# Integration of PerfumeJS Plugin with Android Runner

**Author:** Khalil Mayyasi

khello1996@gmail.com

2644550

**Supervisor:** Ivano Malavolta

August 2020

# Abstract

Mobile devices generate high amount of web traffic through web applications. Due to the high usage, the performance of mobile web applications must be maintained. Therefore, developers should start focusing on designing/enhancing web application for mobile devices while ensuring that their performance doesn't decrease. This paper implements a PerfumeJS plugin which aims to measure several performance metrics for mobile web applications. Additionally, the paper performs an experiment to observe the performance metrics generated by the plugin. The experiment is performed on 100 web applications randomly selected for which several performance metrics are collected. The result of the experiment shows the four performance variables totalTime, downloadTime, FP and FCP to have an average of 62.6452, 31.3084, 1865.36 and 1886.31 *ms* respectively. In conclusion, developers are able to use the plugin and the result of the experiment to better understand some of the performance metrics of mobile web applications.

# 1 Introduction

## 1.1 Background

The demand for mobile devices has increased in the past few years. A recent report estimates the number of mobile subscriptions is around 8 billion in 2020 [1]. Additionally, mobile devices account for more than half of web traffic worldwide. In the second quarter of 2020, it is found that mobile devices generate 51.53% of a website traffic [2]. To cope with the high usage, the performance of mobile web applications must be maintained. Therefore, developers needs to focus on optimizing and improving web applications for mobile devices while ensuring that their performance does not diminish.

Various tools and libraries are available that allow developers to assess the performance of a web page on mobile devices such as PerfumeJS library [1]. PerfumeJS is a web performance monitoring library that reports field performance data of your web application. This library collects performance metrics such as *Navigation Timing*, *Network Information*, *Storage Estimate*, *First Paint* and *First Contentful Paint*. To automate the performance assessment of mobile web application android-runner framework can be used. Android-runner is a tool used to automatically execute measurement based experiments on native and web apps running on android devices[2]. This tool contains several profilers collecting several different measurements such as energy consumption, CPU usage and other measurements.

## 1.2 Objective

The **objective/goal** of this paper is to extend the android-runner tool with a new PerfumeJS plugin profiler to measure the performance metrics collected from the perfumejs library. Additionally, we will perform an experiment to inspect the performance metrics when the web application is loaded according to the plugin. The experiment is performed on 100 web applications and the performance metrics including Navigation Timing, Network Information and Storage Estimation, FP and FCP are inspected.

The **result** of the experiment shows that four variables provide useful performance measurements which are totalTime, downloadTime, FP and FCP. The variables are observed to have a mean of 62.6452, 31.3084, 1865.36 and 1886.31 milliseconds respectively.

The **target audience** of the paper are developers whose aim is understanding the performance of mobile web applications. The paper supports them by providing a plugin that can be used to

[1] https://zizzamia.github.io/perfume/

[2] https://github.com/S2-group/android-runner

measure some performance metrics of web applications. Additionally, the paper supports them by performing an experiment and providing the results of the performance metrics.

## 2 PerfumeJS Profiler Implementation

To collect metrics assessing the performance of mobile web applications, we implemented the PerfumeJS profiler. For this profiler to work, the mobile web applications used in the experiment to be assessed require to be prepared before using the profiler. First, the PerfumeJS library needs to installed. Then, the web applications to be assessed need to be injected with the code shown in Listing 1 within its "*index.html*" file. This injected code will use the PerfumeJS library to measure five performance metrics until the web application is fully loaded, then it sends an object containing the measurement to an http-server. To inject this code "AddJS.py" can be executed using the command in Listing 2. The "Readme" file in the repository provides additional details for the profiler preparation and setup[3].

Listing 1: Injected PerfumeJS code

```
<script src="/node_modules/perfume.js/dist/perfume.umd.min.js"></script>
<script>perfumeResults = [];
function xml_http_post(url, data, callback) {var req = new XMLHttpRequest();
req.open("POST", url, true);
req.send(data);}
const perfume = new Perfume({  analyticsTracker: (options) => {
const { metricName, data, eventProperties, navigatorInformation } = options;
perfumeResults.push(options); } });
function load_log() {
setTimeout(function(){ objectToSend =
"{'perfumeResults':"+JSON.stringify(perfumeResults)+"}";
xml_http_post("http://IP:8080/",objectToSend,null); },5000); };
window.addEventListener ?window.addEventListener("load",load_log, true) :
window.attachEvent && window.attachEvent("onload", load_log);</script>
```

