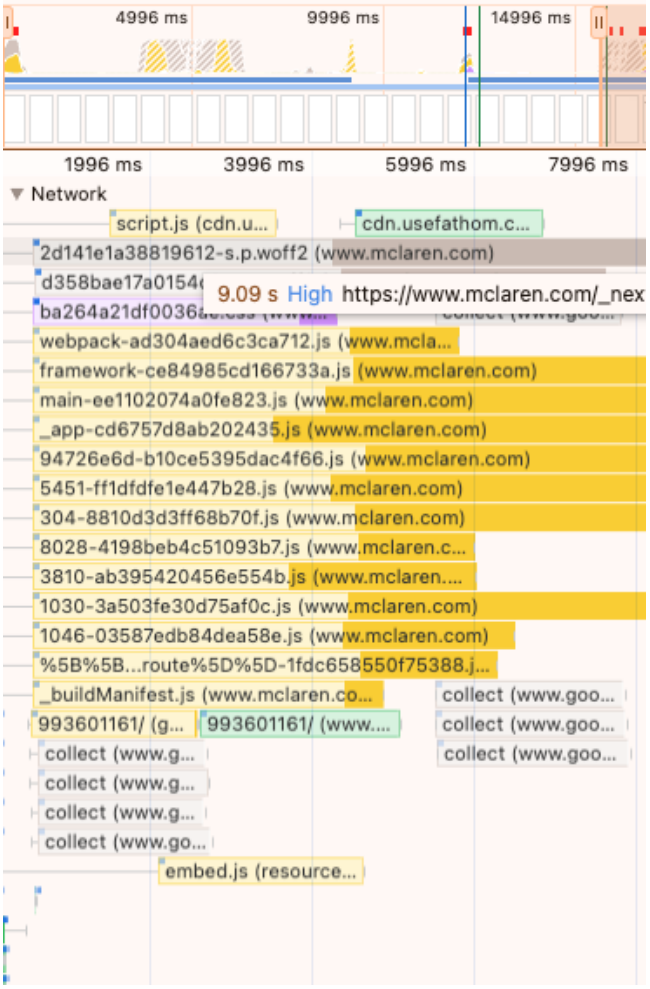

Fig. 2. Events in performance tab

3. Most of the modern sites initiate more than 100 resource requests. The browser started to freeze when we were constantly zooming in and out of the developer tools. This hampered our analysis.

4. A lot of requests were small in duration. This made us miss these requests during our manual analysis.

5. The amount of time it took to analyze a single website was around 3 to 4 hours. Even then, most of our analysis was imprecise because we missed out on lots of small requests.

C. *Automated Analysis*

Our automated analysis phase still required manual intervention to gather the data. The steps mentioned in the Data Collection section were repeated. Chrome's developer tools have an option to export the performance trace data. The data will be exported in JSON format.

For each site, we performed the data collection twice. When a site is reloaded the browser will be able to take advantage of the cached data. Usually, a browser can store images, videos, CSS, and Javascript in its cache database. We performed the data collection twice to identify whether the metrics like First Meaningful Paint time (FMP) and Largest Contentful Paint time (LCP) improved or not.

The collected trace data is given as input to a Node.js script. The script performs the analysis and the output report is created.

IV. Implementation

A. *Reading data*

We developed a Node.js script to perform the analysis. The script will start its execution by reading the directory "trace_data". The directory "trace_data" contains the performance trace data of all the tested sites in JSON format. The script will go through the files initially to identify the tested site URL. Then the script will group the files by the site URL tested. Each site will have two files.

