https://benco.dev/

Performance	Accessibility	Best Practices	SEO	Progressive Web App
98	100	100	100	—

0–49 50–89 90–100

98

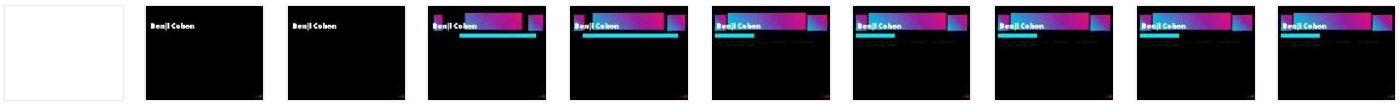
Performance

Metrics

First Contentful Paint	0.7 s	Time to Interactive	0.8 s
Speed Index	0.7 s	Total Blocking Time	0 ms
Largest Contentful Paint	0.9 s	Cumulative Layout Shift	0.094

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

View Original Trace



Opportunities — These suggestions can help your page load faster. They don't directly affect the Performance score.

Opportunity	Estimated Savings
Eliminate render-blocking resources	0.4 s

Resources are blocking the first paint of your page. Consider delivering critical JS/CSS inline and deferring all non-critical JS/styles. Learn more.

~~Show 3rd-party resources~~ (1)

URL	Transfer Size	Potential Savings
/puh4jzz.css (use.typekit.net)	0.9 KiB	240 ms

Opportunity	Estimated Savings
Remove unused JavaScript	Error!

Remove unused JavaScript to reduce bytes consumed by network activity. Learn more.

If you are not server-side rendering, split your JavaScript bundles with `React.lazy()`. Otherwise, code-split using a third-party library such as loadable-components.

Opportunity	Estimated Savings
Remove duplicate modules in JavaScript bundles	Error!

Remove large, duplicate JavaScript modules from bundles to reduce unnecessary bytes consumed by network activity.

▲ Avoid serving legacy JavaScript to modern browsers — Error!

Polyfills and transforms enable legacy browsers to use new JavaScript features. However, many aren't necessary for modern browsers. For your bundled JavaScript, adopt a modern script deployment strategy using module/nomodule feature detection to reduce the amount of code shipped to modern browsers, while retaining support for legacy browsers. Learn More

Diagnostics — More information about the performance of your application. These numbers don't directly affect the Performance score.

▲ Ensure text remains visible during webfont load

Leverage the font-display CSS feature to ensure text is user-visible while webfonts are loading. Learn more.

~~Show 3rd-party resources~~ (4)

URL	Potential Savings
…27/l?primer=7cdcb44…&fvd=n3&v=3 (use.typekit.net)	80 ms
…27/l?subset_id=2&fvd=n6&v=3 (use.typekit.net)	90 ms
…27/l?subset_id=2&fvd=n8&v=3 (use.typekit.net)	80 ms
…27/l?primer=7cdcb44…&fvd=n4&v=3 (use.typekit.net)	80 ms

● Avoid chaining critical requests — 8 chains found

The Critical Request Chains below show you what resources are loaded with a high priority. Consider reducing the length of chains, reducing the download size of resources, or deferring the download of unnecessary resources to improve page load. Learn more.

Maximum critical path latency: **490 ms**

Initial Navigation

- https://benco.dev
  - /puh4jzz.css (use.typekit.net)
    - /p.css?s=… (p.typekit.net) **- 280 ms, 0.15 KiB**
    - …27/l?primer=7cdcb44…&fvd=n3&v=3 (use.typekit.net) **- 80 ms, 82.31 KiB**
    - …27/l?subset_id=2&fvd=n6&v=3 (use.typekit.net) **- 90 ms, 40.22 KiB**
    - …27/l?subset_id=2&fvd=n8&v=3 (use.typekit.net) **- 80 ms, 61.07 KiB**
    - …27/l?primer=7cdcb44…&fvd=n4&v=3 (use.typekit.net) **- 80 ms, 100.98 KiB**
  - …css/main.0a17ce1f.chunk.css (benco.dev) **- 40 ms, 6.82 KiB**
  - …js/2.23e95338.chunk.js (benco.dev) **- 100 ms, 45.67 KiB**
  - …js/main.d5b02c4e.chunk.js (benco.dev) **- 70 ms, 5.24 KiB**

● Keep request counts low and transfer sizes small — 13 requests • 483 KiB

To set budgets for the quantity and size of page resources, add a budget.json file. Learn more.

