SEO PowerSuite

THE ULTIMATE GUIDE

TO IMPROVING OPTIMIZATION SCORE

No-nonsense developer tricks to grow rankings

Once Google announced that they were going to roll out Mobile Speed Update, their approach to measuring page speed changed as well. This, in particular, has been reflected in Google PageSpeed Insights tool. Now, when you paste a URL into the tool, it scores according to two categories: Page Speed and Optimization.

The data for Page Speed Score is taken from the Chrome User Experience (CrUX) database of real users' interactions with webpages. It means that the numbers depend not only on the actual speed of your site but also on the users' connection speeds and their devices (which can be slow and outdated). This shift from lab data to field data makes it impossible to get Page Speed metrics through local tests and influence them directly.

However, as the SEO PowerSuite mobile page speed experiment has shown, there is a strong correlation between a mobile site's position in search results and its average Optimization Score. This is great news since it is totally possible to influence this metric by tweaking and improving your site's technical factors.

Advanced checklist for improving Optimization Score

First of all, plug in your URL into the PageSpeed Insights tool to test the performance of your pages. Google will provide you a list of optimization suggestions. Below you'll find these suggestions explained, expanded and flavored with practical tips.

Contents

1. Avoid landing page redirects.

Recommendation Trigger:

There is more than one redirect from a particular URL to the final landing page.

Why it matters:

Redirects slow down page rendering, thus negatively affecting desktop and mobile experience. Rendering is delayed due to each redirect adding a single HTTP request-response roundtrip (best-case scenario) or numerous additional roundtrips (worst-case scenario) to perform the DNS lookup, TCP handshake, and TLS negotiation.

How to avoid redirects?

1) Create a responsive site.

Google recommends creating a responsive site without unnecessary redirects in order to deliver great multi-device experience.

2) Choose a suitable redirect type.

It is recommended to avoid the use of redirects altogether unless it is absolutely necessary. If you still have to, then choose the redirect type, which is best for your needs:

- ***301 or 302 redirect:*** Permanent redirects (301) are best when there's a need to delete old content and redirect to new content, or when there's no alternate page to redirect users to. Temporary redirects (302) are best for short-term changes (like limited-time promotions) or for redirecting users to device-specific URLs. Whatever the choice of the redirect type is, there will be no loss of link juice.
- ***JavaScript or HTTP redirect:*** Google supports both redirect types. The main difference between them is that HTTP redirects are the cause for some latency on the server side, while JavaScript redirects slow down the client side (as they need to the need to first download the page, then parse and execute the JavaScript before triggering the redirect).

2. Enable compression.

Recommendation Trigger:

Compressible resources are served without compression.

Why it matters:

Smaller content size reduces the time spent to download the resource, reduces data usage for the client, and improves pages' rendering time.

How to avoid redirects?

1) Remove unnecessary resources prior to compression.

While it is super important to compress resources, first of all, it is recommended to prioritize the removal of unnecessary data. The best optimized resource is the resource not sent. Before the actual compression make sure to review your site resources to get rid of needless data.

2) Gzip all compressible resources.

Google recommends to gzip all the compressible content. It's possible to find sample configuration files for most popular servers with detailed comments for any configuration flags or settings through HTML5 Boilerplate project.

3) Consider alternative encoding techniques.

There are some alternatives to gzip encoding if you'd rather not use a gzip tool for some particular reason. For example, Brotli is a lossless compression algorithm, supported by all modern browsers, which has a superior compression ratio, comparable to gzip.

However, Brotli compresses very slowly and decompresses fast, so you should pre-compress static assets with Brotli+Gzip at the highest level, and compress dynamic HTML with Brotli at levels 1–4.

4) Use different compression techniques for different resources.

While compression can be applied to HTML code and digital assets required by a page, web fonts, images, CSS, etc. require different compression techniques to achieve faster page rendering. For example, in case you use HTTP/2, then HPACK compression for the HTTP response headers will eliminate redundant header fields.

3. Improve server response time.

Recommendation Trigger:

Server response time is above 200 ms.

