SVELTEKIT • BEST PRACTICES

SEO

ON THIS PAGE

The most important aspect of SEO is to create high-quality content that is widely linked to from around the web. However, there are a few technical considerations for building sites that rank well.

Out of the box

SSR

While search engines have got better in recent years at indexing content that was rendered with client-side JavaScript, server-side rendered content is indexed more frequently and reliably. SvelteKit employs SSR by default, and while you can disable it in `handle`, you should leave it on unless you have a good reason not to.

> SvelteKit's rendering is highly configurable and you can implement dynamic rendering if necessary. It's not generally recommended, since SSR has other benefits beyond SEO.

Performance

Signals such as Core Web Vitals impact search engine ranking. Because Svelte and SvelteKit introduce minimal overhead, it's easier to build high performance sites. You can test your site's performance using Google's PageSpeed Insights or Lighthouse. Read the performance page for more details.

on your configuration), as duplicate URLs are bad for SEO.

Manual setup

<title> and <meta>

Every page should have well-written and unique `<title>` and `<meta name="description">` elements inside a `<svelte:head>`. Guidance on how to write descriptive titles and descriptions, along with other suggestions on making content understandable by search engines, can be found on Google's Lighthouse SEO audits documentation.

> A common pattern is to return SEO-related `data` from page `load` functions, then use it (as `$page.data`) in a `<svelte:head>` in your root layout.

Sitemaps

Sitemaps help search engines prioritize pages within your site, particularly when you have a large amount of content. You can create a sitemap dynamically using an endpoint:

src/routes/sitemap.xml/+server.js

```
export async function GET() {
  return new Response(
    `
    <?xml version="1.0" encoding="UTF-8" ?>
    <urlset
      xmlns="https://www.sitemaps.org/schemas/sitemap/0.9"
      xmlns:xhtml="https://www.w3.org/1999/xhtml"
      xmlns:mobile="https://www.google.com/schemas/sitemap-mobile/1.0"
      xmlns:news="https://www.google.com/schemas/sitemap-news/0.9"
      xmlns:image="https://www.google.com/schemas/sitemap-image/1.1"
      xmlns:video="https://www.google.com/schemas/sitemap-video/1.1"
    >
      <!-- <url> elements go here -->
    </urlset>`.trim()
```

```
    }
  );
}
```

AMP

An unfortunate reality of modern web development is that it is sometimes necessary to create an Accelerated Mobile Pages (AMP) version of your site. In SvelteKit this can be done by setting the `inlineStyleThreshold` option...

svelte.config.js

```
/** @type {import('@sveltejs/kit').Config} */
const config = {
  kit: {
    // since <link rel="stylesheet"> isn't
    // allowed, inline all styles
    inlineStyleThreshold: Infinity
  }
};

export default config;
```

...disabling `csr` in your root `+layout.js` / `+layout.server.js` ...

src/routes/+layout.server.js

```
export const csr = false;
```

...adding `amp` to your `app.html`

```
<html amp>
...
```

...and transforming the HTML using `transformPageChunk` along with `transform` imported

```
import * as amp from '@sveltejs/amp';
import type { Handle } from '@sveltejs/kit';

export const handle: Handle = async ({ event, resolve }) => {
  let buffer = '';
  return await resolve(event, {
    transformPageChunk: ({ html, done }) => {
      buffer += html;
      if (done) return amp.transform(buffer);
    }
  });
};
```

To prevent shipping any unused CSS as a result of transforming the page to amp, we can use `dropcss` :

src/hooks.server.ts JS TS

```
import * as amp from '@sveltejs/amp';
import dropcss from 'dropcss';
import type { Handle } from '@sveltejs/kit';

export const handle: Handle = async ({ event, resolve }) => {
  let buffer = '';

  return await resolve(event, {
    transformPageChunk: ({ html, done }) => {
      buffer += html;

      if (done) {
        let css = '';
        const markup = amp
          .transform(buffer)
          .replace('⚡', 'amp') // dropcss can't handle this character
          .replace(/<style amp-custom([^>]*?)>([^]+?)<\/style>/, (match, attributes, cont
            css = contents;
            return `<style amp-custom${attributes}></style>`;
          });

        css = dropcss({ css, html: markup }).css;
        return markup.replace('</style>', `${css}</style>`);
```

It's a good idea to use the `handle` hook to validate the transformed HTML using `amphtml-validator`, but only if you're prerendering pages since it's very slow.

Edit this page on GitHub

PREVIOUS
Accessibility

NEXT
Frequently asked questions