Resource Type	Requests	Transfer Size
Total	13	483.4 KiB
Font	4	284.6 KiB

Resource Type	Requests	Transfer Size
Image	1	127.6 KiB
Script	2	50.9 KiB
Other	2	11.1 KiB
Stylesheet	3	7.9 KiB
Document	1	1.3 KiB
Media	0	0 KiB
Third-party	6	285.7 KiB

Largest Contentful Paint element — 1 element found

This is the largest contentful element painted within the viewport. Learn More

Element
p

Avoid large layout shifts — 5 elements found

These DOM elements contribute most to the CLS of the page.

Element	CLS Contribution
::after <::after>	0.056
::before <::before>	0.02
::after <::after>	0.012
::after <::after>	0.006
h1.title	0

Avoid non-composited animations — 9 animated elements found

Animations which are not composited can be janky and contribute to CLS. Learn more

Element	Name
::before <::before>	
Unsupported CSS Property	TitleSlideIn
::after <::after>	
Unsupported CSS Property	TitleSlideInAlt
::before <::before>	
Unsupported CSS Property	TitleSlideIn

Element	Name
::after <::after>	
Unsupported CSS Property	TitleSlideInAlt
::before <::before>	
Unsupported CSS Property	TitleSlideIn
::after <::after>	
Unsupported CSS Property	TitleSlideInAlt
h2.subtitle	
Unsupported CSS Property	ColorFadeIn
::before <::before>	
Unsupported CSS Property	AccentSlideIn
div.copy	
Unsupported CSS Property	ColorFadeIn

Passed audits (25)

- Properly size images

  Serve images that are appropriately-sized to save cellular data and improve load time. Learn more.

- Defer offscreen images

  Consider lazy-loading offscreen and hidden images after all critical resources have finished loading to lower time to interactive. Learn more.

- Minify CSS

  Minifying CSS files can reduce network payload sizes. Learn more.

  If your build system minifies your CSS files automatically, ensure that you are deploying the production build of your application. You can check this with the React Developer Tools extension. Learn more.

- Minify JavaScript

  Minifying JavaScript files can reduce payload sizes and script parse time. Learn more.

  If your build system minifies your JS files automatically, ensure that you are deploying the production build of your application. You can check this with the React Developer Tools extension. Learn more.

- Remove unused CSS

  Remove dead rules from stylesheets and defer the loading of CSS not used for above-the-fold content to reduce unnecessary bytes consumed by network activity. Learn more.

- Efficiently encode images

  Optimized images load faster and consume less cellular data. Learn more.

- Serve images in next-gen formats

  Image formats like JPEG 2000, JPEG XR, and WebP often provide better compression than PNG or JPEG, which means faster downloads and less data consumption. Learn more.

- Enable text compression

  Text-based resources should be served with compression (gzip, deflate or brotli) to minimize total network bytes. Learn more.

- Preconnect to required origins — Potential savings of 80 ms

  Consider adding `preconnect` or `dns-prefetch` resource hints to establish early connections to important third-party origins. Learn more.

URL	Potential Savings
https://use.typekit.net	80 ms

- Initial server response time was short — Root document took 40 ms

  Keep the server response time for the main document short because all other requests depend on it. Learn more.

  If you are server-side rendering any React components, consider using `renderToNodeStream()` or `renderToStaticNodeStream()` to allow the client to receive and hydrate different parts of the markup instead of all at once. Learn more.

- Avoid multiple page redirects

  Redirects introduce additional delays before the page can be loaded. Learn more.

  If you are using React Router, minimize usage of the `<Redirect>` component for route navigations.

- Preload key requests

  Consider using `<link rel=preload>` to prioritize fetching resources that are currently requested later in page load. Learn more.

- Use HTTP/2

  HTTP/2 offers many benefits over HTTP/1.1, including binary headers, multiplexing, and server push. Learn more.

