SVELTEKIT • BEST PRACTICES

Accessibility

ON THIS PAGE

SvelteKit strives to provide an accessible platform for your app by default. Svelte's compile-time accessibility checks will also apply to any SvelteKit application you build.

Here's how SvelteKit's built-in accessibility features work and what you need to do to help these features to work as well as possible. Keep in mind that while SvelteKit provides an accessible foundation, you are still responsible for making sure your application code is accessible. If you're new to accessibility, see the "further reading" section of this guide for additional resources.

We recognize that accessibility can be hard to get right. If you want to suggest improvements to how SvelteKit handles accessibility, please open a GitHub issue.

Route announcements

In traditional server-rendered applications, every navigation (e.g. clicking on an `<a>` tag) triggers a full page reload. When this happens, screen readers and other assistive technology will read out the new page's title so that users understand that the page has changed.

Since navigation between pages in SvelteKit happens without reloading the page (known as client-side routing), SvelteKit injects a live region onto the page that will read out the new page name after each navigation. This determines the page name to announce by inspecting the `<title>` element.

Because of this behavior, every page in your app should have a unique, descriptive title. In SvelteKit, you can do this by placing a `<svelte:head>` element on each page:

Docs

```
  <title>Todo List</title>
</svelte:head>
```

This will allow screen readers and other assistive technology to identify the new page after a navigation occurs. Providing a descriptive title is also important for SEO.

Focus management

In traditional server-rendered applications, every navigation will reset focus to the top of the page. This ensures that people browsing the web with a keyboard or screen reader will start interacting with the page from the beginning.

To simulate this behavior during client-side routing, SvelteKit focuses the `<body>` element after each navigation and enhanced form submission. There is one exception - if an element with the `autofocus` attribute is present, SvelteKit will focus that element instead. Make sure to consider the implications for assistive technology when using that attribute.

If you want to customize SvelteKit's focus management, you can use the `afterNavigate` hook:

```
import { afterNavigate } from '$app/navigation';

afterNavigate(() => {
  /** @type {HTMLElement | null} */
  const to_focus = document.querySelector('.focus-me');
  to_focus?.focus();
});
```

You can also programmatically navigate to a different page using the `goto` function. By default, this will have the same client-side routing behavior as clicking on a link. However, `goto` also accepts a `keepFocus` option that will preserve the currently-focused element instead of resetting focus. If you enable this option, make sure the currently-focused element still exists on the page after navigation. If the element no longer exists, the user's

By default, SvelteKit's page template sets the default language of the document to English. If your content is not in English, you should update the `<html>` element in `src/app.html` to have the correct `lang` attribute. This will ensure that any assistive technology reading the document uses the correct pronunciation. For example, if your content is in German, you should update `app.html` to the following:

src/app.html

```
<html lang="de">
```

If your content is available in multiple languages, you should set the `lang` attribute based on the language of the current page. You can do this with SvelteKit's handle hook:

src/app.html

```
<html lang="%lang%">
```

src/hooks.server.ts JS TS

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

export const handle: Handle = ({ event, resolve }) => {
  return resolve(event, {
    transformPageChunk: ({ html }) => html.replace('%lang%', get_lang(event))
  });
};
```

Further reading

For the most part, building an accessible SvelteKit app is the same as building an accessible web app. You should be able to apply information from the following general accessibility resources to any web experience you build:

MDN Web Docs: Accessibility

Docs

Edit this page on GitHub

PREVIOUS
Images

NEXT
SEO