Why it matters:

Faster response time becomes a necessity in the light of the fact that 53% of mobile users will leave a page that does not load in less than 3 seconds.

How to avoid redirects?

There is an array of factors that may slow down server response time, like slow application logic, slow database queries, slow routing, slow frameworks, slow libraries, CPU/memory starvation, etc. You have to consider all of those to improve the server response.

1) Figure out what slows down your site by inspecting the existing performance data.

Use a tool like WebPage Test, Pingdom, GTmetrix, or Chrome Dev Tools to collect performance data and pinpoint what slows down your content delivery.

Bear in mind that even if your response time is indicated as under 200 ms, the user on an older device or with a slower connection may experience much higher response time. In order to improve such user's experience, you have to aim for a *First Meaningful Paint* < 1s, a *SpeedIndex value* < 1.25s, *Time to Interactive* <5s and <2s for repeat visits.

2) Configure your server in a user-oriented way.

- ***Use HTTP/2*** (bear in mind that your CDNs also support HTTP/2) for a performance boost.
- ***Enable OCSP stapling*** on your server to speed up TLS handshakes.
- ***Support both IPv6 and IPv4.*** IPv6's neighbor discovery (NDP) and route optimization can make websites 10–15% faster.
- ***Add resource hints*** to warm up the connection and speed up delivery with faster dns-lookup, preconnect, prefetch, and preload.

4. Leverage browser caching.

Recommendation Trigger:

The response from the server does not include caching headers or the resources are specified to be cached for only a short time.

Why it matters:

The absence of the caching policy on a site triggers a great number of roundtrips needed between the client and server during the resources fetching process. As a result, it leads to delays, blocking of page rendering, and higher data costs for visitors. When the caching policy is implemented, it helps the client to figure out if and when it can reuse responses it has fetched in the past.

How to avoid redirects?

1) Make sure that each resource specifies an explicit caching policy that answers:

- whether a resource can be cached;
- who can cache it;
- how long the caching will take;
- how it can be efficiently revalidated when the caching policy expires (if applicable).

Per Google recommendation, the minimum cache time should be one week and up to one year for static assets.

2) Use Cache-Control to cut down data charges and eliminate network latency.

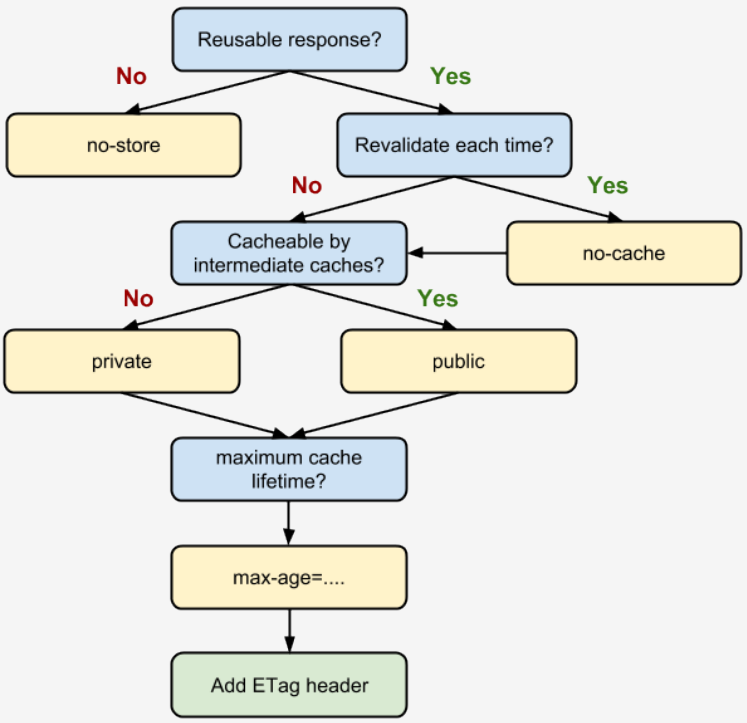
Cache-Control directives allow you to automatically control under which conditions (e.g., "*no-cache*" and "*no-store*") and for how long (e.g., "*max-age*", "*max stale*", and "*mini-fresh*") the browser can cache a response without a need to communicate with the server.