- Use video formats for animated content

  Large GIFs are inefficient for delivering animated content. Consider using MPEG4/WebM videos for animations and PNG/WebP for static images instead of GIF to save network bytes. Learn more

- Avoids enormous network payloads — Total size was 483 KiB

  Large network payloads cost users real money and are highly correlated with long load times. Learn more.

  Show 3rd-party resources (4)

URL	Transfer Size
/images/mocha-grunge.png (benco.dev)	127.6 KiB
…27/l?primer=7cdcb44…&fvd=n4&v=3 (use.typekit.net)	101 KiB
…27/l?primer=7cdcb44…&fvd=n3&v=3 (use.typekit.net)	82.3 KiB
…27/l?subset_id=2&fvd=n8&v=3 (use.typekit.net)	61.1 KiB
…js/2.23e95338.chunk.js (benco.dev)	45.7 KiB
…27/l?subset_id=2&fvd=n6&v=3 (use.typekit.net)	40.2 KiB
/logo192.png (benco.dev)	10.5 KiB
…css/main.0a17ce1f.chunk.css (benco.dev)	6.8 KiB

URL	Transfer Size
…js/main.d5b02c4e.chunk.js (benco.dev)	5.2 KiB
https://benco.dev	1.3 KiB

Uses efficient cache policy on static assets — 1 resource found

A long cache lifetime can speed up repeat visits to your page. Learn more.

Show 3rd-party resources (1)

URL	Cache TTL	Transfer Size
/p.css?s=… (p.typekit.net)	7 d	0 KiB

Avoids an excessive DOM size — 51 elements

A large DOM will increase memory usage, cause longer style calculations, and produce costly layout reflows. Learn more.

Consider using a “windowing” library like `react-window` to minimize the number of DOM nodes created if you are rendering many repeated elements on the page. Learn more. Also, minimize unecessary re-renders using shouldComponentUpdate, PureComponent, or React.memo and skip effects only until certain dependencies have changed if you are using the Effect hook to improve runtime performance.

Statistic	Element	Value
Total DOM Elements		51
Maximum DOM Depth	`<a aria-current="page" class="active" href="/">`	6
Maximum Child Elements	`<body page="Home">`	6

User Timing marks and measures

Consider instrumenting your app with the User Timing API to measure your app's real-world performance during key user experiences. Learn more.

Use the React DevTools Profiler, which makes use of the Profiler API, to measure the rendering performance of your components. Learn more.

JavaScript execution time — 0.0 s

Consider reducing the time spent parsing, compiling, and executing JS. You may find delivering smaller JS payloads helps with this. Learn more.

Show 3rd-party resources (0)

URL	Total CPU Time	Script Evaluation	Script Parse
https://benco.dev	79 ms	12 ms	3 ms

Minimizes main-thread work — 0.2 s

Consider reducing the time spent parsing, compiling and executing JS. You may find delivering smaller JS payloads helps with this. Learn more

Category	Time Spent
Other	48 ms
Script Evaluation	38 ms

Category	Time Spent
Style & Layout	26 ms
Script Parsing & Compilation	20 ms
Rendering	17 ms
Parse HTML & CSS	5 ms
Garbage Collection	1 ms

- Minimize third-party usage — Third-party code blocked the main thread for 0 ms

Third-party code can significantly impact load performance. Limit the number of redundant third-party providers and try to load third-party code after your page has primarily finished loading. Learn more.

Third-Party	Transfer Size	Main-Thread Blocking Time
Adobe TypeKit	286 KiB	0 ms

- Uses passive listeners to improve scrolling performance

Consider marking your touch and wheel event listeners as `passive` to improve your page's scroll performance. Learn more.

- Avoids `document.write()`

For users on slow connections, external scripts dynamically injected via `document.write()` can delay page load by tens of seconds. Learn more.

- Avoid long main-thread tasks

Lists the longest tasks on the main thread, useful for identifying worst contributors to input delay. Learn more

- Image elements have explicit `width` and `height`

Set an explicit width and height on image elements to reduce layout shifts and improve CLS. Learn more

Accessibility

These checks highlight opportunities to improve the accessibility of your web app. Only a subset of accessibility issues can be automatically detected so manual testing is also encouraged.