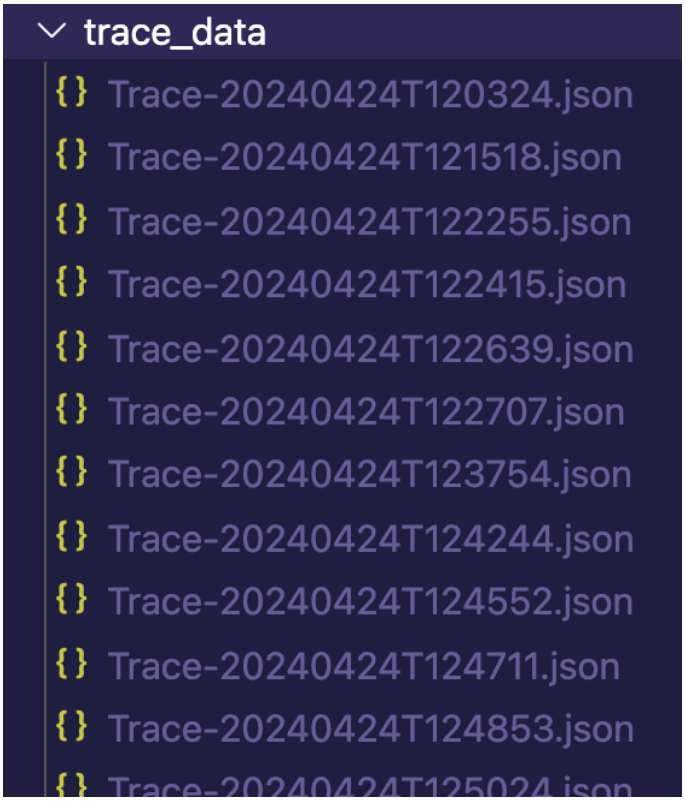

Fig. 3. Trace_data directory containing the input trace data files.

B. *Methodology to calculate the metrics*

1. First Paint Time (FP): This is an event emitted by the browser. The logic is to look for an event named “firstPaint”. Fetch the timestamp of the first event with the above name.

2. First Meaningful Paint Time (FMP): This is also an event emitted by the browser. The logic is to look for an event named “firstMeaningfulPaint”. If the event is not found, look for an event named “firstMeaningfulPaintCandidate”. Fetch the timestamp of the event that occurs last with the above names.

3. Largest Contentful Paint Time (LCP): This is also an event emitted by the browser. The logic is to look for an event named “largestContentfulPaint::Candidate”. Fetch the timestamp of the event that occurs last with the above name.

4. Cached requests: The trace data has a boolean field named “fromCache”. If the value is true, then this means that the request is loaded from the browser cache.

5. Late requests: To identify the late requests present in the network trace we are only looking at requests that got initiated after the initial html document loaded and before the Largest Contentful Paint time (LCP) event. We are also filtering out the requests that were initiated by the initial HTML document. For example, let’s denote the initial HTML document as A. The requests initiated by the initial HTML document are denoted as B. The requests initiated by B are denoted C. All the requests initiated by C are denoted as C’. We continue this until the LCP event is reached. Here we aim to identify the requests C, C’, and so on. These requests are called late requests.

6. Important requests: Here our aim is to identify late requests with Very High/High/Medium priority. The “priority” field is used to identify these requests.

7. Priority Changed requests: Here our aim is to identify late requests whose priority got changed by the browser. The field “priorityChanged” is used to identify these requests.

8. Render-blocking requests: Our aim here is to identify late requests which are render-blocking. The field “renderBlocking” is used for this metric.

C. *Report Creation*

For each site, a separate report is generated. Each report contains two sections. The first section corresponds to the trace data where all the resources were loaded from the server. The second section corresponds to the trace data where some of the requests were loaded from the browser cache.

Fig. 4. The output directory with generated reports.

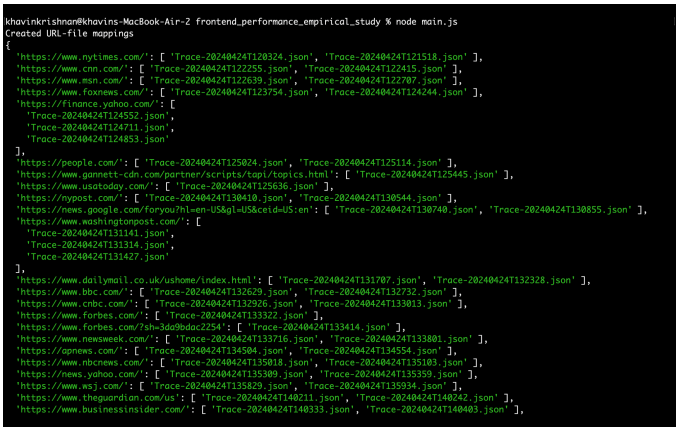

Fig. 5. The console log-1.

The top section of the report contains metrics like the number of late requests and, the number of priority-changed requests, etc., Information about each request can be found in the next section.

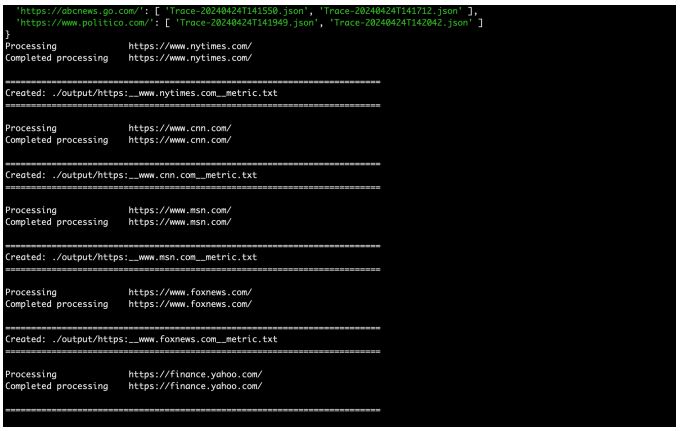

Fig. 6. The console log-2.

