# Remove unused JavaScript

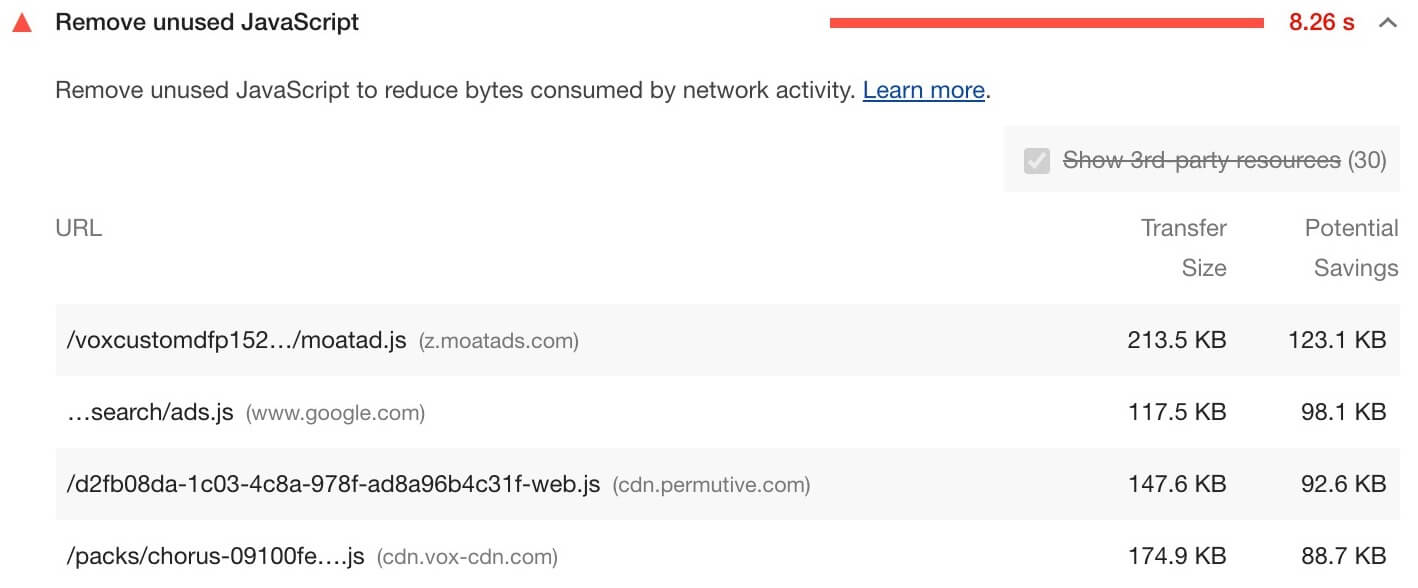
Jul 7, 2020

Unused JavaScript can slow down your page load speed:

* If the JavaScript is [render-blocking](https://developers.google.com/web/fundamentals/performance/critical-rendering-path/adding-interactivity-with-javascript), the browser must download, parse, compile, and evaluate the script before it can proceed with all of the other work that's needed for rendering the page.
* Even if the JavaScript is asynchronous (i.e. not render-blocking), the code competes for bandwidth with other resources while it's downloading, which has significant performance implications. Sending unused code over the network is also wasteful for mobile users who don't have unlimited data plans.

## How the unused JavaScript audit fails [#](https://web.dev/unused-javascript/?utm_source=lighthouse&utm_medium=devtools#how-the-unused-javascript-audit-fails)

[Lighthouse](https://developers.google.com/web/tools/lighthouse/) flags every JavaScript file with more than 20 kibibytes of unused code:

Click a value in the **URL** column to open the script's source code in a new tab.

See the [Lighthouse performance scoring](https://web.dev/performance-scoring) post to learn how your page's overall performance score is calculated.

