**CSS Layers for CSS Resets**

I was always one of those people that preferred the aggressive CSS Reset method, which deletes most of the default styles of the browser. For example, changing the default heading style of <h1> to a <h6> elements that have a bigger font-size and font-weight.

On the other end, I liked the treatment that Normalize CSS does to shadow DOM elements, which doesn’t exist in CSS Reset.

Because of this, I tried to find methods to mix “CSS Reset” and “Normalize CSS”. But even then, I had some issues with CSS specificity that I needed a workaround for.

Fast forwarding to today, all browsers now support CSS Layers. Therefore, while working on Pink Design, the open source design system of Appwrite, we started to rethink how we could handle this better.



Before we start, let’s talk a bit about CSS Reset methods.

CSS Reset Methods

For many years, there was a “fight” over which method of resetting CSS is better.

**The two methods:**

**1. Normalize CSS** — a gentle approach to fix differences between browsers while keeping the native styles of HTML elements like heading elements of <h1>, <h2>, and so on.

As mentioned before, Normalize CSS also takes care of shadow DOM elements that sometimes look different in different browsers.

**Demo of treatment of shadow DOM elements in Normalize CSS:**

/\*\*

\* 1. Correct the inability to style clickable types in iOS and Safari.

\* 2. Change font properties to `inherit` in Safari.

\*/

::-webkit-file-upload-button {

-webkit-appearance: button; /\* 1 \*/

font: inherit; /\* 2 \*/

}

**2. CSS Reset** — CSS Reset is a more aggressive method that often ignores the default style inherited from the browser.

**Demo of CSS reset**

This code will remove the special font-size, font-weight and margin of the <h1> to a <h6> elements:

h1, h2, h3, h4, h5, h6 {

margin: 0;

font-size: inherit;

font-weight: inherit;

}

Combining Methods

By combining both, Normalize CSS and CSS Reset, you can profit from both approaches.

This ensures that both inner shadow DOM elements are taken care of, and unhelpful styles inherited from the “user-agent stylesheet” are ignored.

The easiest way to accomplish this is by loading both. Demo of how to implement in Sass preprocessor:

/\* CSS Resets \*/

@use 'normalize';

@use 'reset';

You would think that if we first load Normalize CSS and then CSS Reset, this would make our CSS Reset have a stronger Specificity, right? Not precisely; let’s talk about some issues with this.

Issues with Combining Methods

In Appwrite Pink, we use Normalize CSS while combining it with ‘The New CSS Reset’. ‘The New CSS Reset’ is a new method of resetting CSS that uses the new native CSS reset features.

For both projects of “Normalize CSS” & “The New CSS Reset”, we are taking them as is, without any changes (from [NPM](https://npmjs.com/)), even the unnecessary parts from Normalize CSS, like fixing the different style of <h1> element that will need to be removed with the CSS Reset.

**Header <h1> style in “Normalize CSS”:**

/\*\*

\* Correct the font size and margin on `h1` elements within `section` and

\* `article` contexts in Chrome, Firefox, and Safari.

\*/

h1 {

font-size: 2em;

margin: 0.67em 0;

}

**General remove style in “The New CSS Reset” (including <h1> element):**

/\*

Remove all the styles of the "User-Agent-Stylesheet",

except for the 'display' property.

- The "symbol \*" part is to solve Firefox SVG sprite bug

\*/

\*:where(:not(html, iframe, canvas, img, svg, video, audio):not(svg \*, symbol \*)) {

all: unset;

display: revert;

}

But here, starting our problems. And now, let's talk about basic CSS stuff which defines our styles:

Order Does Matter

The order of the CSS selectors is important. This is because, usually what is coming last is stronger then what is come first. In our case, the order of our CSS Resets files is correct.

First we want to load the “Normalize CSS” which will normalize our differences between different browsers, and then we want to remove what we don’t need with a CSS reset, in our case “The New CSS Reset”.

**Example of imports in Scss:**

/\* CSS Resets files order \*/

@use 'normalize'; /\* 1 \*/

@use 'reset'; /\* 2 \*/

/\* In general, last code is stronger in CSS \*/

CSS Specificity

We have the strength of our selectors defined according to the CSS selector strength (elements, class names, and ID names). From weakest to strongest selectors, we have the element, class, and ID selectors.

**Example:**

In this example, the selector of the ID will win the “CSS Specificity War”, because the ID selector is stronger than the selector of a class name or an element.

This means that the color of the <h1> element, in this case, will be pink.