Listing 2: AddsJS.py execution

```
$ python3 AddJS.py path_to_directory_containing_all_WebApps http://IP:8080/
```

The profiler is implemented to create a thread that start a http-server at profiler startup. The http-server stays running throughout the experiment duration to receive the perfumejs object sent

[3]https://github.com/KhalilMay/android-runner/tree/master/AndroidRunner/Plugins/perfume$_js$

from the injected code. Then, the profiler collects and writes the desired metrics specified in the configuration to the output directory of the experiment. Furthermore, an example of the profiler setup in the configuration file is shown in Listing 3 and the supported metrics by the plugin are shown in Table 1. Finally, the server is closed when the experiment is finished or terminated manually.

Listing 3: PerfumeJS profiler configuration

```
"profilers":{
    "perfume_js": {
      "metrics":["fp","fcp","storageEstimate","navigationTiming","
          networkInformation"]
    }
```

Table 1: Supported Metrics

| **Metrics** | **Description** |
|---|---|
| navigationTiming | collects seven performance metrics describing the life and timings of a network request |
| networkInformation | collects four metrics describing network information |
| storageEstimate | collects five metrics describing data about storage used by the web application |
| fp | collects the time it takes the browser to render any visual aspect different from what was on screen before the navigation to the web page |
| fcp | collects the time it takes for the browser to render the first bit of content from the DOM (Document Object Model) |

# 3 Experiment Definition

To validate the functionality of the profiler, we need to perform an experiment to inspect the result of the performance metrics. In this section, we will define the goal of the experiment.

## 3.1 Goal

The goal of the experiment is defined using the *Goal-Question-Metric* approach [3][4]. The goal is to analyze the selected web applications for the purpose of evaluation with respect to the performance as reported by the PerfumeJS tool of mobile web applications from the point view of software developers. The goal is also shown in Table 2.

Table 2: Goal Definition

| | |
|---|---|
| **Analyze** | Selected web applications |
| **For the purpose of** | Evaluation |
| **With respect to their** | Performance as reported by the PerfumeJS tool |
| **From the point of view of** | Software Developers |
| **In the context of** | Mobile web applications |

## 3.2 Question

From the goal of the experiment, we construct one research question shown below:

**Research Question:**

*What is the performance of web applications generated according to the PerfumeJS tool?*

For the research question, we will collect five metrics generated by the PerfumeJS plugin extension. The output will allow us to understand several performance data measured by the PerfumeJS tool.

## 3.3 Metrics (Performance)

For the performance metric we will be tracking all the five metrics collected by the PerfumeJS profiler, all five metrics will help answer the research question of the experiment.

### 3.3.1 Navigation Timing

Navigation Timing is tracked by collecting seven data measurements including totalTime, fetchTime, timeToFirstByte, dnsLookupTime, downloadTime, workerTime and headerSize. However, some of these data are not useful regarding the performance of mobile web applications. Therefore, we will only focus on data such as totalTime and downloadTime. However, all seven data are still collected throughout the experiment.

### 3.3.2 Network Information

Network Information is tracked by collecting four data measurements including round-trip-time(rtt), effectiveType, downlink and saveData. Although this metric is collected throughout the experiment, it does not provide useful data regarding the performance of mobile web applications. This metric is useful to improve information regarding the network. Therefore, we will not focus on this metric in our experiment, although it is still collected by the plugin as mentioned earlier.

### 3.3.3 Storage Estimation

Storage Estimation is tracked by collecting five data measurements including usage, indexedDB, quota, serviceWorker and caches. Similarly to the network information metric, this metric is collected throughout the experiment, however does not provide useful data regarding the performance of mobile web applications. Therefore, we will not focus on this metric in our experiment, although it is still collected by the plugin as mentioned earlier.

### 3.3.4 First Paint (FP)

First Paint is the time it takes for the browser to render any visual aspect different from what was on screen before the navigation to the web page. This metric is tracked by collecting this render time only.

### 3.3.5 First Contentful Paint (FCP)

First Contentful Paint is the time it takes for the browser to render the first bit of content from the DOM. Similarly to FP, this metric is tracked by collecting this render time only.