3) Use ETags to ensure efficient revalidation.

The server uses ETag HTTP headers to communicate a validation token to ensure that no data is transferred if a resource has not changed since it was requested last time. It makes the process of the resource update checks much more efficient.

4) Check Google's recommendations for optimal Cache-Control policy.

Google has compiled a checklist and a decision tree to work out the most appropriate caching policy aimed at caching as many responses as possible for the longest possible period and to provide validation tokens for each response for more efficient revalidation:

Checklist	Flowchart
✔ Use consistent URLs ✔ Ensure that the server provides validation tokens (ETags) ✔ Identify which resources can be cached by intermediaries ✔ Determine the optimal cache lifetime for each resource ✔ Determine the best cache hierarchy for your site ✔ Minimize churn	

A general rule to apply: resources that are likely to change should be cached for a short time, while static resources should be cached for an indefinite period to avoid revalidation.

5. Minify resources (HTML, CSS, and JavaScript).

Recommendation Trigger:

The size of one of the resources could be reduced through minification.

Why it matters:

Minification helps to get rid of redundant data from the resources delivered to visitors without affecting how a resource is processed by the browser. It can have a tangible impact on your loading speed and site performance.

How to avoid redirects?

1) Eliminate all the redundant data from your web assets.

Use shorter variable and function names, remove code comments or space symbols in HTML, repeated styles in CSS, unnecessary image metadata, etc.

2) Use minification together with compression.

Minification may sound like compression, but in reality, it is much more granular. Compression algorithms are good for reducing the overall size of a page, however, most of them do not remove the unused code from (/* ... */), HTML (<!-- ... -->), and JavaScript (// ...) comments, collapse the CSS rules, or perform dozens of other content-specific optimization activities.

3) Minify other resources as well.

Apart from minifying just text-based assets like HTML, CSS, and JavaScript, it is also possible to minify images, video, and other types of content depending on your needs. For example, there are some forms of metadata in images that you may want to keep when publishing them on a photo-sharing site, but all the other similar data can be useless, so do not think twice to remove it.

4) Automate minification.

There is a number of tools that can ease the whole process by minifying thousands of different resources on a site. For example, PageSpeed Module does this automatically, and it can be integrated with Apache or Nginx web servers. Plus, it is possible to use third-party tools for these needs, like HTMLMinifier (for HTML), CSSNano or CSSO (for CSS), and UglifyJS (for JavaScript).

6. Optimize images.

Recommendation Trigger:

The images on the page can be optimized by reducing their size without significantly impacting their visual quality.

Why it matters:

Images answer for an average of 60% of a page size, and heavy images can significantly slow down site's rendering. Optimization of images helps to reduce the overall file size without impacting visual quality, which means less competition for the client's bandwidth and faster downloading and rendering of the content.

How to avoid redirects?

1) Serve responsive images.

Images need to be responsive to adapt to different viewport sizes and usage scenarios to load quickly and simultaneously, and look great. To achieve this, use relative sizes for images, use the picture element when there's a need to specify different images depending on the device characteristics, and use srcset and the x descriptor in the img element to inform browsers when to use specific images.

2) Find optimal settings for image optimization.

As images usually amount to the heaviest part of a page, they happen to be the hardest to optimize for in terms of efficient performance. The reason: there is no definite answer for how to best compress an image. There are many techniques that can help to reduce an image, however, the optimal settings require thorough consideration of format capabilities, content of encoded data, quality, pixel dimensions, and more.

Go through the following checklist with common optimization techniques that may help you to work out your custom image optimization plan:

Image Optimization Checklist:

- ✔ Eliminate unnecessary image resources
- ✔ Leverage CSS3 to replace images
- ✔ Use web fonts instead of encoding text in an image
- ✔ Use vector formats where possible
- ✔ Minify *and* compress SVG assets to reduce their size
- ✔ Pick the best raster formats (start by selecting the right universal format: GIF, PNG or JPEG, but

also consider adding WebP and JPEG XR assets for modern clients)

- ✔ Experiment with optimal quality settings
- ✔ Resize on the server & serve images scaled to their display size
- ✔ Remove metadata
- ✔ Enhance img tags with a srcset parameter for high DPI devices
- ✔ Use the picture element to specify different images depending on device characteristics, like device size, device resolution, orientation, and more
- ✔ Use image spriting techniques carefully (with HTTP/2, it may be best to load individual images)
- ✔ Consider lazy loading for non-critical images
- ✔ Cache your image assets
- ✔ Automate your image optimization process

For more recommendations and tips, consult Google's guide to image optimization.

