Web standards

Throughout this documentation, you'll see references to the standard Web APIs that SvelteKit builds on top of. Rather than reinventing the wheel, we *use the platform*, which means your existing web development skills are applicable to SvelteKit. Conversely, time spent learning SvelteKit will help you be a better web developer elsewhere.

These APIs are available in all modern browsers and in many non-browser environments like Cloudflare Workers, Deno, and Vercel Functions. During development, and in adapters for Node-based environments (including AWS Lambda), they're made available via polyfills where necessary (for now, that is — Node is rapidly adding support for more web standards).

In particular, you'll get comfortable with the following:

Fetch APIs

SvelteKit uses `fetch` for getting data from the network. It's available in hooks and server routes as well as in the browser.

> A special version of `fetch` is available in `load` functions, server hooks and API routes for invoking endpoints directly during server-side rendering, without making an HTTP call, while preserving credentials. (To make credentialled fetches in server-side code outside `load`, you must explicitly pass `cookie` and/or `authorization` headers.) It also allows you to make relative requests, whereas server-side `fetch` normally requires a fully qualified URL.

An instance of `Request` is accessible in hooks and server routes as `event.request`. It contains useful methods like `request.json()` and `request.formData()` for getting data that was posted to an endpoint.

Response

An instance of `Response` is returned from `await fetch(...)` and handlers in `+server.js` files. Fundamentally, a SvelteKit app is a machine for turning a `Request` into a `Response`.

Headers

The `Headers` interface allows you to read incoming `request.headers` and set outgoing `response.headers`. For example, you can get the `request.headers` as shown below, and use the `json` convenience function to send modified `response.headers`:

src/routes/what-is-my-user-agent/+server.ts

JS TS

```
import { json } from '@sveltejs/kit';
import type { RequestHandler } from './$types';

export const GET: RequestHandler = ({ request }) => {
  // log all headers
  console.log(...request.headers);

  // create a JSON Response using a header we received
  return json({
    // retrieve a specific header
    userAgent: request.headers.get('user-agent')
  }, {
    // set a header on the response
    headers: { 'x-custom-header': 'potato' }
  });
};
```

FormData

objects.

src/routes/hello/+server.ts

JS TS

```
import { json } from '@sveltejs/kit';
import type { RequestHandler } from './$types';

export const POST: RequestHandler = async (event) => {
  const body = await event.request.formData();

  // log all fields
  console.log([...body]);

  return json({
    // get a specific field's value
    name: body.get('name') ?? 'world'
  });
};
```

Stream APIs

Most of the time, your endpoints will return complete data, as in the `userAgent` example above. Sometimes, you may need to return a response that's too large to fit in memory in one go, or is delivered in chunks, and for this the platform provides streams — ReadableStream, WritableStream and TransformStream.

URL APIs

URLs are represented by the `URL` interface, which includes useful properties like `origin` and `pathname` (and, in the browser, `hash`). This interface shows up in various places — `event.url` in hooks and server routes, `$page.url` in pages, `from` and `to` in `beforeNavigate` and `afterNavigate` and so on.

which is an instance of `URLSearchParams` :

```
const foo = url.searchParams.get('foo');
```

Web Crypto

The Web Crypto API is made available via the `crypto` global. It’s used internally for Content Security Policy headers, but you can also use it for things like generating UUIDs:

```
const uuid = crypto.randomUUID();
```

Edit this page on GitHub