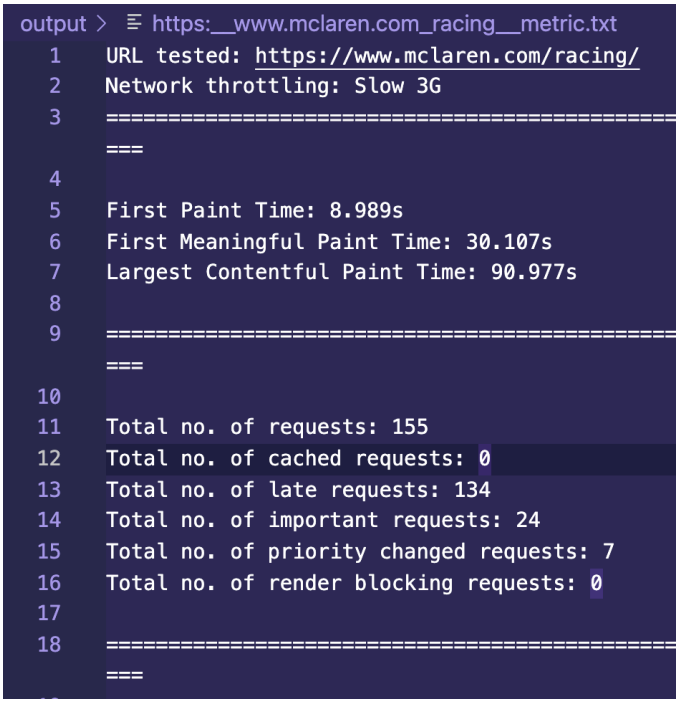

Fig. 7. The top section of mclaren.com/racing report

Public Repository where the code is hosted: https://github.com/khavin/frontend-performance-analysis-tool

V. Results

We created the distribution for different resource types. The field "resourceType" which is present in all the resource requests is used for this purpose. We found that the late request belongs to one of the following eight resource types: Script, Image, Document, Stylesheet, Font, Manifest, Media, and Other. Media type refers to video or audio elements. Other resource types include requests to analytics services like googletagservices.com, advertisement services like Google Ads, doubleclick.net, amazon-adsystem, etc.,

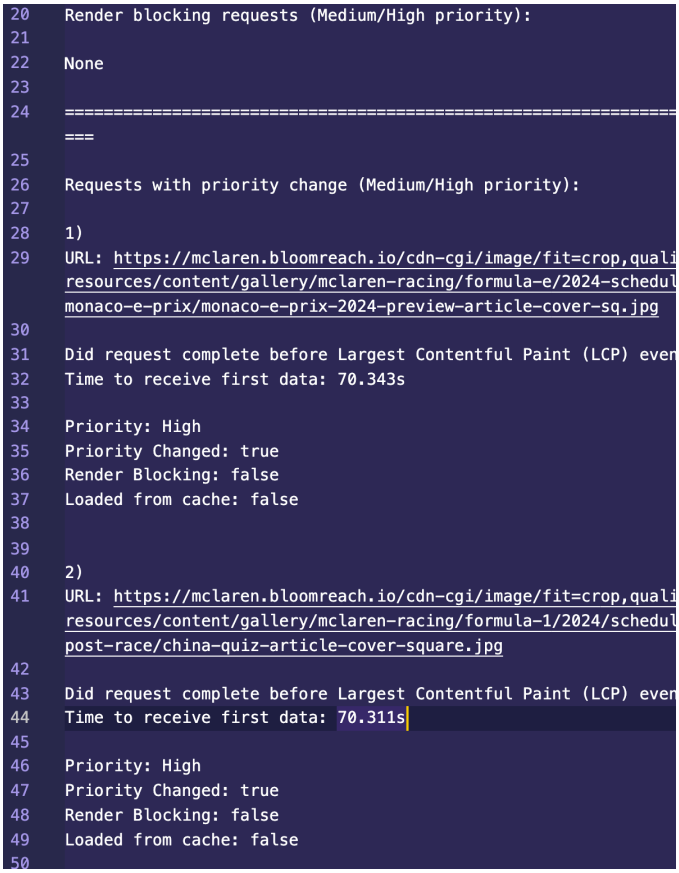

Fig. 8. Info about individual requests.

