**Improve Web Performance**

## Tools Required for Performance Optimization

### How Browsers Load

Browser goes through multiple behind the scenes operations, like downloading the necessary resources, transpiling said resources, and etc., to display the content on the screen. These operations consist of download, parsing, style, layout, paint, and composite, and are collectively referred to as the loading phase

#### Parsing

First, when a page is loaded on the browser, the browser downloads the HTML files. Parsing is a stage where the browser converts the downloaded HTML in order to construct the **DOM Tree**. If the browser discovers <script/>, <link/>, or <div/> elements during the parsing stage, it requests the appropriate resources and downloads them as well. If the CSS is included in either the HTML or the resources, it constructs the **CSSOM Tree** at the same time. The constructions of DOM Tree and CSSOM Tree are explained in greater detail below.

**DOM Tree Construction**

To better explain the DOM Tree construction, let’s refer to the following HTML file. When parsing happens, the browser analyzes the HTML file to create the DOM, and connects each DOM object in a tree data structure to establish parent-child relationships. Tags like <body>, <p>, and <div> are created as nodes in the DOM tree and reference child nodes.

**CSSOM Tree Construction**

As in the example above, if the file includes an external stylesheet file or an internal stylesheet like the style.css, the browser translates the CSS and constructs the CSS Object Model (CSSOM) Tree. Selectors like body, p, and span are used as nodes and each node references different styles.

#### Style Calculation

During the style calculation stage, the browser builds another tree by using the DOM and CSSOM Trees constructed during the parsing stage, to match each element to the corresponding style. The image below depicts how the render tree is crafted by combining the two trees built during the parsing stage.

#### 3. Layout

In the layout stage, the browser mathematically calculates the exact position and size of the node. In order to do so, the browser starts at the root and calculates each node by iterating through the entire tree, and applies the results of the layout at the exact location with the exact number of pixels in the render tree.

#### 4. Paint

In the previous layout stage, each node in the render tree was converted into actual pixels by using the computed properties. Now, the browser applies the CSS properties that are not related to the positioning of the element like color and transparency. Then, the pixelated results are crafted into layers like that of the Photoshop, and are managed in layers. However, not all elements are transformed into layers. The [transform](https://developer.mozilla.org/en-US/docs/Web/CSS/transform) property build them into layers, and this process is called painting.

#### 5. Compositing and Rendering

The browser updates the screen with a composite of layers created in the paint stage. The page can finally be viewed on the screen after the compositing and rendering process are finished.

### Layout and Repaint

The browser loading stages after the style calculation (style calculation -> layout -> paint -> composite) are collectively referred to as rendering. This rendering process can happen multiple times depending on the situation. The render tree, constructed during the style calculation stage, must be reconstructed every time the DOM Tree or the CSSOM Tree change because of JavaScript. If, for example, DOM has been inserted/removed or if a CSS element that has a geometric effect on the design (height, width, position) is changed, the render tree must be reconstructed. Therefore, the process must be repeated from the layout stage, and this process is called a **layout**. (It is also referred to as **reflow** in Firefox.)

layout occurs when a CSS element that affects the geometry changes, but contrarily, if a change has been made that does not affect the geometry of the structure, the browser engine skips the layout stage. The process continues from the paint stage, and this process is called **repaint**.

### Performance Improvement Index

In order to improve the loading process, to find what could be improved, **on which basis** the loading speed is slow must be made clear, the index used to measure performance is introduced. The performance index’s measurement standards can mainly be categorized into two: **the browser perspective** and the **user perspective**.

#### Performance Measurement from the Browser’s Perspective

The traditional method of measuring performance was to use the events taking place in the browsers as milestones. When the page loads, DOMContentLoaded and load events occur, and the performance was measured based on when each event occurred

**DOMContentLoaded Event**

* When the HTML and CSS parsing have finished
* When the engine is ready to construct the render tree (DOM and CSSOM have been constructed)

**load Event**

* When all resources on the HTML have been loaded

The points of occurrence for both of these events can be monitored by using the Chrome DevTools or by using the navigation timing API.

However, as the development paradigm continues to change, it became difficult to judge the application’s performance solely on the points of occurrence of DOMContentLoaded and load. Recently popular Single Page Applications (SPA) have very early point of occurrence for DOMContentLoaded and load events, but still have slow loading due to other scripts that continue to run. Therefore, the developer community had to come up with a method to measure performance from the user’s perspective.

#### **Performance Measurement from the User’s Perspective**

Measuring performance from the user’s perspective is based on multiple timestamps of the content being displayed to the user. The faster the meaningful content is displayed to the user, the better the performance is, and developers should attempt to optimize the code to make it happen faster.

Google has defined multiple moments that the user feels that the page has loaded quickly or not, and such moments as indices to measure performance. Such moments are described as follows.

**First Paint (FC)**

The moment something appears on the blank page

**First Contentful Paint (FCP)**

The moment a text or an image begins to be printed

**First Meaningful Paint (FMP)**

The moment the content that could be useful to the user is printed. CSS and JavaScript needed to display the content has started to load and the style has been applied so that the content is apprehensible.

The **FMP** is the most important among them, and it is critical to display useful information to the users before the loading finishes in order to leave an impression that the application is fast. From the user’s perspective, good performance means fast FMP, and such can be achieved by optimizing the aforementioned critical rendering path.

**Time to Interactive (TTI)**

The moment JavaScript has finished executing so that the user can perform actions

### Performance Measuring Tools

**Lighthouse Chrome Extension Tool**

### Audit Options

Google Lighthouse allows you to run Audits for Mobile as well as Desktop version of your application – that’s your first level of customization, ‘Device’. Next, we choose which aspects are to be audited. You can choose from the following:

1. Performance
2. Progressive Web Apps
3. Best Practises
4. Accessibility
5. SEO. Yes, you can get a score for the SEO of your page here. The evaluated parameters include features such as
   * Document having a meta description,
   * Presence(or absence) of a title element,
   * Use of legible font sizes,
   * Links not having descriptions
   * Presence of unsuccessful HTTP status codes like 400, 404, 500 etc.

The other option that Google Lighthouse provides you with, is Throttling. Basically, this is a chance for you to simulate the performance of the page/ application on a slower internet connection or CPU power of the host machine. Here, you can choose from:

1. Simulated Fast 3G, 4x CPU Slowdown
2. Applied Fast 3G, 4x CPU Slowdown
3. No throttling

‘Clear Storage‘, which you can check so that the browser’s storage, distributed across the Local Storage, Indexed DB, etc. are cleared ahead of the Audit. This is particularly helpful when you want as accurate a score as possible for the Performance and PWA metrics.

## Webpage Loading Optimization

#### CSS Optimization

#### JavaScript Optimization

### Minimizing the Number of Resource Requests

#### Image Sprite

#### Bundling CSS and JavaScript

#### Using Internal Stylesheets

#### Substitute Small Images with HTML and CSS

#### Eliminating Redundant Codes

#### HTML Markup Optimization

## Webpage Rendering Optimization

### Layout Optimization

#### JavaScript Optimization

#### Animation Optimization

##### Using CSS Animation