7. Optimize CSS delivery.

Recommendation Trigger:

A page includes render blocking external stylesheets, which delay the time to first render.

Why it matters:

A page needs to process CSS prior to its rendering. When CSS is stuffed with render-blocking external stylesheets, the process often requires multiple roundtrips, which delays the time to first render.

How to avoid redirects?

1) Inline small external CSS resources directly into HTML.

Inserting small CSS directly into an HTML document allows the browser to proceed with rendering the page. Here is an example of inlining a small CSS file.

2) Avoid inlining large CSS files.

While inserting small CSS speeds up the rendering time, inlining large CSS files will increase the size of the above-the-fold HTML code, which, in its turn, slows the rendering time down. Bear in mind that "large" and "small" measures are relative here: define the optimal measure according to your business needs.

3) Avoid inlining CSS attributes.

Inlining CSS attributes on HTML elements (e.g., <p style=...>) often leads to unnecessary code duplication, which is blocked by default with Content Security Policy.

8. Prioritize visible content.

Recommendation Trigger:

Additional network roundtrips are required to render the above-the-fold content of the page.

Why it matters:

If your above-the-fold content exceeds the initial congestion window – as a rule, 14.6kB compressed – it will require multiple roundtrips between the server and user's browser to load and render content. It leads to high latencies and delays page loading, especially when it comes to mobile pages.

How to avoid redirects?

1) Reduce the size of the data, required to render the above-the-fold content.

Here you need to use all those techniques we've discussed above for reducing the amount of data used by your resources:

- Minification of HTML, CSS, and JavaScript;
- Image optimization;
- Compression.

2) Structure your HTML to load the above-the-fold content first.

When you change the HTML markup structure to load the above-the fold-content immediately, it can significantly speed up rendering time. However, what you change should vary from page to page. For example, there can be a need to split your CSS into different parts: an inlined part, responsible for styling the above-the-fold portion of the conteny, and a stylesheet that defers the remaining part. Or, you may need to change the order of what loads on your page first (e.g., main content before widgets).

9. Remove render-blocking JavaScript.

Recommendation Trigger:

HTML references a blocking external JavaScript file in the above-the-fold portion of the page.

Why it matters:

Browsers typically process HTML markup and build the DOM tree before they are able to render a page. Thus, every time a parser encounters JavaScript, it has to stop and execute this new script before it can continue building the DOM tree. The delay is even more critical in case of an external script - it can considerably slow down the rendering process.

How to avoid redirects?

1) Make JavaScript non-render blocking.

When you mark your <script> tag as async, it tells the browser not to block DOM construction while it waits for the script to be loaded and executed. However, bear in mind that you can do this only if you know that you don't need to change anything within the DOM tree in the process of its parsing/constructing.

2) Inline critical scripts and defer non-critical scripts.

Scripts, necessary for rendering, should be inlined into the HTML document to avoid extra network requests. They should be as small as possible to be executed quickly and deliver good performance. At the same time, non-critical scripts should be made asynchronous and deferred until after the first render. Pay attention to the fact that such asynchronous scripts are not guaranteed to execute in a specified order.

3) Defer third-party JavaScript libraries until after the fold.

JavaScript libraries for enhancing interactivity or adding effects (e.g., JQuery) usually do not need to be rendered above the fold. So, make these elements asynchronous and defer them down the page.

Conclusion

Before making any changes, keep in mind that nowadays speed improvement – just like SEO as a whole – is a process. You cannot do it once and forever. So first of all, you need to choose the right metrics and set your goals wisely in line with the steps above, and after that, work continuously on improving your Optimization Score.