<h1 class="title" id="mainTitle">some content</h1>

#mainTitle { color:pink; } /\* 1 (ID), 0 (Classes), 0(element) \*/

.title { color:yellow; } /\* 0 (ID), 1 (Classes), 0(element) \*/

h1 { color:red; } /\* 0 (ID), 0 (Classes), 1(element) \*/

Our CSS Specificity Conflict

If we take a look at Normalize CSS selector of <h1> elements, it has the power of one element:

/\* 0 (ID), 0 (Classes), 1 (element) \*/

h1 {...}

This is a relatively low-strength selector.

Let’s have a look at the main CSS Reset selector from “The New CSS Reset” method:

/\* 0 (ID), 0 (Classes), 0 (element) \*/

\*:where(:not(html, iframe, canvas, img, svg, video, audio):not(svg \*, symbol \*)) {...}

To make it with the lowest specificity possible, the :where() pseudo selector is used. One of the main ideas of the :where() pseudo selector is to remove any CSS specificity created by the selector.

But here comes the conflict, the style of the <h1> is stronger than the style of the CSS Reset, and this is an issue for us because we want the CSS Reset to overcome the Normalize CSS.

One way to solve this conflict is to remove the unnecessary parts of Normalize CSS, which means removing all the un shadow DOM parts style. But this can be an issue if we load the project automatically from an NPM package, which is complex to maintain.

CSS Layers to the Rescue

CSS layers were invented to solve cases like this, where we want to tell the browsers that a specific layer is more important than the other and ignore the CSS specificity of another layer.

To do so, we have the @layer rule that defines a layer. This will wrap part of the styles, define the parts of the layer, and act out the CSS specificity only inside the layer itself.

@layer normalize {

/\* CSS Normalize Here \*/

}

@layer the-new-css-reset {

/\* CSS Reset here \*/

}

This in itself will solve our issue.

To be more precise in defining the order of the layer, we can add the “layer statement” that will determine the order you want the code to appear.

**Example:**

/\* layer statment - define the order,

even if the order of the code will not be in the same way \*/

@layer normalize, the-new-css-reset;

@layer normalize {

/\* CSS Normalize Here \*/

}

@layer the-new-css-reset {

/\* CSS Reset here \*/

}

**CodePen Demo:**

Sass Preprocessor Support

To keep the separation of files while using Sass, we will need to add some adjustments to our Sass code.

@use 'sass:meta';

@layer normalize, the-new-css-reset;

@layer normalize {

@include meta.load-css('normalize');

}

@layer the-new-css-reset {

@include meta.load-css('the-new-css-reset');

}

This way, we can keep our CSS layers separate in separate files and ensure that the last layers will win the “cascade war” while keeping our code well organized.

Browser Support

CSS Layers have already been implemented in all evergreen browsers for quite a while. In iOS, from version 15.4, which at the time of writing this post, we are currently on version 16.2.

To Summarize

This post covered how we can solve CSS specificity issues, specifically on CSS Reset layers.

This method of using CSS Layers to solve CSS conflicts will be more commonly used as we approach the end of 2023 when users upgrade their browsers.

While building Pink Design, we had the luxury of building a product for developers, a type of user that updates their browsers regularly. Therefore, we chose to move forward with this new method using CSS Layers.

Final Words

[Pink Design](https://pink.appwrite.io/) is finally out today. [Pink Design](https://pink.appwrite.io/) was built with new inventive CSS methods, and prioritizes accessibility and developer experience. As a company developing an open-source product, this is a big step for Appwrite’s community of contributors. As always, we will keep improving [Pink Design](https://pink.appwrite.io/), allowing it to grow alongside [Appwrite](https://appwrite.io/).

Thank you for being a part of our journey! If you haven’t already, browse the 💻 [Pink Design GitHub Repository](https://github.com/appwrite/pink) or check out our 🚀 [Getting Started Guide](https://pink.appwrite.io/getting-started) to use Pink Design in your projects or when contributing to [Appwrite](https://appwrite.io/).

Until next time!

**Further reading/watching on CSS Layers:**

* [A Complete Guide to CSS Cascade Layers](https://css-tricks.com/css-cascade-layers/) — by *Miriam Suzanne*
* [Getting Started With CSS Cascade Layers](https://www.smashingmagazine.com/2022/01/introduction-css-cascade-layers/) — by *Stephanie Eckles*
* [The CSS Cascade, a deep dive](https://www.youtube.com/watch?v=zEPXyqj7pEA) (video) — by *Bramus Van Damme*

**Further reading on CSS Resets:**

* [How Does CSS Work?](https://elad.medium.com/how-does-css-work-92fe7116916d) — by *Elad Shechter*
* [The New CSS Reset](https://elad.medium.com/the-new-css-reset-53f41f13282e) — by *Elad Shechter*