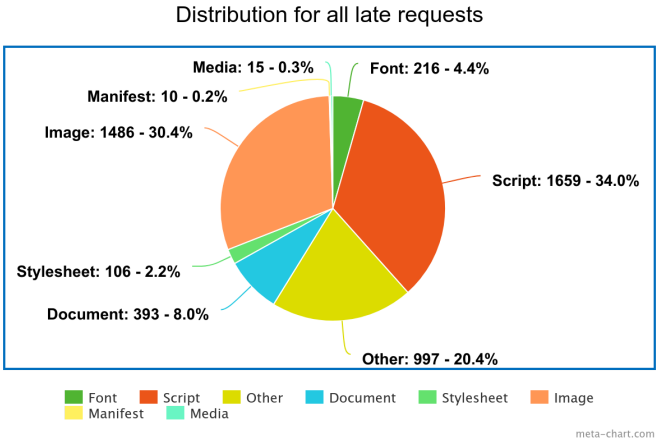

Fig. 9. Distribution of resource type in all late requests

Images and scripts contribute to 64.4% of all late requests. The majority of late requests with high/medium priorities are

in the “Other” category. This means that Ads are loaded with high priority.

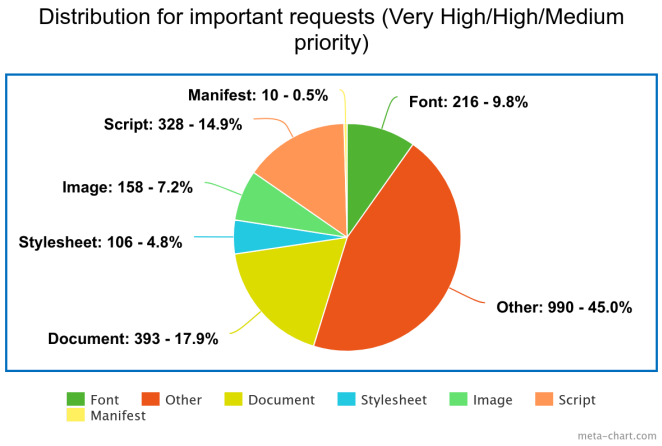

Fig. 10. Distribution of resource type in important late requests.

All the priority changed requests are of resource-type image. The browser will change the priority when it discovers that the resource will be rendered in the area visible to the user. The render-blocking requests are of type Stylesheet or Script. The render-blocking requests are very harmful to the user experience as the browser will stop responding to the user's actions. A well-optimized site should have very few or zero render-blocking requests.

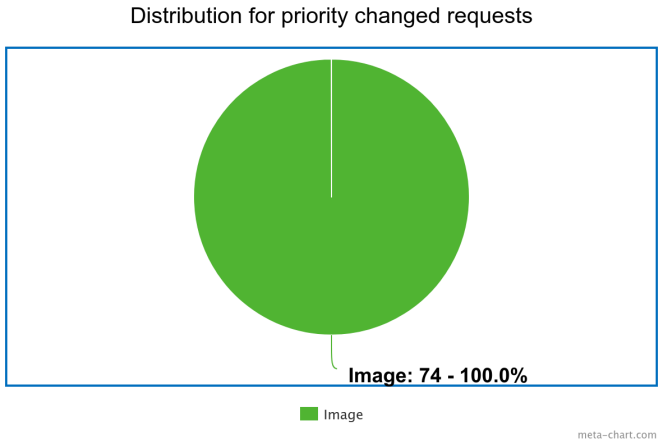

Fig. 11. Distribution of resource type in priority changed late requests

VI. Evaluation

To evaluate the correctness of the generated report, we decided to contact the developers and provide the generated report. We went through the generated reports and emailed the developers whose sites had a large number of late requests. We did not get any feedback from the developers as of now. We will update the report if we receive any feedback in the future.