Additional items to manually check (10) — These items address areas which an automated testing tool cannot cover. Learn more in our guide on conducting an accessibility review.

- The page has a logical tab order

Tabbing through the page follows the visual layout. Users cannot focus elements that are offscreen. Learn more.

- Interactive controls are keyboard focusable

Custom interactive controls are keyboard focusable and display a focus indicator. Learn more.

- Interactive elements indicate their purpose and state

Interactive elements, such as links and buttons, should indicate their state and be distinguishable from non-interactive elements. Learn more.

- The user's focus is directed to new content added to the page

  If new content, such as a dialog, is added to the page, the user's focus is directed to it. Learn more.

- User focus is not accidentally trapped in a region

  A user can tab into and out of any control or region without accidentally trapping their focus. Learn more.

- Custom controls have associated labels

  Custom interactive controls have associated labels, provided by aria-label or aria-labelledby. Learn more.

- Custom controls have ARIA roles

  Custom interactive controls have appropriate ARIA roles. Learn more.

- Visual order on the page follows DOM order

  DOM order matches the visual order, improving navigation for assistive technology. Learn more.

- Offscreen content is hidden from assistive technology

  Offscreen content is hidden with display: none or aria-hidden=true. Learn more.

- HTML5 landmark elements are used to improve navigation

  Landmark elements (<main>, <nav>, etc.) are used to improve the keyboard navigation of the page for assistive technology. Learn more.

Passed audits (14)

- `[aria-*]` attributes match their roles

  Each ARIA `role` supports a specific subset of `aria-*` attributes. Mismatching these invalidates the `aria-*` attributes. Learn more.

- `[aria-hidden="true"]` is not present on the document `<body>`

  Assistive technologies, like screen readers, work inconsistently when `aria-hidden="true"` is set on the document `<body>`. Learn more.

- `[aria-*]` attributes have valid values

  Assistive technologies, like screen readers, can't interpret ARIA attributes with invalid values. Learn more.

- `[aria-*]` attributes are valid and not misspelled

  Assistive technologies, like screen readers, can't interpret ARIA attributes with invalid names. Learn more.

- The page contains a heading, skip link, or landmark region

  Adding ways to bypass repetitive content lets keyboard users navigate the page more efficiently. Learn more.

- Background and foreground colors have a sufficient contrast ratio

  Low-contrast text is difficult or impossible for many users to read. Learn more.

- Document has a `<title>` element

  The title gives screen reader users an overview of the page, and search engine users rely on it heavily to determine if a page is relevant to their search. Learn more.

- Heading elements appear in a sequentially-descending order

  Properly ordered headings that do not skip levels convey the semantic structure of the page, making it easier to navigate and understand when using assistive technologies. Learn more.

- `<html>` element has a `[lang]` attribute

  If a page doesn't specify a lang attribute, a screen reader assumes that the page is in the default language that the user chose when setting up the screen reader. If the page isn't actually in the default language, then the screen reader might not announce the page's text correctly. Learn more.

- `<html>` element has a valid value for its `[lang]` attribute

  Specifying a valid BCP 47 language helps screen readers announce text properly. Learn more.

- Links have a discernible name

  Link text (and alternate text for images, when used as links) that is discernible, unique, and focusable improves the navigation experience for screen reader users. Learn more.

- Lists contain only `<li>` elements and script supporting elements (`<script>` and `<template>`).

  Screen readers have a specific way of announcing lists. Ensuring proper list structure aids screen reader output. Learn more.

- List items (`<li>`) are contained within `<ul>` or `<ol>` parent elements

  Screen readers require list items (`<li>`) to be contained within a parent `<ul>` or `<ol>` to be announced properly. Learn more.

- `[user-scalable="no"]` is not used in the `<meta name="viewport">` element and the `[maximum-scale]` attribute is not less than 5.

  Disabling zooming is problematic for users with low vision who rely on screen magnification to properly see the contents of a web page. Learn more.

Not applicable (27)

- `[accesskey]` values are unique

  Access keys let users quickly focus a part of the page. For proper navigation, each access key must be unique. Learn more.

- `[aria-hidden="true"]` elements do not contain focusable descendents

  Focusable descendents within an `[aria-hidden="true"]` element prevent those interactive elements from being available to users of assistive technologies like screen readers. Learn more.