## 3.4 GQM-Tree

Since we defined the goal, question and metrics, we use them to construct a GQM-Tree shown in Figure 1.

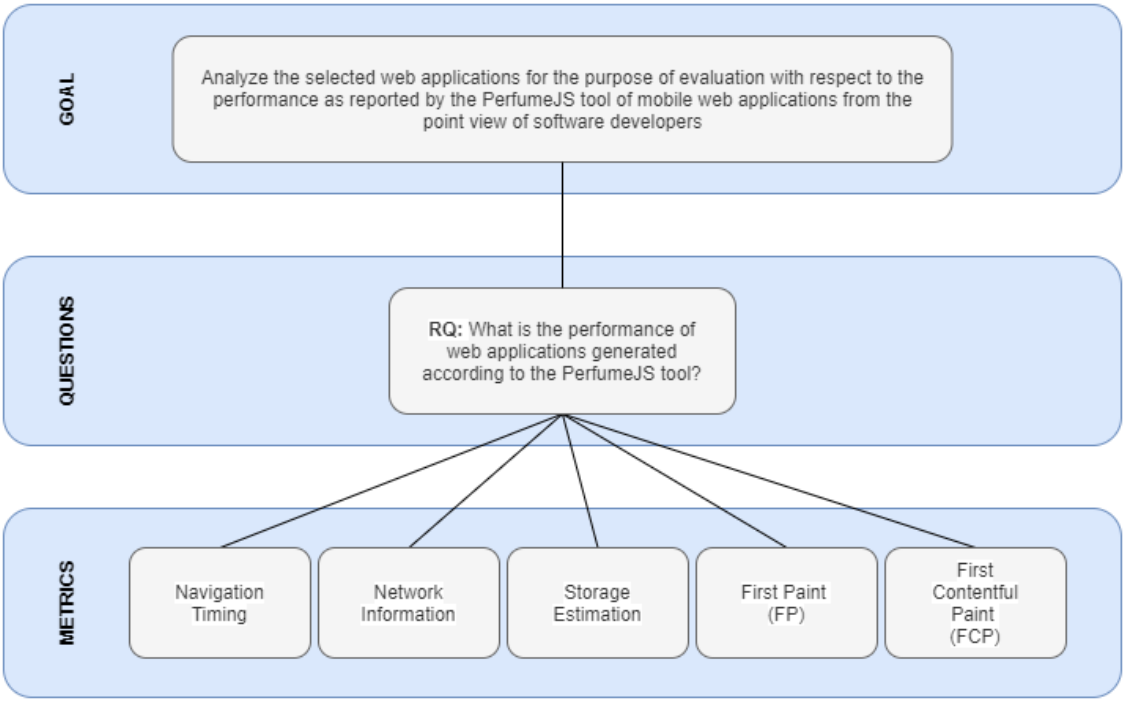


Figure 1: GQM-Tree

# 4 Experiment Planning

## 4.1 Subjects Selection

The subjects for the experiment are selected from Tranco list [4]. From the list, we randomly select 100 web applications to use in our experiment shown in Table 3. Furthermore, the selection includes two exclusion criteria which are web applications that redirects to a different domain and web applications that contain pop-ups that deny the content of the application to be fully loaded. The first exclusion criteria is included because if the web application redirects to another domain whenever the "index.html" file is loaded, then this denies the PerfumeJS library to measure the performance metrics. The second exclusion criterion is included because if the pop-up denies the web application content to be fully loaded, then the PerfumeJS library will not be called to measure the metrics.

Table 3: Selected mobile web applications

| ID | URL |
|---|---|
| 1 | residentialproductsonline.com |
| 2 | halorescue.org |
| 3 | sigmasport.com |
| 4 | chinakongzi.com |
| 5 | imgbin.com |
| 6 | atri-online.org |
| 7 | lasvegashowto.com |
| 8 | uny.edu.ve |
| 9 | educaweb.com |
| 10 | donttellmamanyc.com |
| 11 | geniol.com.br |
| 12 | ffidrac.com |
| 13 | wisebrain.org |
| 14 | musiciansfriend.com |
| 15 | newsquest.co.uk |
| 16 | motiv-art.com |
| 17 | startlijsten.nl |
| 18 | pdf4me.com |