## How to remove unused JavaScript [#](https://web.dev/unused-javascript/?utm_source=lighthouse&utm_medium=devtools#how-to-remove-unused-javascript)

### Detect unused JavaScript [#](https://web.dev/unused-javascript/?utm_source=lighthouse&utm_medium=devtools#detect-unused-javascript)

The [Coverage tab](https://developers.google.com/web/tools/chrome-devtools/coverage) in Chrome DevTools can give you a line-by-line breakdown of unused code.

The [Coverage](https://pptr.dev/#?product=Puppeteer&version=v4.0.0&show=api-class-coverage) class in Puppeteer can help you automate the process of detecting unused code and extracting used code.

### Build tool for support for removing unused code [#](https://web.dev/unused-javascript/?utm_source=lighthouse&utm_medium=devtools#build-tool-for-support-for-removing-unused-code)

Check out the following [Tooling.Report](https://tooling.report) tests to find out if your bundler supports features that make it easier to avoid or remove unused code:

* [Code Splitting](https://bundlers.tooling.report/code-splitting/)
* [Unused Code Elimination](https://bundlers.tooling.report/transformations/dead-code/)
* [Unused Imported Code](https://bundlers.tooling.report/transformations/dead-code-dynamic/)

## Stack-specific guidance [#](https://web.dev/unused-javascript/?utm_source=lighthouse&utm_medium=devtools#stack-specific-guidance)

### Angular [#](https://web.dev/unused-javascript/?utm_source=lighthouse&utm_medium=devtools#angular)

If you are using Angular CLI, include source maps in your production build to [inspect your bundles](https://angular.io/guide/deployment#inspect-the-bundles).

### Drupal [#](https://web.dev/unused-javascript/?utm_source=lighthouse&utm_medium=devtools#drupal)

Consider removing unused JavaScript assets and only attach the needed Drupal libraries to the relevant page or component in a page. See the [Defining a library](https://www.drupal.org/docs/8/creating-custom-modules/adding-stylesheets-css-and-javascript-js-to-a-drupal-8-module#library) for details.

### Joomla [#](https://web.dev/unused-javascript/?utm_source=lighthouse&utm_medium=devtools#joomla)

Consider reducing, or switching, the number of [Joomla extensions](https://extensions.joomla.org/) loading unused JavaScript in your page.

### Magento [#](https://web.dev/unused-javascript/?utm_source=lighthouse&utm_medium=devtools#magento)

Disable Magento's built-in [JavaScript bundling](https://devdocs.magento.com/guides/v2.3/frontend-dev-guide/themes/js-bundling.html).

### React [#](https://web.dev/unused-javascript/?utm_source=lighthouse&utm_medium=devtools#react)

If you are not server-side rendering, [split your JavaScript bundles](https://web.dev/code-splitting-suspense/) with React.lazy(). Otherwise, code-split using a third-party library such as [loadable-components](https://www.smooth-code.com/open-source/loadable-components/docs/getting-started/).

### Vue [#](https://web.dev/unused-javascript/?utm_source=lighthouse&utm_medium=devtools#vue)

If you are not server-side rendering and using the [Vue router](https://next.router.vuejs.org), split the bundles by [lazy loading routes](https://next.router.vuejs.org/guide/advanced/lazy-loading.html).

### WordPress [#](https://web.dev/unused-javascript/?utm_source=lighthouse&utm_medium=devtools#wordpress)

Consider reducing, or switching, the number of [WordPress plugins](https://wordpress.org/plugins/) loading unused JavaScript in your page.

**How to Detect Unused CSS or JavaScript in Your Code**

[](https://akashkriplani.medium.com/?source=post_page-----8d200ef07e50-----------------------------------)

[Akash Kriplani](https://akashkriplani.medium.com/?source=post_page-----8d200ef07e50-----------------------------------)

[Jun 20](https://medium.com/geekculture/detect-unused-css-or-javascript-in-your-code-8d200ef07e50?source=post_page-----8d200ef07e50-----------------------------------) · 4 min read



Photo by [Christopher Robin Ebbinghaus](https://unsplash.com/@cebbbinghaus?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com?utm_source=medium&utm_medium=referral)

How often have we come across situations where your project has a lot of code that may have been used at an earlier stage to build a [MVP](https://en.wikipedia.org/wiki/Minimum_viable_product) and then years later it is just lying there because someone did not do a proper job of refactoring it?

