

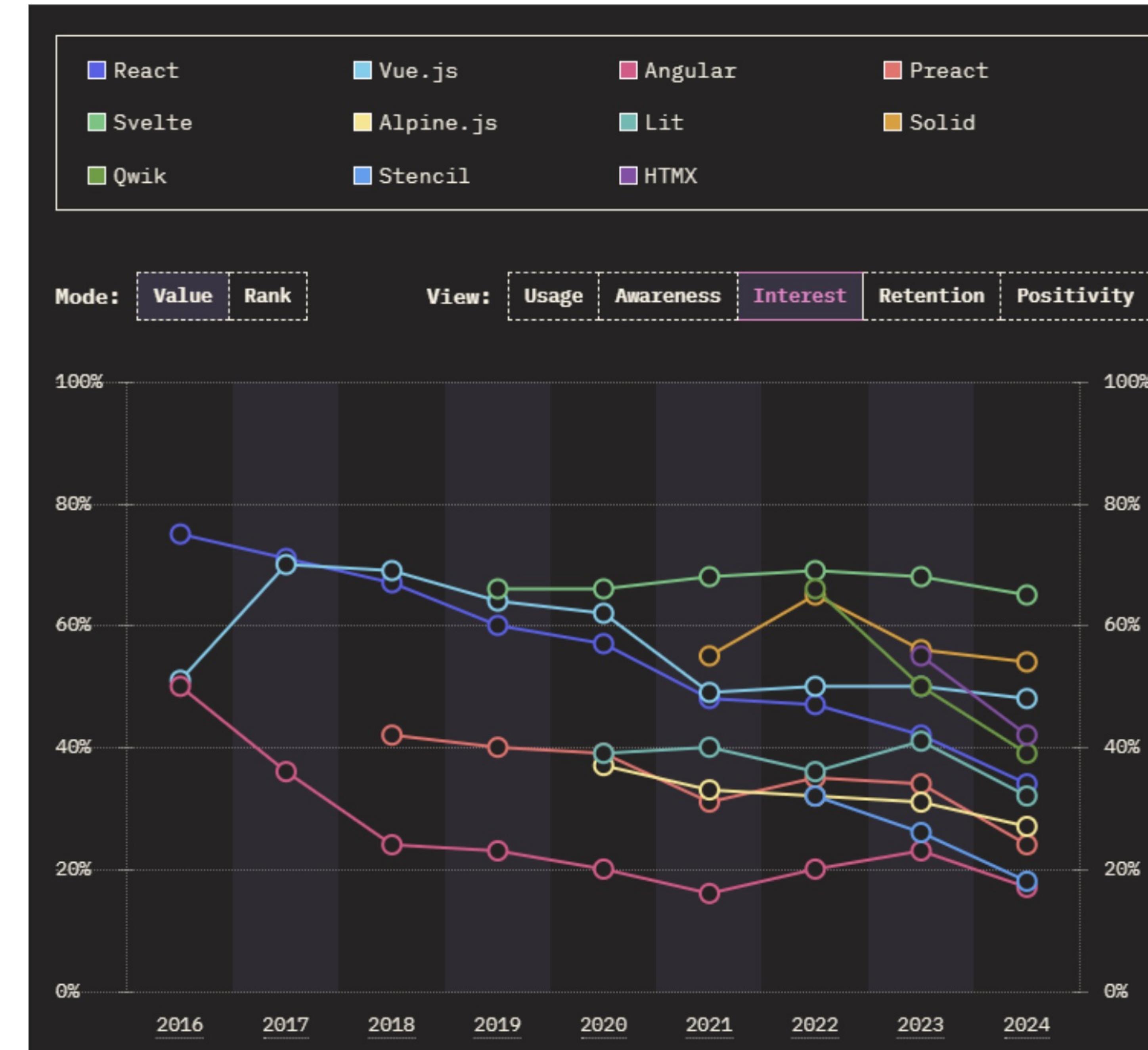


Intro to Sveltekit

A web framework

Overview:

0. Scaffolding an app with `sv`
1. Recommended Editor plugins
2. Core concepts
3. Loading data
4. Handing user data / forms
5. Composability
6. Lightning round
7. Adapters and deployment



State of JS 2024

Who am I?

Sam Peterson

🇺🇸 Minnesota, USA

Living in 🇩🇰 for 3 years

Full Stack since 2018

Passionate about:

- traveling
- programming
- learning



Scaffolding an app

```
npx sv create
```

```
Welcome to the Svelte CLI! (v0.8.0)

▷ Which template would you like?
  SvelteKit minimal

▷ Add type checking with TypeScript?
  Yes, using TypeScript syntax

◆ Project created

▷ What would you like to add to your project? (use arrow keys / space bar)
  prettier, eslint, tailwindcss

▷ tailwindcss: Which plugins would you like to add?
  typography, forms

◆ Successfully setup add-ons

▷ Which package manager do you want to install dependencies with?
  deno
```



Editor plugin



Svelte for VS Code

Svelte svelte.dev

| 1,979,920 installs | ★★★★★ (44) | Free

Svelte language support for VS Code

Install

Trouble Installing?