[4] https://tranco-list.eu/

Table 3: Selected mobile web applications

| ID | URL |
|---|---|
| 19 | atamalek.ir |
| 20 | hossintropia.com |
| 21 | createhealthylife.ru |
| 22 | hereisthecity.com |
| 23 | c7discount.com |
| 24 | foxroach.com |
| 25 | theduramater.com |
| 26 | merchprint.ru |
| 27 | reachtheworldonfacebook.com |
| 28 | bonsaijs.org |
| 29 | drweb.de |
| 30 | lmaga.jp |
| 31 | amazingfilehosting.com |
| 32 | cinematheque.ch |
| 33 | lovinsoap.com |
| 34 | associateshomecare.com |
| 35 | goldenfront.ru |
| 36 | peek.link |
| 37 | mastgeneralstore.com |
| 38 | tilesdirect.net |
| 39 | brasilwagen.com.br |
| 40 | audiencex.com |
| 41 | 919lab.com |
| 42 | drpepper.com |
| 43 | 1hoursloansbaltimore.com |
| 44 | callbells.com.ua |
| 45 | irocks.com |
| 46 | pte.pl |
| 47 | morningread.com |
| 48 | feralcat.com |
| 49 | hamradiolicenseexam.com |

Table 3: Selected mobile web applications

| ID | URL |
|---|---|
| 50 | haikangduoli.com |
| 51 | fnyyb.com |
| 52 | nwmissouri.edu |
| 53 | ausflowers.com.au |
| 54 | yunanfengji.net |
| 55 | internetcalls.com |
| 56 | zbyszl.com |
| 57 | spinservers.com |
| 58 | siicex-caaarem.org.mx |
| 59 | lassho.edu.vn |
| 60 | fathersraisingsons.com |
| 61 | 1249455.live |
| 62 | infidigit.com |
| 63 | ssylka-na-gidru.com |
| 64 | spaceaduanas.com |
| 65 | chihiro.jp |
| 66 | name95.cn |
| 67 | manuceau.net |
| 68 | janm.org |
| 69 | addictivedesertdesigns.com |
| 70 | nabzgroup.com |
| 71 | dunkirkmovie.com |
| 72 | herballegacy.com |
| 73 | spitfog.ro |
| 74 | ironmountain.co.uk |
| 75 | wallpapereez.com |
| 76 | healthychildren.org |
| 77 | 518db.com |
| 78 | greenlining.org |
| 79 | ojo.com |
| 80 | digivate.com |

Table 3: Selected mobile web applications

| ID | URL |
|---|---|
| 81 | beammachine.net |
| 82 | folhasantista.com.br |
| 83 | pioneersacademy.com |
| 84 | dienlanhgiakhang.net |
| 85 | shamra.net |
| 86 | zonalandeducation.com |
| 87 | serviceexperts.com |
| 88 | bpynu.com |
| 89 | jquerybyexample.net |
| 90 | 18000.com.ua |
| 91 | mintmuseum.org |
| 92 | bodytrading.fr |
| 93 | luxcosmeticsdv.ru |
| 94 | indiaties.in |
| 95 | nass.co.uk |
| 96 | fabrykaform.pl |
| 97 | mercycollege.edu |
| 98 | caremountmedical.com |
| 99 | jsonwebtoken.io |
| 100 | penskeusedtrucks.com |

## 4.2 Experiment Variables

As discussed in Section 3.3, we focus on some of the data measurements from the five metrics. Table 4 shows a summary of the variables selected for our experiment.

Table 4: Experiment Variables

| Variable Name | Value |
|---|---|
| totalTime | milliseconds *(ms)* |
| downloadTime | milliseconds *(ms)* |
| FP | milliseconds *(ms)* |
| FCP | milliseconds *(ms)* |

- *totalTime*: is request plus response time.
- *downloadTime*: is response time to a request.
- *FP*: is the render time of any visual aspect different to what was on screen before navigation to the web page.
- *FCP*: is the render time of the first bit of content from the DOM.

### 4.3 Experiment Design

The design of our experiment encompasses one set of subjects containing 100 web applications randomly selected. The 100 web applications are loaded to observe the performance metrics according to the PerfumeJS library. To ensure the collected metrics from PerfumeJS are consistent, we execute each web application for 10 runs. Throughout the experiment runs, all the variables elected in Section 4.2 are collected.