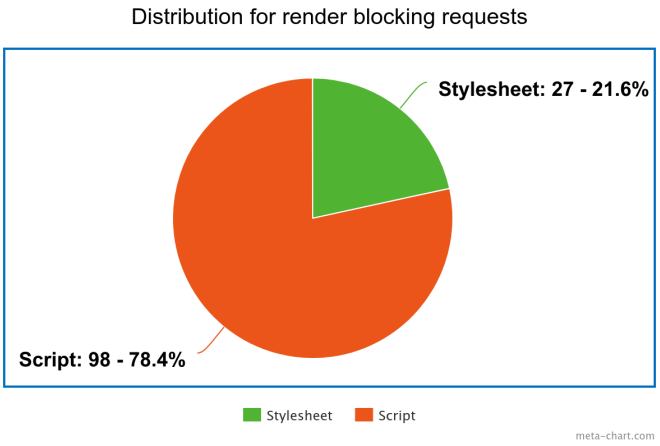

Fig. 12. Distribution of resource type in render-blocking late requests

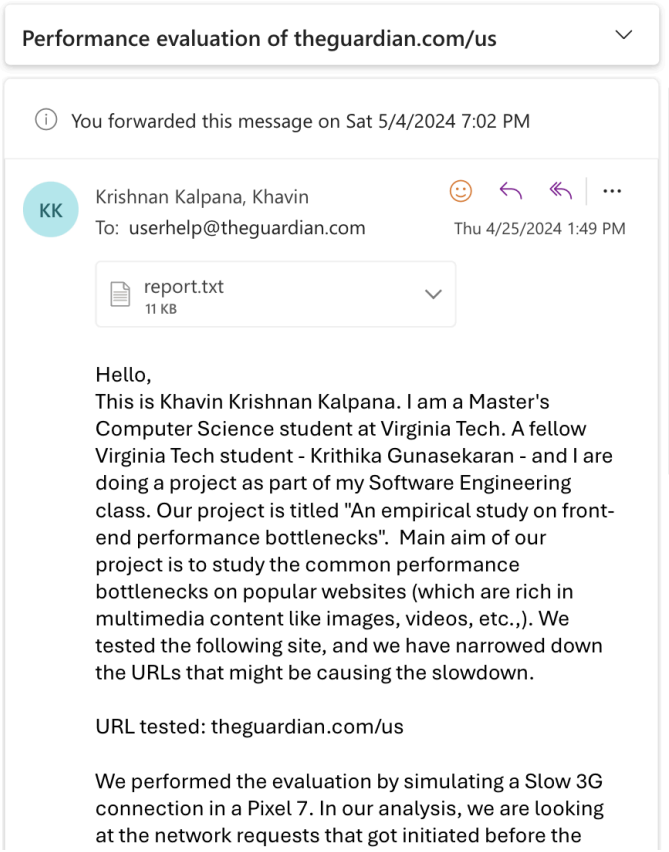

Fig. 13. The email sent to The Guardian

Fig. 14. The Guardian's response

VII. Future work

As of now the data collection part is done manually by going to the performance tab and downloading the trace JSON file. This workflow can be automated by Selenium or by using automation tools like Microsoft RPA (Robotic Process Automation).

VIII. Conclusion

As part of this project, we accomplished two tasks: 1. We did a study on resource types causing the performance bottleneck on popular sites. 2. To facilitate the study we created a tool to analyze the sites. However, this tool can be used by individual developers as well to identify the requests that are causing the bottlenecks on their site. We believe this project can provide a good starting point for developers to increase the frontend performance of their sites.

IX. References

[1] Jake Archibald 2019, *Who has the fastest website in F1?, Jakearchibald website*, accessed 10 February 2024, <https://jakearchibald.com/2019/f1-perf/>.

[2] Aisha Majid 2024, *Top 50 biggest news websites in the world: Newsweek doubles visits year-on-year in March, Press Gazette website,* accessed 4 March 2024, <https://pressgazette.co.uk/media-audience-and-business-data/media_metrics/most-popular-websites-news-world-monthly-2/>

[3] Google 2019, *First Meaningful Paint, Chrome for Developers website*, accessed 26 February 2024, <https://developer.chrome.com/docs/lighthouse/performance/first-meaningful-paint>

[4] Philip Walton and Barry Pollard 2024, *Largest Contentful Paint (LCP), Web.dev website*, accessed 19 March 2024, <https://web.dev/articles/lcp>

[5] Mozilla 2024, *webRequest.ResourceType, Resources for developers by Mozilla Website*, accessed 23 March 2024, <https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest/ResourceType>

[6] Google 2019, *Eliminate render-blocking resources, Chrome for Developers website,* accessed 25 March 2024, <https://developer.chrome.com/docs/lighthouse/performance/render-blocking-resources>

[7] Paul Irish 2016, *Trace Event Format,* accessed 2 April 2024, <https://docs.google.com/document/d/1CvAClvFfyA5R-PhYUmn5OOQtYMH4h6I0nSsKchNAySU/preview>