Apply instant loading with the PRPL pattern

- Audit your page with Lighthouse
- Preload critical resources
- Render the initial route as soon as possible
- Pre-cache assets
- Lazy load
- Next Steps

PRPL is an acronym that describes a pattern used to make web pages load and become interactive, faster:

- **Preload** the most important resources.
- **Render** the initial route as soon as possible.
- **Pre-cache** remaining assets.
- **Lazy load** other routes and non-critical assets.

In this guide, learn how each of these techniques fit together but still can be used independently to achieve performance results.

Audit your page with Lighthouse

Run Lighthouse to identify opportunities for improvement aligned with the PRPL techniques:

1. Select the **Performance** and **Progressive Web App** checkboxes.
2. Click **Run Audits** to generate a report.

For more information, see Discover performance opportunities with Lighthouse.

Preload critical resources

Lighthouse shows the following failed audit if a certain resource is parsed and fetched late:

Lighthouse: Preload key requests audit

Preload is a declarative fetch request that tells the browser to request a resource as soon as possible. Preload critical resources by adding a `<link>` tag with `rel="preload"` to the head of your HTML document:

```
<link rel="preload" as="style" href="css/style.css">
```

The browser sets a more appropriate priority level for the resource in order to try to download it sooner while not delaying the `window.onload` event.

For more information about preloading critical resources, refer to the Preload critical assets guide.

Render the initial route as soon as possible

Lighthouse provides a warning if there are resources that delay **First Paint**, the moment when your site renders pixels to the screen:

Lighthouse: Eliminate render-blocking resources audit

To improve First Paint, Lighthouse recommends inlining critical JavaScript and deferring the rest using `async`, as well as inlining critical CSS used above-the-fold. This improves performance by eliminating round-trips to the server to fetch render-blocking assets. However, inline code is harder to maintain from a development perspective and cannot be cached separately by the browser.

Another approach to improve First Paint is to **server-side render** the initial HTML of your page. This displays content immediately to the user while scripts are still being fetched, parsed, and executed. However, this can increase the payload of the HTML file significantly, which can harm **Time to Interactive**, or the time it takes for your application to become interactive and can respond to user input.

There is no single correct solution to reduce the First Paint in your application, and you should only consider inlining styles and server-side rendering if the benefits outweigh the tradeoffs for your application. You can learn more about both of these concepts with the following resources.

- Optimize CSS Delivery
- What is Server-Side Rendering?

Requests/responses with service worker

Pre-cache assets

By acting as a proxy, **service workers** can fetch assets directly from the cache rather than the server on repeat visits. This not only allows users to use your application when they are offline, but also results in faster page load times on repeat visits.

Use a third-party library to simplify the process of generating a service worker unless you have more complex caching requirements than what a library can provide. For example, Workbox provides a collection of tools that allow you to create and maintain a service worker to cache assets. For more information on service workers and offline reliability, refer to the service worker guide in the reliability learning path.

Lazy load

Lighthouse displays a failed audit if you send too much data over the network.

Lighthouse: Has enormous network payloads audit

This includes all asset types, but large JavaScript payloads are especially costly due to the time it takes the browser to parse and compile them. Lighthouse also provides a warning for this when appropriate.

Lighthouse: JavaScript boot-up time audit

To send a smaller JavaScript payload that contains only the code needed when a user initially loads your application, split the entire bundle and lazy load chunks on demand.

Once you've managed to split your bundle, preload the chunks that are more important (see the Preload critical assets guide). Preloading ensures more important resources are fetched and downloaded sooner by the browser.

Aside from splitting and loading different JavaScript chunks on demand, Lighthouse also provides an audit for lazy loading non-critical images.

Lighthouse: Defer offscreen images audit

If you load many images on your web page, defer all that are below the fold, or outside the device viewport, when a page is loaded (see Use lazysizes to lazyload images).

Next Steps

Now that you understand some of the basic concepts behind the PRPL pattern, continue to the next guide in this section to learn more. It's important to remember that not all of the techniques need to be applied together. Any efforts made with any of the following will provide noticeable performance improvements.

- **Preload** critical resources.
- **Render** the initial route as soon as possible.
- **Pre-cache** remaining assets.
- **Lazy load** other routes and non-critical assets.

You can read up more about PRPL patterns.

□□□ #3
□ □□□ □□□ 2026-10-31 22:33:01 UTC
□ cocobook-sync □□□ 2026-11-04 17:38:43 UTC