‘Often’ would be the obvious answer.

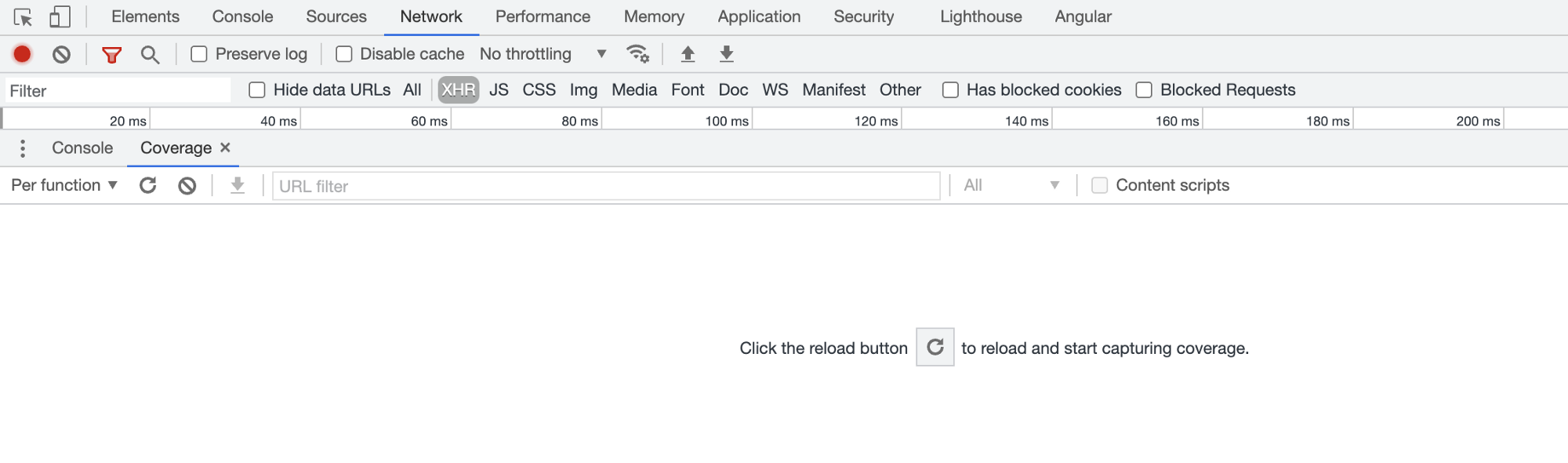
Since the real world applications have various developers working across multiple locations, the task can be equally tedious for a developer who has been working for quite some time on the same codebase or someone who is new to the team. Sounds like an overwhelming job, isn’t it, especially if it is a very large code base with several modules in it.

Here’s where you can breathe easy by using a very powerful tool provided by Google — [Chrome DevTools](https://developer.chrome.com/docs/devtools/).

**Open the DevTools on your Chrome tab** by right clicking and inspecting using ‘Inspect’ or pressing F12 from your keyboard.

Press Ctrl + Shift + P on Windows *(or Command + Shift + P for Mac users)*

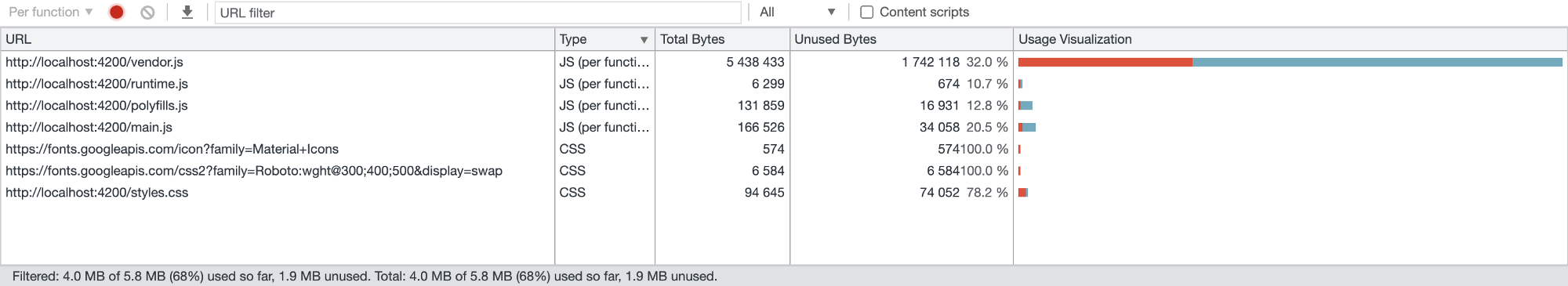
**Type in ‘Show Coverage’** and the new Coverage tab will open as shown in the screen below:



Coverage tab of Chrome DevTools

Click on the reload icon shown in the above text.

The coverage of the code (JavaScript and CSS) will be calculated. Attached below is the sample screenshot of an Angular app that I have been working on.



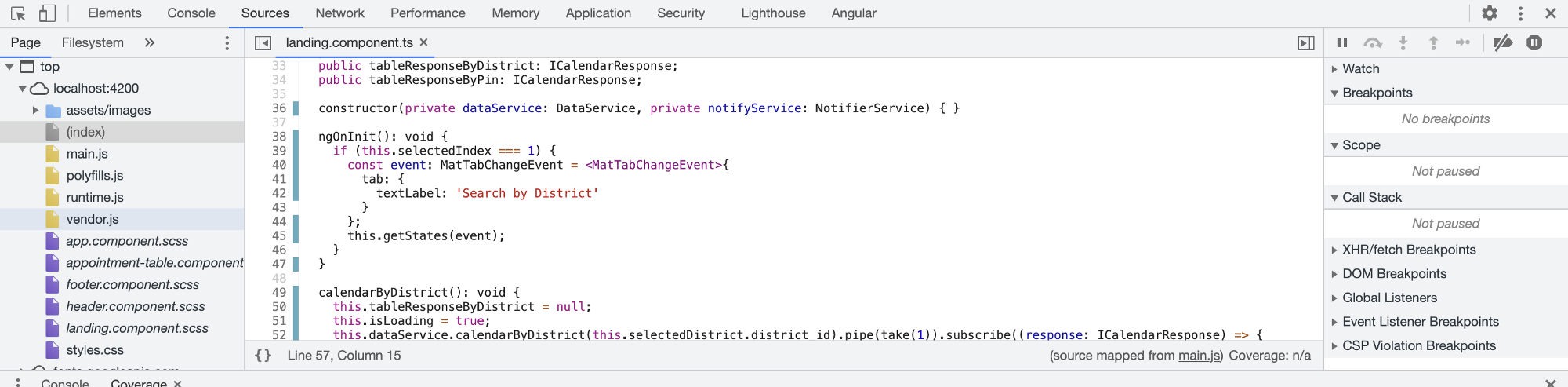
Code coverage analysis in the coverage tab

The coverage tab will give you the **Total Bytes**, the **Unused Bytes** and the **Usage Visualization** categorized by the file type.