## 5 Experiment Execution

To begin the experiment we will need to setup the devices and tools used throughout the experiment. For the devices, we use one Android mobile phone and one laptop machine to execute all runs of the experiment to have a consistent results and avoid fluctuation in the results due to hardware/device changes. The specification of the android mobile phone and the laptop machine are as follows:

**Samsung Galaxy J7 Duo**

- Android version 8.0.0 Oreo
- 64 bit Octa Core Processor
- 139.5 mm (5.5") HD sAMOLED display
- 3 GB RAM
- 32 GB Memory

**HP Laptop**

- Ubuntu 19.10
- 64-bit OS type
- Intel Core i7-7500U CPU @ 2.70GHz x 4
- Intel HD Graphics 620 (Kaby Lake GT2) / AMD Radeon
- 1 TB Memory

For the tools, we only need to setup the PerfumeJS library for which the setup and execution steps are explained in Section 2. Additionally, the execution of the Android Runner is explained in the framework's repository[5]. Finally, we setup the configuration of the experiment so each subject run takes 30 seconds to execute and load the web pages. As mentioned earlier, we execute each subject for 10 runs and we have a total of 100 web applications. Additionally, the Android Runner frameworks includes a cool down time between each run which takes 2 minutes, Therefore, the total time to execute the entire experiment is 2,500 minutes.

# 6 Results

The output of the experiment can be found in "outputs" directory of the repository used to execute the experiment[6].

## 6.1 Summary Statistics

The summary statistics which includes minimum, maximum, mean, median, standard deviation and coefficient of variation for all runs of all web applications is shown in Table 5.

Table 5: PerfumeJS metrics summary statistic including all web applications (milliseconds)

| **Metric** | **Min** | **Max** | **Mean** | **Median** | **Standard Deviation** | **Coefficient of Variation** |
|---|---|---|---|---|---|---|
| **Total time** | 4.7 | 1052.9 | 62.6452 | 33.2 | 100.1408 | 1.59854 |
| **Download time** | 1.3 | 1028 | 31.3084 | 12.2 | 81.61759 | 2.606891 |
| **FP** | 582.1 | 4427.9 | 1865.36 | 1859.5 | 469.8954 | 0.251906 |
| **FCP** | 665.5 | 4711.2 | 1886.31 | 1873 | 480.6902 | 0.2548309 |

[5]https://github.com/S2-group/android-runner
[6]https://github.com/KhalilMay/android_runner_perfumejs

## 6.2 Data Representation

Figure 2 shows a violin plot representing the distribution of all the data for the totalTime metric. The plot shows that the upper quartile is at 117.35 milliseconds(*ms*) and the maximum data point is at 1052.9 *ms*, however many data points are dispersed higher than 117.35 *ms* towards the maximum data point. A possible reason for outlier data points to be present is due to both the android device and laptop machine being connected to a network which might cause delays in some cases.

Figure 3 shows a violin plot representing the distribution of all the data for the downloadTime metric. The plot shows the upper quartile is at 51.625 *ms* and a maximum data point is at 1028 *ms*, however similarly to the totalTime metric many data points are dispersed higher than the upper quartile towards the maximum data point. The reason for these data spikes is the same as the totalTime metric, where the connection to the network could cause a possible delay.

Figure 4 shows a violin plot representing the distribution of all the data for the FP metric. The plot shows the upepr quartile is at 2989.788 *ms* and the maximum data point is at 4427.9 *ms*. Unlike the first two metrics, there are few data points between the upper quartile data point and the maximum data point. Furthermore, the plot shows that the data in this metric is better distributed when looking at the dispersion of the data compared to the distribution of the data in the totalTime and downloadTime metrics, this is also visible when looking at the coefficient of variation (CV) in the summary statistics.

Figure 5 shows a violin plot representing the distribution of all the data for the FCP metric. The plot shows the upper quartile is at 3008.225 *ms* and the maximum data point is at 4711.2 *ms*. Similarly, to the FP metric there are few data points between the upper quartile data point and the maximum data point. Additionally, the plot also shows that the data is better distributed when looking at the dispersion of the data compared to the distribution of the data in the totalTime and downloadTime metrics, this is again also visible by looking at the CV in the summary statistics.