- Emmet support
- Code actions
- Go to definitions
- Autocompletions
- Helpers -- Extract Component
- more...

A screenshot of the VS Code interface. On the left, there's a terminal window showing a list of Svelte snippets: `await :then`, `await then`, `each`, `if`, `key`, and `snippet`. The main editor area shows some code with a yellow star icon in the top right corner. The status bar at the bottom indicates "You, 22 hours ago • Uncommitted changes".

A screenshot of the VS Code Command Palette. The search bar contains the text ">svelte". Below it, a list of commands is displayed, including:

- SvelteKit: Create +page.svelte
- SvelteKit: Create +page.server.js/ts
- SvelteKit: Create +layout.svelte
- SvelteKit: Create +error.svelte
- Svelte: Restart Language Server
- SvelteKit: Create route
- Profiles: Open Sveltekit Profile
- Svelte: Extract Component
- Svelte: Migrate Component to Svelte 5 Syntax
- SvelteKit: Create +layout.js/ts
- SvelteKit: Create +layout.server.js/ts
- SvelteKit: Create +page.js/ts
- SvelteKit: Create +server.js/ts

The "recently used" and "other commands" sections are also visible.

What makes it different?

Svelte is a superset
of HTML, not of
Javascript

Svelte 5

Introduction of runes

- \$state
- \$derived
- \$effect
- \$props
- \$bindable
- \$inspect
- \$host

Template syntax

- {#if...}...{/if}
- {#each ...}...{/each}
- {#await ...}
- {#snippet ...}
- {@render ...}
- {@html ...}
- bind:
- use:
- transition:

And more...

Loading patterns can be simple.



Loading data



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

JS TS



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

export const load: PageLoad = ({ params }) => {
  return {
    post: {
      title: `Title for ${params.slug} goes here`,
      content: `Content for ${params.slug} goes here`
    }
  };
};

<script lang="ts">
  import type { PageProps } from './$types';

  let { data }: PageProps = $props();
</script>

<h1>{data.post.title}</h1>
<div>{@html data.post.content}</div>
```

JS TS



Loading data

Remix

app/routes/products.tsx

```
1 import { json } from "@remix-run/node"; // or cloudflare/deno
2 import { useLoaderData } from "@remix-run/react";
3
4 export const loader = async () => {
5   return json([
6     { id: "1", name: "Pants" },
7     { id: "2", name: "Jacket" },
8   ]);
9 }
10
11 export default function Products() {
12   const products = useLoaderData<typeof loader>();
13   return (
14     <div>
15       <h1>Products</h1>
16       {products.map((product) => (
17         <div key={product.id}>{product.name}</div>
18       ))}
19     </div>
20   );
21 }
```

app/routes/users.\$userId.projects.\$projectId.tsx

```
1 import type { LoaderFunctionArgs } from "@remix-run/node";
2
3 export const loader = async ({
4   params,
5 }: LoaderFunctionArgs) => {
6   console.log(params.userId);
7   console.log(params.projectId);
8 }
```

User data

The `<form>` [HTML](#) element represents a document section containing interactive controls for submitting information. Source [MDN](#)

src/routes/login/+page.svelte

```
<form method="POST">
  <label>
    Email
    <input name="email" type="email">
  </label>
  <label>
    Password
    <input name="password" type="password">
  </label>
  <button>Log in</button>
</form>
```

src/routes/login/+page.server.ts

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

export const actions = {
  default: async (event) => {
    // TODO log the user in
  }
} satisfies Actions;
```

User data

Server

```
project
  src
    lib
    server
      database.js
    routes
      +page.server.js
      +page.svelte
      remove.svg
    app.html
  static
    shared.css
package.json
svelte.config.js
vite.config.js
```

```
4   let id = cookies.get('userid');
5
6   if (!id) {
7     id = crypto.randomUUID();
8     cookies.set('userid', id, { path: '/' });
9   }
10
11  return {
12    todos: db.getTodos(id)
13  };
14}
15
16  export const actions = {
17    create: async ({ cookies, request }) => {
18      const data = await request.formData();
19      db.createTodo(cookies.get('userid'), data.get('description'));
20    },
21
22    delete: async ({ cookies, request }) => {
23      const data = await request.formData();
24      db.deleteTodo(cookies.get('userid'), data.get('id'));
25    }
26  };
27
```

Client

```
8   <form method="POST" action="/create">
9     <label>
10       add a todo:
11       <input
12         name="description"
13         autocomplete="off"
14       />
15     </label>
16   </form>
17
18   <ul class="todos">
19     {#each data.todos as todo (todo.id)}
20     <li>
21       <form method="POST" action="/delete">
22         <input type="hidden" name="id" value={todo.id} />
23         <span>{todo.description}</span>
24         <button aria-label="Mark as complete"></button>
25       </form>
26     </li>
27   {/each}
28 </ul>
```