- ARIA input fields have accessible names

  When an input field doesn't have an accessible name, screen readers announce it with a generic name, making it unusable for users who rely on screen readers. Learn more.

- `[role]`s have all required `[aria-*]` attributes

  Some ARIA roles have required attributes that describe the state of the element to screen readers. Learn more.

- Elements with an ARIA `[role]` that require children to contain a specific `[role]` have all required children.

  Some ARIA parent roles must contain specific child roles to perform their intended accessibility functions. Learn more.

- `[role]`s are contained by their required parent element

  Some ARIA child roles must be contained by specific parent roles to properly perform their intended accessibility functions. Learn more.

- [role] values are valid

  ARIA roles must have valid values in order to perform their intended accessibility functions. Learn more.

- ARIA toggle fields have accessible names

  When a toggle field doesn't have an accessible name, screen readers announce it with a generic name, making it unusable for users who rely on screen readers. Learn more.

- Buttons have an accessible name

  When a button doesn't have an accessible name, screen readers announce it as "button", making it unusable for users who rely on screen readers. Learn more.

- <dl>'s contain only properly-ordered <dt> and <dd> groups, <script>, <template> or elements.

  When definition lists are not properly marked up, screen readers may produce confusing or inaccurate output. Learn more.

- Definition list items are wrapped in <dl> elements

  Definition list items (`<dt>` and `<dd>`) must be wrapped in a parent `<dl>` element to ensure that screen readers can properly announce them. Learn more.

- [id] attributes on active, focusable elements are unique

  All focusable elements must have a unique `id` to ensure that they're visible to assistive technologies. Learn more.

- ARIA IDs are unique

  The value of an ARIA ID must be unique to prevent other instances from being overlooked by assistive technologies. Learn more.

- No form fields have multiple labels

  Form fields with multiple labels can be confusingly announced by assistive technologies like screen readers which use either the first, the last, or all of the labels. Learn more.

- <frame> or <iframe> elements have a title

  Screen reader users rely on frame titles to describe the contents of frames. Learn more.

- Image elements have [alt] attributes

  Informative elements should aim for short, descriptive alternate text. Decorative elements can be ignored with an empty alt attribute. Learn more.

- <input type="image"> elements have [alt] text

  When an image is being used as an `<input>` button, providing alternative text can help screen reader users understand the purpose of the button. Learn more.

- Form elements have associated labels

  Labels ensure that form controls are announced properly by assistive technologies, like screen readers. Learn more.

- Presentational <table> elements avoid using <th>, <caption> or the [summary] attribute.

  A table being used for layout purposes should not include data elements, such as the th or caption elements or the summary attribute, because this can create a confusing experience for screen reader users. Learn more.

- The document does not use <meta http-equiv="refresh">

  Users do not expect a page to refresh automatically, and doing so will move focus back to the top of the page. This may create a frustrating or confusing experience. Learn more.

- <object> elements have [alt] text

Screen readers cannot translate non-text content. Adding alt text to `<object>` elements helps screen readers convey meaning to users. Learn more.

- No element has a `[tabindex]` value greater than 0

  A value greater than 0 implies an explicit navigation ordering. Although technically valid, this often creates frustrating experiences for users who rely on assistive technologies. Learn more.

- Cells in a `<table>` element that use the `[headers]` attribute refer to table cells within the same table.

  Screen readers have features to make navigating tables easier. Ensuring `<td>` cells using the `[headers]` attribute only refer to other cells in the same table may improve the experience for screen reader users. Learn more.

- `<th>` elements and elements with `[role="columnheader"/"rowheader"]` have data cells they describe.

  Screen readers have features to make navigating tables easier. Ensuring table headers always refer to some set of cells may improve the experience for screen reader users. Learn more.

- `[lang]` attributes have a valid value

  Specifying a valid BCP 47 language on elements helps ensure that text is pronounced correctly by a screen reader. Learn more.

- `<video>` elements contain a `<track>` element with `[kind="captions"]`

  When a video provides a caption it is easier for deaf and hearing impaired users to access its information. Learn more.

