A Framework for Smart CSS

T G Ashwin Kumar
Department of Computer Science and Engineering
PSG College of Technology
Coimbatore, India
tgashwinkumar@gmail.com

***Abstract*—Cascading Style sheet (CSS) has been an integral part of all the websites that we visit in our day-to-day life. Thanks to that, we have reached a state of developing masterly crafted websites which are more or less similar to the graphics seen in films. It brings to a point of creating the most optimal and developer-friendly CSS framework. This paper aims to describe the features and discuss the working of such a framework. There are many popular and established libraries that are used by the current frontend developers and this paper compares those libraries towards a better framework.**

***Index Terms*—Postprocessors, Preprocessors, HTML, Minification**

I. Introduction

For front-end developers, CSS is the first tool they'll learn. As HTML creates the structures and semantics, CSS lays out the style and design of the website. Basic attributes like color, alignment, fonts, and background and complex concepts like flexbox and grid systems are native to CSS and there have been multiple libraries to support these concepts. It allows one to adapt the presentation to different types of devices, such as large screens, small screens, or printers. CSS is independent of HTML and can be used with any XML-based markup language.

II. Literature Survey

There have been multiple innovations in the field of Cascading Style Sheets. All these innovations do not aim only at web optimization, most of the time, for the developer's comfort and ease of access. For a beginner, CSS will always be a tricky concept in one's learning appetite, so the developer starts to search for libraries and frameworks that are popular in the market and thus feeds on their learning curve. The literature survey provided here will focus both on the optimization features and as well as, the other face of the coin - developer comfort.

A. Working of CSS

Firstly, the browsers parse through the HTML page provided and then pass the parsing to the asset files such as Javascript files (JSX files), Image files (PNG, JPG, GIF, etc.), and Stylesheets (CSS, SCSS, etc.). Browser constructs the Document Object Model (DOM) from the HTML file and stylesheets being the next priority element, gets downloaded and loaded to the browser as CSSOM.

And so the other asset files are loaded in the order. The reason for this is that styles define the look of the page and rendering the page first without them would be a waste of processing powers and a bad user experience.

Only when the browser has both DOM and CSSOM available it can create a render tree by combining them and start rendering the screen. In short, no CSS was downloaded, and no page was rendered.

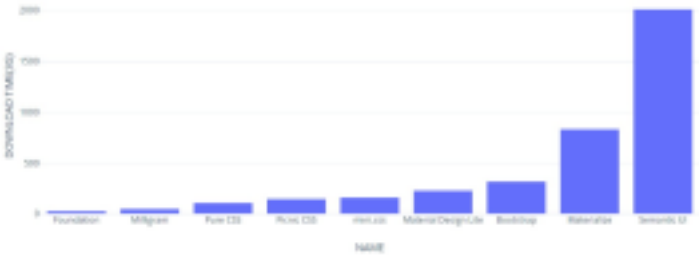

Fig. 1. CSS Framework vs Download Time in Slow 2G/3G Network (in ms)

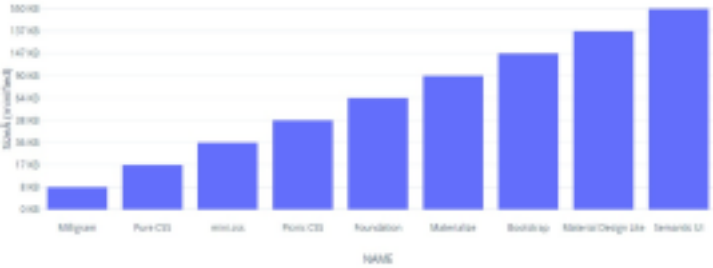

Fig. 2. CSS Framework vs Size of the bundle (minified)

B. Minified CSS

CSS Minimization has been one of the to-go methods to optimize the tons of stylesheets to hard-to-read formats, only to make it easy for the machines. Mostly the CSS files, that are used in the application and also user defined, consists of too many new line characters, spacing, indentations and comments. Hence a processor to check for these attributes can be developed for the given CSS input file and a minified CSS output file can be generated. In Fig. 3 a code example is randomly chosen and tried for minimized. Even though the reduction is in terms of mere bytes, the percentage reduction is considerable.

```
.ui-helper-hidden {
display: none;
}
.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;
}
```

```
.ui-helper-hidden{display:none}.u
i-helper-hidden-accessible{border:
0;clip:rect(0 0 0 0);height:1px;ma
rgin:-1px;overflow:hidden;padding:
0;position:absolute;width:1px}
```