Composability

Components:

```
□ App.svelte* □ Adjective.svelte* × ⌂ RUNES 🔒 ▾
```

```
1 <script lang="ts">
2   let props = $props<{adjective: string}>();
3 </script>
4
5 <p>this component is {props.adjective}</p>
```

```
□ App.svelte* □ Adjective.svelte* ⌂ RUNES 🔒 ▾
```

```
1 <script>
2   import Adjective from './Adjective.svelte'
3   let name = 'world';
4 </script>
5
6 <h1>Hello {name}!</h1>
7
8 <Adjective adjective="Cool" />
9 <Adjective adjective="Dry" />
10 <Adjective adjective="Simple" />
11 |
```

The screenshot shows the Svelte REPL interface. On the left, the `App.svelte*` file is open, displaying code that imports the `Adjective` component and uses it three times with different `adjective` props: "Cool", "Dry", and "Simple". The `Adjective.svelte*` file is also listed in the tabs. On the right, the `Result` tab is selected, showing the rendered output: "Hello world!", followed by three lines of text: "this component is Cool", "this component is Dry", and "this component is Simple".

Result

Hello world!

this component is Cool

this component is Dry

this component is Simple

Composability

Snippets:

```
{#snippet figure(image)}  
  <figure>  
    <img src={image.src} alt={image.caption} width={image.width}>  
    <figcaption>{image.caption}</figcaption>  
  </figure>  
{/snippet}
```

```
{#each images as image}  
  {#if image.href}  
    <a href={image.href}>  
      {@render figure(image)}  
    </a>  
  {:else}  
    {@render figure(image)}  
  {/if}  
{/each}
```

App.svelte*

RUNES 🔒

Result JS output CSS output AS

```
1 <script>  
2   let { message = `it's great to see you!` } = $props();  
3 </script>  
4  
5 {#snippet hello(name)}  
6   <p>hello {name}! {message}!</p>  
7 {/snippet}  
8  
9 {@render hello('alice')}  
10 {@render hello('bob')}
```

hello alice! it's great to see you!!

hello bob! it's great to see you!!

Lightning Round: CSS scoped by default

```
□ App.svelte*
```

```
1 <script>
2   import Button from "./Button.svelte"
3 </script>
4
5 <style>
6   main {
7     font-family: sans-serif;
8     text-align: center;
9   }
10  button {
11    border: black;
12    background-color: rgba(
13      200, 200, 235, 25%
14    );
15    border: solid black 1px;
16    padding: 10px;
17  }
18 </style>
19
20 <main>
21   <h1>Hello Svelte</h1>
22   <button>Different</button>
23   <Button />
24 </main>
```

```
□ App.svelte* □ Button.svelte* × RUNES 🔒
1 <script>
2   let count = 0;
3
4   function handleClick() {
5     count += 1;
6   }
7 </script>
8
9 <button on:click={handleClick}>
10  Clicked {count} {count === 1 ? 'time' : 'times'}
11 </button>
12
13 <style>
14   button {
15     background: #ff3e00;
16     color: white;
17     border: none;
18     padding: 8px 12px;
19     border-radius: 2px;
20   }
21 </style>
```

Hello Svelte

Different

Clicked 0 times

Lightning Round: Class helpers

Attribute:

The screenshot shows a Svelte component named "App.svelte". The code uses attribute-based class helpers to apply styles based on item length. A green bracket highlights the class assignment in line 9.

```
1 <script>
2 let items = ['One', 'Two', 'Three'];
3 </script>
4
5 <ul>
6 {#each items as item}
7 <li key={item}
8   class={{
9     'font-bold': true,
10    'text-green-400': item.length <= 3,
11    'text-red-400': item.length > 3
12   }}
13 >
14   {item}
15 </li>
16 {/each}
17 </ul>
```

Directive:

The screenshot shows a Svelte component named "App.svelte". The code uses directive-based class helpers to apply styles based on item length. A green bracket highlights the class assignment in line 8.

```
1 <script>
2 let items = ['One', 'Two', 'Three'];
3 </script>
4
5 <ul>
6 {#each items as item}
7 <li key={item}
8   class='font-bold'
9   class:text-green-400={item.length <= 3}
10  class:text-red-400={item.length > 3}
11 >
12   {item}
13 </li>
14 {/each}
15 </ul>
```

Adapters

Allow SvelteKit apps to be deployed to a variety of locations.

```
svelte.config.js

import adapter from 'svelte-adapter-foo';

/** @type {import('@sveltejs/kit').Config} */
const config = {
    kit: {
        adapter: adapter({
            // adapter options go here
        })
    }
};

export default config;
```

Official adapters exist for a variety of platforms – these are documented on the following pages:

1. [@sveltejs/adapter-cloudflare](#) for Cloudflare Workers and Cloudflare Pages
2. [@sveltejs/adapter-netlify](#) for Netlify
3. [@sveltejs/adapter-node](#) for Node servers
4. [@sveltejs/adapter-static](#) for static site generation (SSG)
5. [@sveltejs/adapter-vercel](#) for Vercel

Additional [community-provided adapters](#) exist for other platforms.



Repository

Code demo and questions

Thanks for your attention



For making it to the end enjoy
a funny photo of my dog.