Speed up navigations in React with Quicklink

- Determine chunks associated with each route
- Prefetch chunks for in-viewport routes
- Conclusion

Prefetching is a technique to speed up navigations by downloading the resources for the next page, ahead of time. Quicklink is a library that allows you to implement this technique at scale, by automatically prefetching links as they come into the view.

In multi-page apps the library prefetches documents (e.g. `/article.html`), for in-viewport links, so that when the user clicks on these links they can be picked up from the HTTP cache.

Single-page apps commonly use a technique called route-based code splitting. This allows the site to load the code for a given route only when the user navigates to it. These files (JS, CSS) are commonly referred to as "chunks".

With that said, in these sites, instead of prefetching documents the biggest performance gains come from prefetching these chunks before the page needs them.

Achieving this presents some challenges:

- It's not trivial to determine which chunks (e.g. `article.chunk.js`) are associated with a given route (e.g. `/article`) before landing on it.
- The final URL names of these chunks can't be predicted, as modern module bundlers typically use long-term hashing for versioning (e.g. `article.chunk.46e51.js`).

This guide explains how Quicklink solves these challenges and allows you to achieve prefetching at scale in React single page apps.

At the moment this solution is only compatible with react-router.

Check out the Prefetching in create-react-app with Quicklink codelab for a guided, hands-on demonstration of Quicklink.

Determine chunks associated with each route

One of the core components of `quicklink` is webpack-route-manifest, a webpack plugin that lets you generate a JSON dictionary of routes and chunks. This allows the library to know which files are going to be needed by each route of the application and prefetch them as the routes come into the view.

After integrating the plugin with the project, it will produce a JSON manifest file associating each route with its corresponding chunks:

```
{
  '/about': [
    {
      type: 'style',
      href: '/static/css/about.f6fd7d80.chunk.css',
    },
    {
      type: 'script',
      href: '/static/js/about.1cdfef3b.chunk.js',
    },
  ],
  '/blog': [
    {
      type: 'style',
      href: '/static/css/blog.85e80e75.chunk.css',
    },
    {
      type: 'script',
      href: '/static/js/blog.35421503.chunk.js',
    },
  ],
}
```

This manifest file can be requested in two ways:

- By URL, e.g. `https://site_url/rmanifest.json`.
- Through the window object, at `window.__rmanifest`.

Prefetch chunks for in-viewport routes

Once the manifest file is available, the next step is to install Quicklink by running `npm install quicklink`.

Then, the higher order component (HOC) `withQuicklink()` can be used to indicate that a given route should be prefetched when the link comes into the view.

The following code belongs to an `App` component of a React app that renders a top menu with four links:

```
const App = () => (
  <div className={style.app}>
    <Hero />
    <main className={style.wrapper}>
      <Suspense fallback={<div>Loading…</div>}>
        <Route path="/" exact component={Home} />
        <Route path="/blog" exact component={Blog} />
        <Route path="/blog/:title" component={Article} />
        <Route path="/about" exact component={About} />
      </Suspense>
    </main>
    <Footer />
  </div>
);
```

To tell Quicklink that these routes should be prefetched as they come into the view:

1. Import the `quicklink` HOC at the beginning of the component.
2. Wrap each route with the `withQuicklink()` HOC, passing the page component and options parameter to it.

```
const options = {
  origins: [],
};
const App = () => (
  <div className={style.app}>
    <Hero />
    <main className={style.wrapper}>
      <Suspense fallback={<div>Loading…</div>}>
        <Route path="/" exact component={withQuicklink(Home, options)} />
        <Route path="/blog" exact component={withQuicklink(Blog, options)} />
        <Route
          path="/blog/:title"
          component={withQuicklink(Article, options)}
        />
        <Route path="/about" exact component={withQuicklink(About, options)} />
      </Suspense>
    </main>
    <Footer />
  </div>
);
```

The `withQuicklink()` HOC uses the path of the route as a key to obtain its associated chunks from `rmanifest.json`. Under the hood, as links come into the view, the library injects a `<link rel="prefetch">` tag in the page for each chunk so they can be prefetched. Prefetched resources will be requested at the lowest priority by the browser and kept in the HTTP Cache for 5 minutes, after which point, the `cache-control` rules of the resource apply. As a result of this, when a user clicks on a link and moves to a given route, the chunks will be retrieved from the cache, greatly improving the time it takes to render that route.

Conclusion

Prefetching can greatly improve load times for future navigations. In React single-page apps, this can be achieved by loading the chunks associated with each route, before the user lands in them. Quicklink's solution for React SPA uses `webpack-route-manifest` to create a map of routes and chunks, in order to determine which files to prefetch, when a link comes into the view. Implementing this technique throughout your site can greatly improve navigations to the point of making them appear instant.

□□□ #3
□ □□□ □□□ 2026-10-31 23:11:08 UTC
□ cocobook-sync □□□ 2026-11-04 17:39:51 UTC