- `<video>` elements contain a `<track>` element with `[kind="description"]`

  Audio descriptions provide relevant information for videos that dialogue cannot, such as facial expressions and scenes. Learn more.

Best Practices

Passed audits (15)

- Uses HTTPS

  All sites should be protected with HTTPS, even ones that don't handle sensitive data. This includes avoiding mixed content, where some resources are loaded over HTTP despite the initial request being servedover HTTPS. HTTPS prevents intruders from tampering with or passively listening in on the communications between your app and your users, and is a prerequisite for HTTP/2 and many new web platform APIs. Learn more.

- Links to cross-origin destinations are safe

  Add `rel="noopener"` or `rel="noreferrer"` to any external links to improve performance and prevent security vulnerabilities. Learn more.

- Avoids requesting the geolocation permission on page load

  Users are mistrustful of or confused by sites that request their location without context. Consider tying the request to a user action instead. Learn more.

- Avoids requesting the notification permission on page load

Users are mistrustful of or confused by sites that request to send notifications without context. Consider tying the request to user gestures instead. Learn more.

- Avoids front-end JavaScript libraries with known security vulnerabilities

  Some third-party scripts may contain known security vulnerabilities that are easily identified and exploited by attackers. Learn more.

- Allows users to paste into password fields

  Preventing password pasting undermines good security policy. Learn more.

- Displays images with correct aspect ratio

  Image display dimensions should match natural aspect ratio. Learn more.

- Serves images with appropriate resolution

  Image natural dimensions should be proportional to the display size and the pixel ratio to maximize image clarity. Learn more.

- Page has the HTML doctype

  Specifying a doctype prevents the browser from switching to quirks-mode. Learn more.

- Properly defines charset

  A character encoding declaration is required. It can be done with a <meta> tag in the first 1024 bytes of the HTML or in the Content-Type HTTP response header. Learn more.

- Avoids `unload` event listeners

  The `unload` event does not fire reliably and listening for it can prevent browser optimizations like the Back-Forward Cache. Consider using the `pagehide` or `visibilitychange` events instead. Learn More

- Avoids Application Cache

  Application Cache is deprecated. Learn more.

- Detected JavaScript libraries

  All front-end JavaScript libraries detected on the page. Learn more.

Name	Version
React	
Create React App	

- Avoids deprecated APIs

  Deprecated APIs will eventually be removed from the browser. Learn more.

- No browser errors logged to the console

  Errors logged to the console indicate unresolved problems. They can come from network request failures and other browser concerns. Learn more

SEO

These checks ensure that your page is optimized for search engine results ranking. There are additional factors Lighthouse does not check that may affect your search ranking. Learn more.

Additional items to manually check (1) — Run these additional validators on your site to check additional SEO best practices.

- Structured data is valid

  Run the Structured Data Testing Tool and the Structured Data Linter to validate structured data. Learn more.

Passed audits (10)

- Has a `<meta name="viewport">` tag with `width` or `initial-scale`

  Add a `<meta name="viewport">` tag to optimize your app for mobile screens. Learn more.

- Document has a `<title>` element

  The title gives screen reader users an overview of the page, and search engine users rely on it heavily to determine if a page is relevant to their search. Learn more.

- Document has a meta description

  Meta descriptions may be included in search results to concisely summarize page content. Learn more.

- Page has successful HTTP status code

  Pages with unsuccessful HTTP status codes may not be indexed properly. Learn more.

- Links have descriptive text

  Descriptive link text helps search engines understand your content. Learn more.

- Links are crawlable

  Search engines may use `href` attributes on links to crawl websites. Ensure that the `href` attribute of anchor elements links to an appropriate destination, so more pages of the site can be discovered. Learn More

- Page isn't blocked from indexing

  Search engines are unable to include your pages in search results if they don't have permission to crawl them. Learn more.

- robots.txt is valid

  If your robots.txt file is malformed, crawlers may not be able to understand how you want your website to be crawled or indexed. Learn more.

- Document has a valid `hreflang`

  hreflang links tell search engines what version of a page they should list in search results for a given language or region. Learn more.

- Document avoids plugins

  Search engines can't index plugin content, and many devices restrict plugins or don't support them. Learn more.

