SVELTEKIT • APPENDIX

Frequently asked questions

ON THIS PAGE

Other resources

Please see the Svelte FAQ and `vite-plugin-svelte` FAQ as well for the answers to questions deriving from those libraries.

What can I make with SvelteKit?

SvelteKit can be used to create most kinds of applications. Out of the box, SvelteKit supports many features including:

- Dynamic page content with load functions and API routes.
- SEO-friendly dynamic content with server-side rendering (SSR).
- User-friendly progressively-enhanced interactive pages with SSR and Form Actions.
- Static pages with prerendering.

SvelteKit can also be deployed to a wide spectrum of hosted architectures via adapters. In cases where SSR is used (or server-side logic is added without prerendering), those functions will be adapted to the target backend. Some examples include:

- Self-hosted dynamic web applications with a Node.js backend.
- Serverless web applications with backend loaders and APIs deployed as remote functions. See zero-config deployments for popular deployment options.

dynamic content. SPAs are shipped without a backend and are not server-rendered. This option is commonly chosen when bundling SvelteKit with an app written in PHP, .Net, Java, C, Golang, Rust, etc.

A mix of the above; some routes can be static, and some routes can use backend functions to fetch dynamic information. This can be configured with page options that includes the option to opt out of SSR.

In order to support SSR, a JS backend — such as Node.js or Deno-based server, serverless function, or edge function — is required.

It is also possible to write custom adapters or leverage community adapters to deploy SvelteKit to more platforms such as specialized server environments, browser extensions, or native applications. See integrations for more examples and integrations.

How do I use HMR with SvelteKit?

SvelteKit has HMR enabled by default powered by svelte-hmr. If you saw Rich's presentation at the 2020 Svelte Summit, you may have seen a more powerful-looking version of HMR presented. This demo had `svelte-hmr`'s `preserveLocalState` flag on. This flag is now off by default because it may lead to unexpected behaviour and edge cases. But don't worry, you are still getting HMR with SvelteKit! If you'd like to preserve local state you can use the `@hmr:keep` or `@hmr:keep-all` directives as documented on the svelte-hmr page.

How do I include details from package.json in my application?

You cannot directly require JSON files, since SvelteKit expects `svelte.config.js` to be an ES module. If you'd like to include your application's version number or other information from `package.json` in your application, you can load JSON like so:

```
import { fileURLToPath } from 'node:url';

const path = fileURLToPath(new URL('package.json', import.meta.url));
const pkg = JSON.parse(readFileSync(path, 'utf8'));
```

How do I fix the error I'm getting trying to include a package?

Most issues related to including a library are due to incorrect packaging. You can check if a library's packaging is compatible with Node.js by entering it into the publint website.

Here are a few things to keep in mind when checking if a library is packaged correctly:

- `exports` takes precedence over the other entry point fields such as `main` and `module`. Adding an `exports` field may not be backwards-compatible as it prevents deep imports.
- ESM files should end with `.mjs` unless `"type": "module"` is set in which any case CommonJS files should end with `.cjs`.
- `main` should be defined if `exports` is not. It should be either a CommonJS or ESM file and adhere to the previous bullet. If a `module` field is defined, it should refer to an ESM file.
- Svelte components should be distributed as uncompiled `.svelte` files with any JS in the package written as ESM only. Custom script and style languages, like TypeScript and SCSS, should be preprocessed as vanilla JS and CSS respectively. We recommend using `svelte-package` for packaging Svelte libraries, which will do this for you.

Libraries work best in the browser with Vite when they distribute an ESM version, especially if they are dependencies of a Svelte component library. You may wish to suggest to library authors that they provide an ESM version. However, CommonJS (CJS) dependencies should work as well since, by default, `vite-plugin-svelte` will ask Vite to pre-bundle them using `esbuild` to convert them to ESM.

If you are still encountering issues we recommend searching both the Vite issue tracker and

short-term workaround in favor of fixing the library in question.

How do I use the view transitions API with SvelteKit?

While SvelteKit does not have any specific integration with view transitions, you can call `document.startViewTransition` in `onNavigate` to trigger a view transition on every client-side navigation.