Fig. 3. CSS file before minimization vs after minimization

C. Lazy loading by grouped CSS

This a device-centered solution for reducing the bulk size. Media queries is a technique in CSS for categorizing the CSS to a specific device, mostly with respective to its viewport width. Lazy loading refers to the technique of loading an asset to the renderer of the browser only when its required. Thus if the CSS classes are grouped according to the media queries and separated to different files and loaded, it can reduce the bundle size.

TABLE I
COMPARISONS BETWEEN FAMOUS CSS LIBRARIES IN SIZE BEFORE MINIMIZATION AND AFTER MINIMIZATION

Library	Size before binimization	Size after minimization
Bootstrap	242.25 KB	190.36 KB
Bulma	248.31 KB	202.46 KB
Foundation	166.47 KB	131.66 KB
Milligram	10.62 KB	9.014 KB
Half Moon	494.7 KB	364.64 KB

For CSS, which varies lot for different pages, for example a page in the application can be full of tables and another page consists only of form components. For such type of templative nature of application, the CSS files can be split accordingly. But in a CSS framework point of view, such type of occurences and splitting becomes rare, but the idea holds good.

D. Nomenclature for predictable styling

For an application, CSS can be globally operating and any changes to existing style codes can lead to consequences of an undetermined style. This leaky nature of CSS makes the developers dare not touch the existing CSS of a particular application. Almost all problems with CSS at scale boil down to confidence (or lack thereof): People don't know what things do anymore. People dare not make changes because they don't know how far-reaching the effects will be.

With a predictable nomenclature, people will know what changes to implement in the stylesheets, and hence the scare of CSS will reduce. Another advantage of nomenclature is the ease of learning it provides. Bootstrap is one of the libraries that has reached a household status in terms of learning and compatibility, compared to others CSS frameworks. There's also a systematic nomenclature with all these CSS frameworks that help the user to spend less time on documentation and API of these libraries and hence pace up the development process of the application/module.

E. CSS Preprocessors

A CSS preprocessor is a program that lets you generate CSS from the preprocessor's own unique syntax. There are many CSS preprocessors to choose from, however, most CSS preprocessors will add some features that don't exist in pure CSS, such as mixin, nesting selector, inheritance selector, and so on. These features make the CSS structure more readable and easier to maintain.

There are various CSS Preprocessors such as LESS, SASS, SCSS, and Stylus. With the provision of various programming paradigms such as loops and variables, provides extra features to the styling.

III. DISCUSSION

The intent for the discussion will be to create a smart CSS framework/library where a developer is availed of all the listed features as briefed above and also to make it much easier and optimized for usage. This framework will include a post-processor architecture with the class names and identifiers following a particular nomenclature. This post processor will scan for the class names and then provide the necessary CSS definitions to the attached style sheet.

A. Just in Time mode

For a CSS framework like Bootstrap which has 1,00,000 lines to 2,00,000 lines of minified CSS code, not all the class names are used. There can be a preprocessor to filter out the CSS identifiers used in the application and discard all the other classes.

TABLE II
COMPARISONS BETWEEN VARIOUS TEST INSTANCES IN REDUCTION OF BUNDLE SIZES WITH RESPECT TO JUST IN TIME MODE

Test Instances	Time taken for CSS Input Loading	Size of CSS Output CSS Output	Time taken for CSS Output loading
For a static loaded webpage	4622 ms	21.3 KB	198 ms
For a dynamic loaded webpage including user defined form components and color based theme templates	4622 ms	21.3 KB	198 ms
Portfolio site which is pages driven and has mutliple templates with hindered color palette	4622 ms	21.3 KB	198 ms

B. CSS Postprocessors

Like CSS Preprocessors, which provide multiple features for the vanilla CSS, there is also postprocessors, which on the given stylesheets, create alterations that make it more efficient and usable. PostCSS allows you to define custom CSS-like syntax that could be understandable and transformed by plugins. That being said PostCSS is not strictly about CSS spec but about syntax definition manner of CSS. In such a way you can define custom syntax constructs like at-rule, which could be very helpful for tools built around PostCSS. PostCSS plays the role of a framework for building outstanding tools for CSS manipulations.