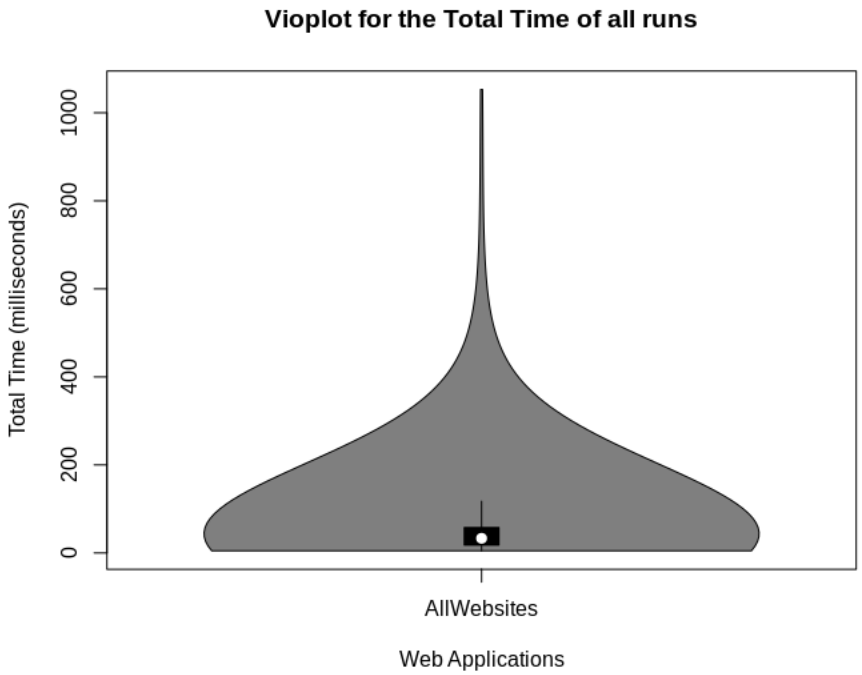


Figure 2: Violin Plot of totalTime for all web applications

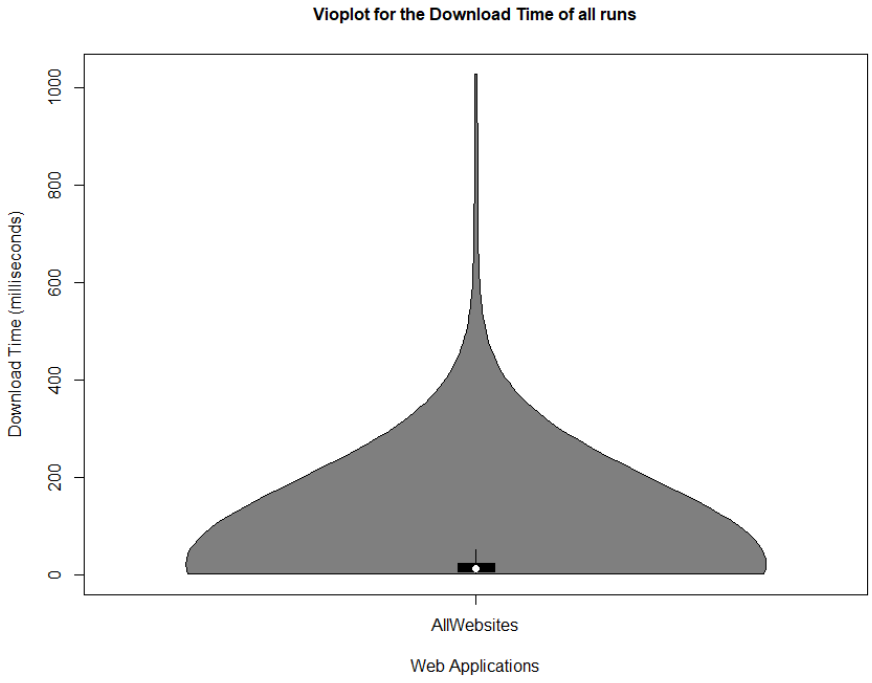


Figure 3: Violin Plot of downloadTime for all web applications

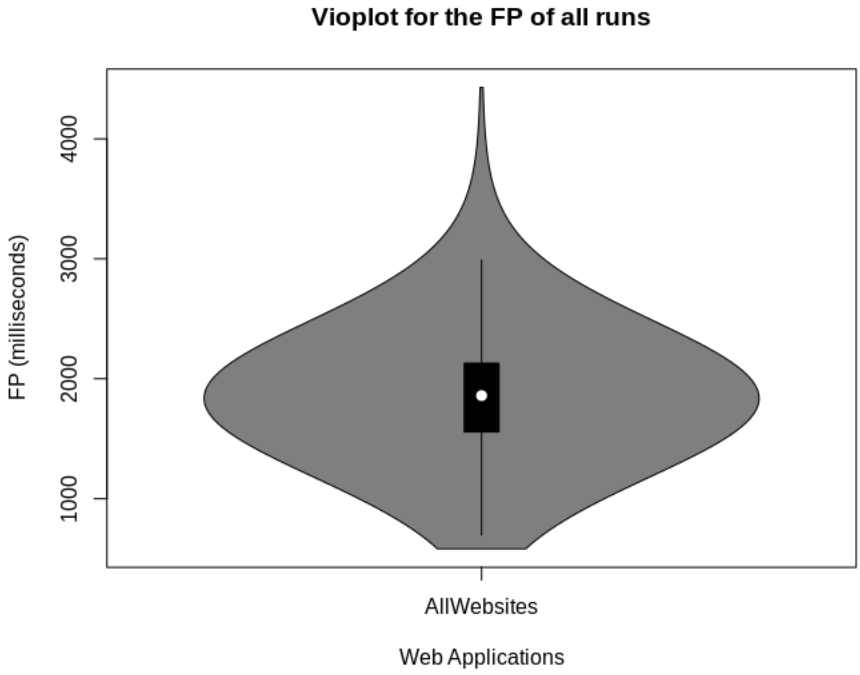


Figure 4: Violin Plot of FP for all web applications

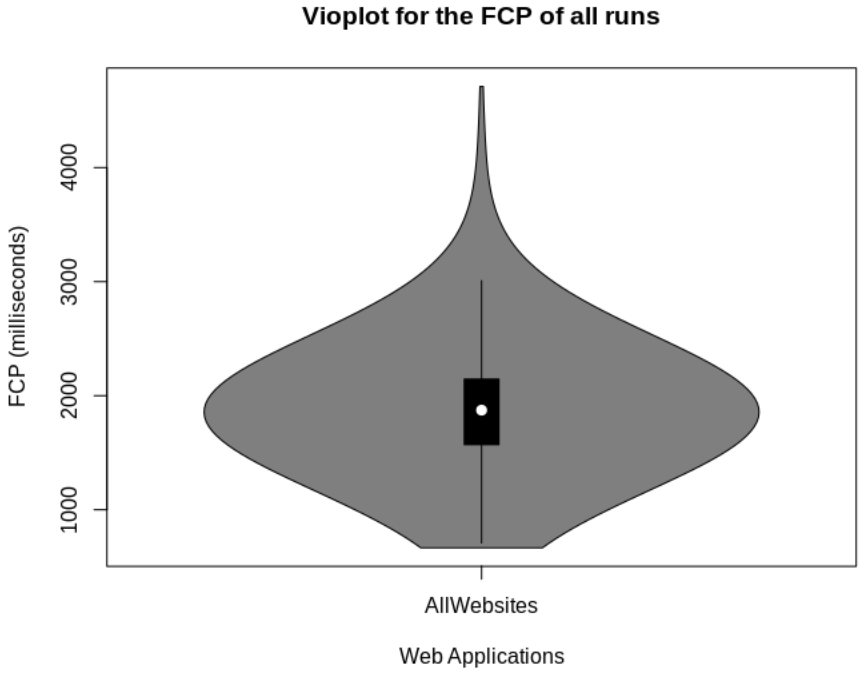


Figure 5: Violin Plot of FCP for all web applications

# 7 Discussion

The outcome of the experiment display the potential outcome of the using PerfumeJS. To further understand the results of the tool, we need to answer the **Research Question (RQ)** of the paper.

The paper contains one *RQ* stating *"What is the performance of web applications generated according to the PerfumeJS tool?"* which the result of the experiment will answer. The result of the experiment shows that only the data of four variables (totalTime, downloadTime, FP and FCP) are useful to understand the performance of web applications. First, the totalTime performance metric is observed to be with a mean of 62.6452 *ms* for the web applications. Secondly, the download time performance metric is observed to be with a mean of 31.3084 *ms* for the web applications. Thirdly, the FP performance metric is observed to be with a mean of 1865.36 *ms* for the web applications. Finally, the FCP performance metric is observed to be with a mean of 1886.31 *ms* for the web applications.