Not applicable (4)

- Image elements have `[alt]` attributes

  Informative elements should aim for short, descriptive alternate text. Decorative elements can be ignored with an empty alt attribute. Learn more.

- Document has a valid `rel=canonical`

  Canonical links suggest which URL to show in search results. Learn more.

- Document uses legible font sizes

  Font sizes less than 12px are too small to be legible and require mobile visitors to “pinch to zoom” in order to read. Strive to have >60% of page text ≥12px. Learn more.

- Tap targets are sized appropriately

  Interactive elements like buttons and links should be large enough (48x48px), and have enough space around them, to be easy enough to tap without overlapping onto other elements. Learn more.

Progressive Web App

These checks validate the aspects of a Progressive Web App. Learn more.

Fast and reliable

- Page load is fast enough on mobile networks

  A fast page load over a cellular network ensures a good mobile user experience. Learn more.

- Current page does not respond with a 200 when offline

  If you're building a Progressive Web App, consider using a service worker so that your app can work offline. Learn more.

- `start_url` does not respond with a 200 when offline Timed out waiting for start_url (https://benco.dev/) to respond.

  A service worker enables your web app to be reliable in unpredictable network conditions. Learn more.

Installable

- Uses HTTPS

  All sites should be protected with HTTPS, even ones that don't handle sensitive data. This includes avoiding mixed content, where some resources are loaded over HTTP despite the initial request being servedover HTTPS. HTTPS prevents intruders from tampering with or passively listening in on the communications between your app and your users, and is a prerequisite for HTTP/2 and many new web platform APIs. Learn more.

- Does not register a service worker that controls page and `start_url`

  The service worker is the technology that enables your app to use many Progressive Web App features, such as offline, add to homescreen, and push notifications. Learn more.

- Web app manifest meets the installability requirements

  Browsers can proactively prompt users to add your app to their homescreen, which can lead to higher engagement. Learn more.

PWA Optimized

- Redirects HTTP traffic to HTTPS

  If you've already set up HTTPS, make sure that you redirect all HTTP traffic to HTTPS in order to enable secure web features for all your users. Learn more.

- Configured for a custom splash screen

  A themed splash screen ensures a high-quality experience when users launch your app from their homescreens. Learn more.

- Sets a theme color for the address bar.

  The browser address bar can be themed to match your site. Learn more.

- Content is sized correctly for the viewport

  If the width of your app's content doesn't match the width of the viewport, your app might not be optimized for mobile screens. Learn more.

- Has a `<meta name="viewport">` tag with `width` or `initial-scale`

  Add a `<meta name="viewport">` tag to optimize your app for mobile screens. Learn more.

- Contains some content when JavaScript is not available

  Your app should display some content when JavaScript is disabled, even if it's just a warning to the user that JavaScript is required to use the app. Learn more.

- Provides a valid `apple-touch-icon`

  For ideal appearance on iOS when users add a progressive web app to the home screen, define an `apple-touch-icon`. It must point to a non-transparent 192px (or 180px) square PNG. Learn More.

- ▲ Manifest doesn't have a maskable icon

  A maskable icon ensures that the image fills the entire shape without being letterboxed when installing the app on a device. Learn more.

Additional items to manually check (3) — These checks are required by the baseline PWA Checklist but are not automatically checked by Lighthouse. They do not affect your score but it's important that you verify them manually.

- Site works cross-browser

  To reach the most number of users, sites should work across every major browser. Learn more.

- Page transitions don't feel like they block on the network

  Transitions should feel snappy as you tap around, even on a slow network. This experience is key to a user's perception of performance. Learn more.

- Each page has a URL

  Ensure individual pages are deep linkable via URL and that URLs are unique for the purpose of shareability on social media. Learn more.

Runtime Settings

URL	https://benco.dev/

Fetch Time	Oct 24, 2020, 1:00 PM GMT+1
Device	Emulated Desktop
Network throttling	40 ms TCP RTT, 10,240 Kbps throughput (Simulated)
CPU throttling	1x slowdown (Simulated)
Channel	devtools
User agent (host)	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.111 Safari/537.36
User agent (network)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4143.7 Safari/537.36 Chrome-Lighthouse
CPU/Memory Power	2424