Svelte - Basics

Introduction

Create a svelte App

```
$> npm create svelte@latest myapp
```

Run a svelte App

```
$> npm run dev
```

Deploy a svelte App & SvelteKit

Deployments (Adapters)

Deploying a svelte app can be done easily using SvelteKit Adapters.

SvelteKit is a **framework for rapidly developing** robust, performant **web applications using Svelte**.
If you're coming from React, SvelteKit is similar to Next. If you're coming from Vue, SvelteKit is similar to Nuxt.

Bundler

SvelteKit uses Vite to build your code.

Basics

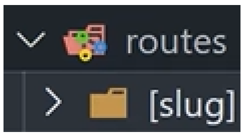
Routing

Project structure

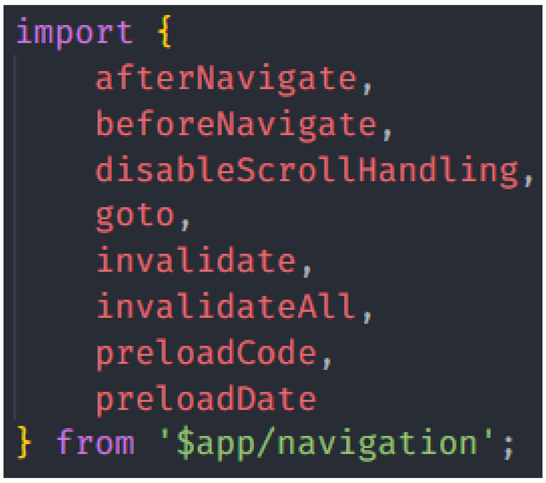
It's a file system base router.

```
src
├── routes
│   ├── /
│   ├── +page.svelte
│   ├── /about
│   └── about
│       ├── +page.svelte
│       ├── /about/contact
│       └── contact
│           └── +page.svelte
│
├── app.d.ts
└── app.html
```

Dynamic routes



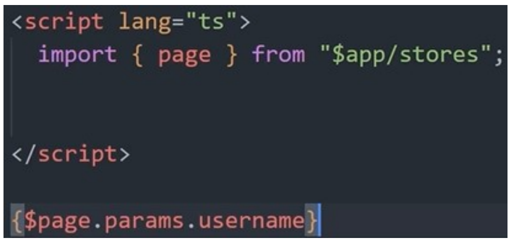
Navigation functions



```
import {
    afterNavigate,
    beforeNavigate,
    disableScrollHandling,
    goto,
    invalidate,
    invalidateAll,
    preloadCode,
    preloadDate
} from '$app/navigation';
```

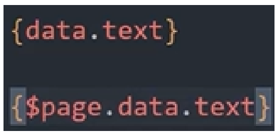
Page store

The page store **contains the full URL**, the route **parameters**, **and more** information from the router.



```
<script lang="ts">
  import { page } from "$app/stores";


</script>

{$page.params.username}
```

It also allows you to **retrieve** any **fetched data**.

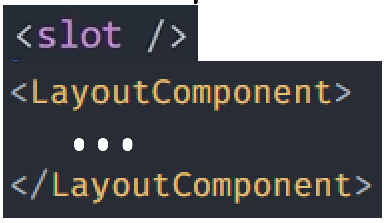


```
{data.text}

{$page.data.text}
```

Client & Server

Svelte file structure (.ts, .server.ts, .svelte)

+ is used to **mark files reserved by svelteKit**.

NAME	DESCRIPTION	CLIENT	SERVER
+page.svelte	**UI Component.**	X	
+page.ts	Can do both **data fetcing client and server side**.	X	X
+page.server.ts	Can do **data fetching server side**.		X
+layout.ts	Work like page.svelte, but the **UI can be shared across multiple child routes**.  `<slot />` `<LayoutComponent>` `...` `</LayoutComponent>`	X	X
+server.ts	Used to create **API routes** which returns data.		X
+error.svelte	Fallback for a page when **data-fetching fails**.		X

Rendering

Svelte implements **Client-side rendering**, **server side rendering, static site generation**** and **prerendering****.

** Check the nextjs tutorial*

*** Needs a sveltekit adapter*

**** **Prerendering** : Render pages on the server at build time.*

`export const prerender = true;`
If added, will enable prerendering for your page.

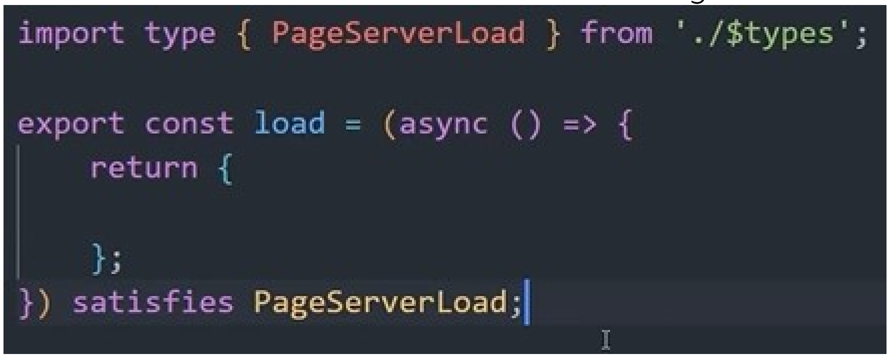
SSR is the default behavior !
`export const ssr = true;`