The result of the experiment will allow developers understand some performance metrics of web applications. Furthermore, developers are also able to use the new PerfumeJS plugin added with the Android Runner framework to help them better understand and optimize web applications under the development stage.

# 8 Threats To Validity

## 8.1 External Validity

This validity addresses the generalisability of the experiment result, which means that the results should be valid for the complete population targets the web applications are selected from. The subjects consisting of 100 web applications are all selected from the Tranco List. However, the two exclusion criteria implemented with the selection drops a portion of web applications from the population target, thus making it difficult to generalise the results of the experiment.

Furthermore, the experiment is performed using android device and laptop machine with good specifications(e.g. Samsung Galaxy J7 Duo and HP "1TB Memory" Laptop). Running the experiment under the same hardware devices, we can be certain that the experiment setting will reflect similarly to a real world scenario. Nonetheless, performing the experiment under newer devices may generate different results of the metrics collected. Therefore, future replication of the experiment will be beneficial to validate the result of this experiment and reduce the threat to this validity.

## 8.2 Internal Validity

The maturation threat in this validity addresses the subjects potentially reacting different between trial over a period of time. This threat is mitigated by the two minutes cool down between runs implemented in the Android Runner framework. Furthermore, this cool down allows buffers used by the operating system of the mobile device/machine to timeout. Additionally, the option to clear the cache of the browser available with the Android Runner framework is used to mitigate its effect on the measurements of the metrics.

Furthermore, the reliability of the results can be impacted by several factors such as services running in the background and consuming assets of the device/machine. These factors are mitigated as much as possible by manually closing the services before starting the experiment. However, a network connection is needed for both devices in order for this experiment to work. Therefore, this connection might cause a slight bias for some of the measurements collected which will increase the threat for this validity.

## 8.3 Construct Validity

The construct validity addresses the threat of possibly having the constructs incorrectly defined before translating them into measures. To mitigate this threat, we used the GQM approach to define the constructs and measures as mentioned in Section 3.

Furthermore, mono-operation and mono-method biases are two threats to construct validity stating the possibility of a bias in a single operation or a single method. We mitigate the mono-operation bias by performing 10 runs for each of the web applications selected. However, the experiment suffers from mono-methed bias as our results are obtained from one method which is the PerfumeJS library.

## 8.4 Conclusion Validity

The conclusion validity addresses the statistical correctness and the relationship between the treatment and the outcome of the paper. Furthermore, all threats for this validity are aimed to address the statistical analysis of the experiment result. However, the aim our experiment was only to show the performance metrics according to the PerfumeJS library. Therefore, we did not apply any statistical tests in our paper.

# 9 Conclusion

This paper implemented a new plugin aimed to measure performance metrics of web applications using PerfumeJS library. The library is injected in the "index.html" file of the application along with a custom code used to allow the plugin to collect the metrics measured by the perfumejs library. Furthermore, an experiment is conducted to understand the performance of mobile web applications according to the perfumejs library. The result shows that totalTime, downloadTime, FP and FCP of web applications have an average of 62.6452, 31.3084, 1865.36 and 1886.31 *ms* respectively, therefore allowing us to answer the research question of the paper. To conclude, developers can use the plugin and the result of the experiment to better understand and optimize the performance of web applications.

## 9.1 Future Work and Limitations

This paper provides a PerfumeJS extension plugin to measure performance metrics of web applications according to the PerfumeJS library. Furthermore, the paper provides an experiment performed to show the results of the variables generated using the plugin. However, the plugin still has some limitation that can be improved, the plugin has a limitation which prevents the plugin to measure and collect the metrics if the android device and machine are not connected to a network.

Therefore, there are possible future work that can improve and optimize the plugin and verify the validity of the experiment. The following are potential future work:

- Make changes to the plugin so it does not need to send the metrics collected through a server.
- The current plugin needs to be tested under different android device to validate the results of the experiment
- An experiment could be done to collect additional metrics that are collected after the web application is loaded (e.g. fid, lcp, cls and tbt)

# References


[1] Ericsson. "Ericsson Mobility Report". In: (June, 2020). URL: https://www.ericsson.com/en/mobility-report/reports/june-2020.

[2] J. Clement. "Share of global mobile website traffic 2015-2020t". In: (July 21, 2020). URL: https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/.

[3] Claes Wohlin et al. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[4] Victor R Basili. *Software modeling and measurement: the Goal/Question/Metric paradigm*. Tech. rep. 1992.