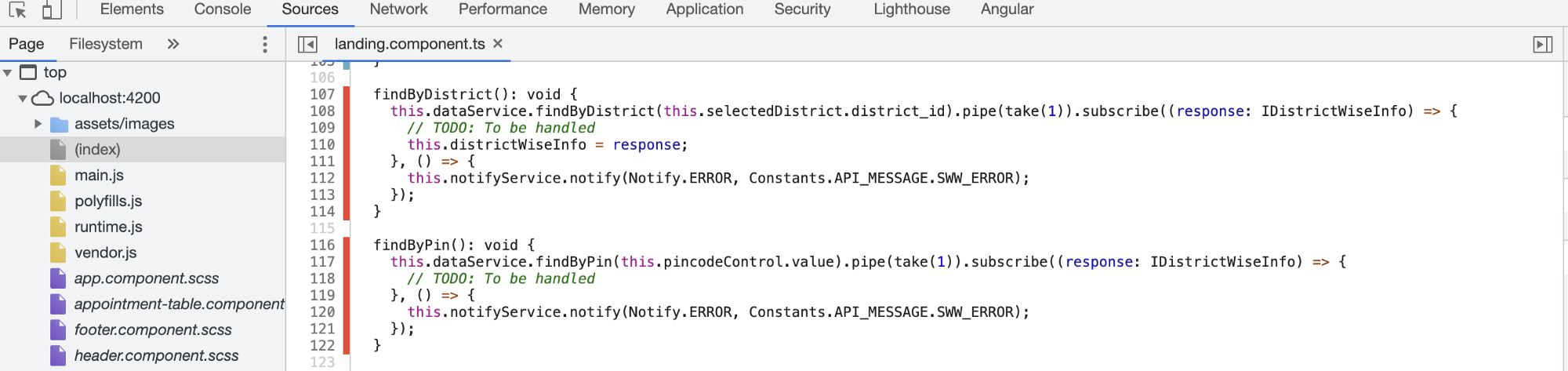
The progress bar highlighted in red in the usage visualization progress bar is the unused code. 32% belong to the functions that have not yet executed (or may never execute if that is an unused code in your app) in the vendor.js file.

The progress bar highlighted in the bluish green color around 68% has been executed already. If you look at the image at the bottom left, it shows you exactly that.

Since, in an Angular app, the TS files get bundled and then splits into JS files, I will try and show you the current TS file and the unused (dead) code it contains. Open the current file, using the ‘Sources’ tab. In my case, it is a **landing.component.ts** file:



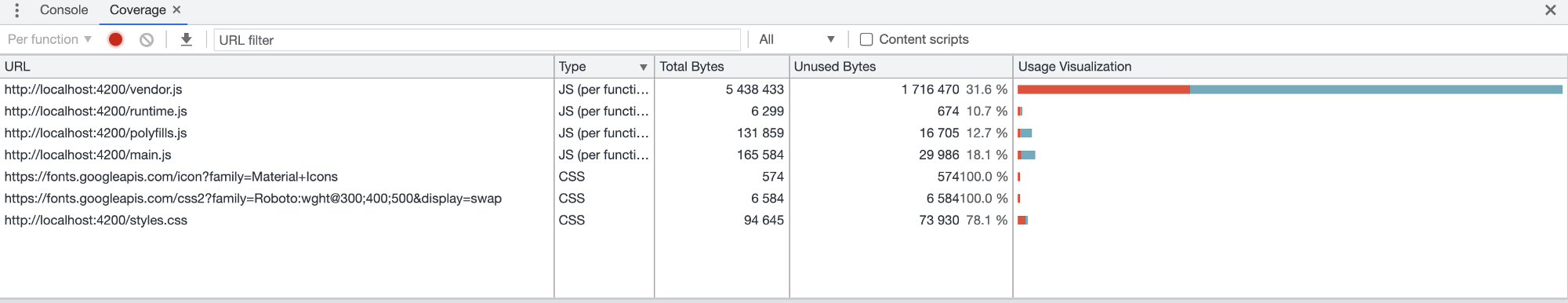
Executed code highlighted in bluish green



Unexecuted code highlighted in red

The first screenshot shows the lines executed already in bluish green background. The second one shows the lines that may have not been executed still (or may not execute ever which can correspond to a dead code in your app).

In my case, these methods are extra and I do not require them and hence I will be removing them. The coverage of unused code of the **vendor.js** file decreases to 31.6%. So, the idea behind it is to reduce to the count of **Unused Bytes** so that your code bundle is optimized and contains the minimal code that is needed. The same can be done for all the CSS files that you have in your project.



Improved code coverage in the usage utilisation

The overwhelming job that you were assigned seems now can be resolved quite easily, right?

**Conclusion**

And there we have it. I hope you have found this article useful. Thank you for reading. Please feel free to provide your comments, suggestions, and any other feedback that may be helpful.