SVELTEKIT • ADVANCED

Errors

ON THIS PAGE

Errors are an inevitable fact of software development. SvelteKit handles errors differently depending on where they occur, what kind of errors they are, and the nature of the incoming request.

Error objects

SvelteKit distinguishes between expected and unexpected errors, both of which are represented as simple `{ message: string }` objects by default.

You can add additional properties, like a `code` or a tracking `id`, as shown in the examples below. (When using TypeScript this requires you to redefine the `Error` type as described in type safety).

Expected errors

An *expected* error is one created with the `error` helper imported from `@sveltejs/kit`:

src/routes/blog/[slug]/+page.server.ts

JS TS

```
import { error } from '@sveltejs/kit';
import * as db from '$lib/server/database';
import type { PageServerLoad } from './$types';

export const load: PageServerLoad = async ({ params }) => {
  const post = await db.getPost(params.slug);
```

```
    });
  }

  return { post };
};
```

This throws an exception that SvelteKit catches, causing it to set the response status code to 404 and render an `+error.svelte` component, where `$page.error` is the object provided as the second argument to `error(...)`.

src/routes/+error.svelte

```
<script>
  import { page } from '$app/stores';
</script>

<h1>{$page.error.message}</h1>
```

You can add extra properties to the error object if needed…

```
error(404, {
  message: 'Not found',
  code: 'NOT_FOUND'
});
```

…otherwise, for convenience, you can pass a string as the second argument:

```
error(404, { message: 'Not found' });
error(404, 'Not found');
```

In SvelteKit 1.x you had to `throw` the `error` yourself

Unexpected errors

Docs

can contain sensitive information, unexpected error messages and stack traces are not exposed to users.

By default, unexpected errors are printed to the console (or, in production, your server logs), while the error that is exposed to the user has a generic shape:

```
{ "message": "Internal Error" }
```

Unexpected errors will go through the `handleError` hook, where you can add your own error handling — for example, sending errors to a reporting service, or returning a custom error object which becomes `$page.error` .

Responses

If an error occurs inside `handle` or inside a `+server.js` request handler, SvelteKit will respond with either a fallback error page or a JSON representation of the error object, depending on the request's `Accept` headers.

You can customise the fallback error page by adding a `src/error.html` file:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>%sveltekit.error.message%</title>
  </head>
  <body>
    <h1>My custom error page</h1>
    <p>Status: %sveltekit.status%</p>
    <p>Message: %sveltekit.error.message%</p>
  </body>
</html>
```

SvelteKit will replace `%sveltekit.status%` and `%sveltekit.error.message%` with their render the `+error.svelte` component nearest to where the error occurred. If the error occurs inside a `load` function in `+layout(.server).js`, the closest error boundary in the tree is an `+error.svelte` file *above* that layout (not next to it).

The exception is when the error occurs inside the root `+layout.js` or `+layout.server.js`, since the root layout would ordinarily *contain* the `+error.svelte` component. In this case, SvelteKit uses the fallback error page.

Type safety

If you're using TypeScript and need to customize the shape of errors, you can do so by declaring an `App.Error` interface in your app (by convention, in `src/app.d.ts`, though it can live anywhere that TypeScript can 'see'):

src/app.d

```
declare global {
  namespace App {
    interface Error {
      code: string;
      id: string;
    }
  }
}

export {};
```

This interface always includes a `message: string` property.

Further reading

- Tutorial: Errors and redirects
- Tutorial: Hooks

Hooks

Link options