- **Mobile-First CSS & Testing:** Write your CSS with mobile in mind first (e.g., use relative units, avoid large fixed widths). Test the site on actual phones or emulators: ensure text is legible without zoom, tap targets (links, toggle button) are finger-friendly, and that layout doesn't break on small screens. Use media queries to progressively enhance for larger screens (e.g., show a multi-column grid only when min-width is say 600px). This strategy ensures the base CSS is lightweight and focused. It also helps prevent large-screen styles from being unnecessarily sent to mobile. Tools like Chrome DevTools or online testers can simulate slow network and device throttling – use these to see how the site behaves on 3G speeds and mid-range phones.

- **Performance Testing & Tuning:** Once your site is built, run performance tests (Lighthouse in Chrome, or PageSpeed Insights). These will highlight any bottlenecks. Common things to address:

- **Initial Load:** Aim for a quick Time to First Byte and Start Render. With static hosting, TTFB is usually low. Start Render depends on not having large render-blocking resources. This means inlining critical CSS (above-the-fold styles) could help, or just keeping CSS small and in the head. Ensure any non-critical scripts are loaded deferred or at the bottom.
- **Core Web Vitals:** Check metrics like LCP (Largest Contentful Paint) – often images are the largest element, so optimize their size/format. Also CLS (Layout Shift) – reserve space for images with width/height attributes or CSS aspect-ratio to avoid shift as they load. And TTI (Time to Interactive) – with minimal JS, this should be near-instant.
- **Caching:** GitHub Pages will generally serve static files with ETags. You can add `<meta http-equiv="Cache-Control" ...>` or better, use proper HTTP headers if possible to leverage browser caching. Since your content doesn't change often, you can set long cache times on assets (or use file name hashing for cache-busting when you do update).

By aggressively optimizing, you'll ensure the site **feels instantaneous**, especially on mobile. This not only pleases visitors but also improves SEO (Google rewards fast, mobile-friendly sites [22] [23]).

4. Light CMS Options for Easy Content Updates

Even though you don't need a full CMS (since you update infrequently), you might appreciate some lightweight tools to edit content without digging into code each time:

- **Netlify CMS (Decap CMS):** This is an open-source CMS that adds a **user-friendly admin interface to static sites** by abstracting the Git workflow [24] . You set up an `/admin/` page on your site, and through that you can log in (with GitHub or Netlify Identity) and edit content in a rich text editor. Netlify CMS will commit changes to your Git repository behind the scenes. This can be a great middle-ground if you want to update text or add new projects via a browser. It works with Jekyll, Eleventy, Hugo, etc. (It was literally *designed to add a CMS-like interface to a Git-based static site* [24] .) For your use, you'd configure it to edit your projects collection (Markdown/YAML files). The CMS UI is lightweight and lives on your site – no external database. Note: if you host on GitHub Pages, you can still use Netlify CMS by using GitHub authentication, but it's often simplest if you host on Netlify. One approach is to use GitHub for code storage and Netlify for deploying the site (Netlify can pull from GitHub and also provide the CMS auth out-of-the-box). This setup would give you free hosting, and an editing UI if needed.