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

onNavigate((navigation) => {
  if (!document.startViewTransition) return;

  return new Promise((resolve) => {
    document.startViewTransition(async () => {
      resolve();
      await navigation.complete;
    });
  });
});
```

For more, see "Unlocking view transitions" on the Svelte blog.

How do I use X with SvelteKit?

Make sure you've read the documentation section on integrations. If you're still having trouble, solutions to common issues are listed below.

How do I setup a database?

Put the code to query your database in a server route - don't query the database in .svelte files. You can create a `db.js` or similar that sets up a connection immediately and makes

them.

How do I use a client-side only library that depends on document or window?

If you need access to the `document` or `window` variables or otherwise need code to run only on the client-side you can wrap it in a `browser` check:

```
import { browser } from '$app/environment';

if (browser) {
  // client-only code here
}
```

You can also run code in `onMount` if you'd like to run it after the component has been first rendered to the DOM:

```
import { onMount } from 'svelte';

onMount(async () => {
  const { method } = await import('some-browser-only-library');
  method('hello world');
});
```

If the library you'd like to use is side-effect free you can also statically import it and it will be tree-shaken out in the server-side build where `onMount` will be automatically replaced with a no-op:

```
import { onMount } from 'svelte';
import { method } from 'some-browser-only-library';

onMount(() => {
  method('hello world');
});
```

```
<script>
  import { browser } from '$app/environment';

  const ComponentConstructor = browser ?
    import('some-browser-only-library').then((module) => module.Component) :
    new Promise(() => {});
</script>

{#await ComponentConstructor}
  <p>Loading...</p>
{:then component}
  <svelte:component this={component} />
{:catch error}
  <p>Something went wrong: {error.message}</p>
{/await}
```

How do I use a different backend API server?

You can use `event.fetch` to request data from an external API server, but be aware that you would need to deal with CORS, which will result in complications such as generally requiring requests to be preflighted resulting in higher latency. Requests to a separate subdomain may also increase latency due to an additional DNS lookup, TLS setup, etc. If you wish to use this method, you may find `handleFetch` helpful.

Another approach is to set up a proxy to bypass CORS headaches. In production, you would rewrite a path like `/api` to the API server; for local development, use Vite's `server.proxy` option.

How to setup rewrites in production will depend on your deployment platform. If rewrites aren't an option, you could alternatively add an API route:

src/routes/api/[...path]/+server.ts JS TS

```
import type { RequestHandler } from './$types';

export const GET: RequestHandler = ({ params, url }) => {
  return fetch(`https://my-api-server.com/${params.path + url.search}`);
};
```

`request.headers`, depending on your needs.)

How do I use middleware?

`adapter-node` builds a middleware that you can use with your own server for production mode. In dev, you can add middleware to Vite by using a Vite plugin. For example:

```
import { sveltekit } from '@sveltejs/kit/vite';

/** @type {import('vite').Plugin} */
const myPlugin = {
  name: 'log-request-middleware',
  configureServer(server) {
    server.middlewares.use((req, res, next) => {
      console.log(`Got request ${req.url}`);
      next();
    });
  }
};

/** @type {import('vite').UserConfig} */
const config = {
  plugins: [myPlugin, sveltekit()]
};

export default config;
```

See Vite's `configureServer` docs for more details including how to control ordering.

Does it work with Yarn 2?

Sort of. The Plug'n'Play feature, aka 'pnp', is broken (it deviates from the Node module resolution algorithm, and doesn't yet work with native JavaScript modules which SvelteKit — along with an increasing number of packages — uses). You can use `nodeLinker: 'node-modules'` in your `.yarnrc.yml` file to disable pnp, but it's probably easier to just use npm or pnpm, which is similarly fast and efficient but without the compatibility headaches.

Currently ESM Support within the latest Yarn (version 3) is considered experimental.

The below seems to work although your results may vary.

First create a new application:

```
yarn create svelte myapp
cd myapp
```

And enable Yarn Berry:

```
yarn set version berry
yarn install
```

Yarn 3 global cache

One of the more interesting features of Yarn Berry is the ability to have a single global cache for packages, instead of having multiple copies for each project on the disk. However, setting `enableGlobalCache` to true causes building to fail, so it is recommended to add the following to the `.yarnrc.yml` file:

```
nodeLinker: node-modules
```

This will cause packages to be downloaded into a local node_modules directory but avoids the above problem and is your best bet for using version 3 of Yarn at this point in time.

Edit this page on GitHub

PREVIOUS
SEO

NEXT
Integrations