C. Purging

It needs an architecture where it scans for the HTML/XML/JSX files the user creates for their applications, scans for the class names that match the nomenclature suggested by the library and created respective CSS definitions for it. Hence it requires a system of libraries that scan for existing CSS files, and other asset files to CSS and append to the existing CSS file. The process where it goes through each and every template file in the project is called Purging. There are some notable libraries that provide the purge functionality and provide the style identifiers in a usable format such as JSON.

IV. METHODOLOGY

From the provided ideas, it is evident that more than one idea can be incorporated and a CSS framework can be created. The ideas were split according to their compatibility with each other and two prototypes were created. These prototypes help us achieve the desired result and also experiment with the performance and provide a solution that is considered to be "smart".

A. Prototype I

This prototype is inclined to make a CSS framework with classes split according to the media queries and also themes used. It also provides a color palette highly inspired by the Google Flutter Color template. It works with the help of the SASS loader. It extends the prevailing CSS to more syntax, boasts more features, and helps us build a procedure-oriented structure for the generation of system-defined CSS class names. Later with the help of PostCSS loaders, it was minified and autoprefixed to a minimal CSS library which will be later uploaded to the cloud and used by the target developer via a CDN. It is to be noted that, here PostCSS is used an external helper to provide the desired output CSS file at the end.

B. Prototype II

Here the idea of Just In time was emphasized. So it needs an architecture where it scans for the HTML/XML/JSX files the user creates for their applications, scans for the class names that match the nomenclature suggested by the library, and created respective CSS definitions for it. Hence it requires a system of libraries that scan for existing CSS files, and other asset files to CSS and append to the existing CSS file.

Here comes PostCSS to the frame. PostCSS allows you to define custom CSS-like syntax that could be understandable and transformed by plugins. That being said PostCSS is not strictly about CSS spec but about syntax definition manner of CSS. In such a way you can define custom syntax constructs like at-rule, which could be very helpful for tools built around PostCSS. PostCSS plays the role of a framework for building outstanding tools for CSS manipulations.

We have built the whole prototype as a PostCSS plugin following the instructions and API provided by the PostCSS documentation pages. This idea was deeply inspired by the CSS Framework TailwindCSS. The whole prototype was required if such an architecture was possible and it made any desired changes to the performance, bundle size, and download.

It reduced the downloading to 0, as it was converted to an NPM/Yarn library and was adapted to be even used offline.

C. Testing

Assertion testing was created to check whether the library provided the desired output with all the plugins provided. Here the plugins are provided in the main CSS file and the output is gathered in output CSS file . The performance is calculated by Javascript Frontend Libraries such as PostCSS.process and native JS's "it" function. There were tests to check for the conversion of the format of color codes. This enables the developer to provide the colors either by RGBA values or hex codes.

V. CONCLUSION

CSS is one of the fundamental concepts in web development and from the discussion above, it is only evident that it has grown toward developer comfort and optimized programming.

Thus, it leads to better websites in the future. Libraries like Bootstrap and TailwindCSS are well-established in the style sheet market and the reason can easily be inferred due to their features similar to the smart CSS framework.

REFERENCES

[1] D. Mazinanian and N. Tsantalis, "An Empirical Study on the Use of CSS Preprocessors," 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Osaka, Japan, 2016, pp. 168-178

[2] Coyier, Chris. "Popularity of CSS Preprocessors."

[3] Jane O'Donnell. 2019. SASS (syntactically awesome style sheets). J. Comput. Sci. Coll. 34, 4 (April 2019), 101–102

[4] "Overview of PostCSS Architecture" (https://postcss.org/docs/postcss-architecture)

[5] Lie, H. W., Bos, B. (2005). Cascading style sheets: Designing for the web. Addison-Wesley Professional.