To disable CSR on a page:
`export const csr = false;`

Examples

+page.server.ts / server side

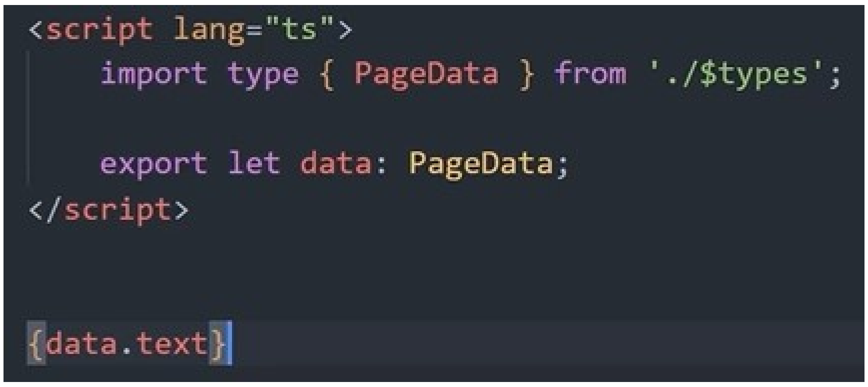
The load function executes when the user navigates to the UI component.



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

export const load = (async () => {
    return {

    };
}) satisfies PageServerLoad;
```

+page.svelte / client side

The data can be accessed from the client side.



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

    export let data: PageData;
</script>


{data.text}
```

+page.ts / client and server side

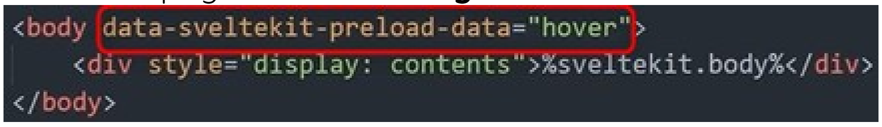
1. On the initial page load the code will load server side.
2. On a subsequent navigation the code will run client side.

Best for public data fetching.

Specificities

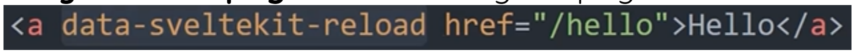
Page fetching

Fetches a page **when hovering** a link.



```
<body data-sveltekit-preload-data="hover">
    <div style="display: contents">%sveltekit.body%</div>
</body>
```

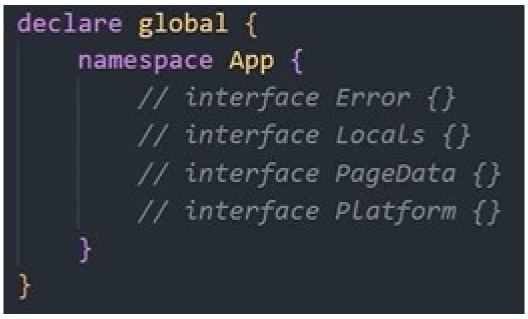
Fully reload a page when loading the page.



```
<a data-sveltekit-reload href="/hello">Hello</a>
```

Types & Variables

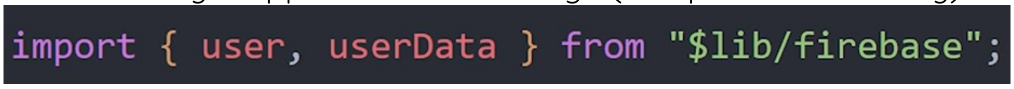
Create custom type definitions inside your app.d.ts file



```
declare global {
    namespace App {
        // interface Error {}
        // interface Locals {}
        // interface PageData {}
        // interface Platform {}
    }
}
```

Lib directory

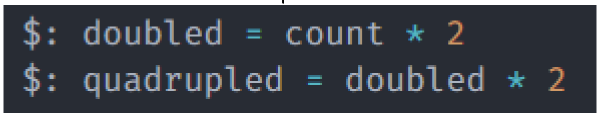
Automatically mapped with a dollar sign (components directory).



```
import { user, userData } from "$lib/firebase";
```

Reactive declerations

Useable with JS expressions .



```
$: doubled = count * 2
$: quadrupled = doubled * 2
```

Special elements & Directives

Element directives

`on:eventname|modifiers={handler}`

Dynamic components

<svelte:component this={currentSelectio.component} />

Window Events

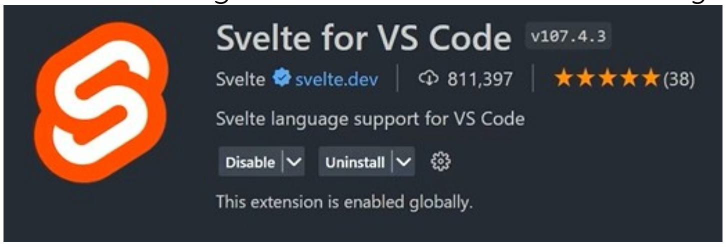
`<svelte:window on:event={handler} />`

Documents

`<svelte:document on:event={handler} />`

Plugins

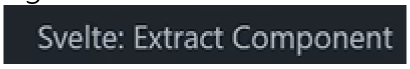
Auto generate boilerplate for page, layout, server and error files.
Works with `Right click on routes folder` inside your project.


Svelte for VS Code v107.4.3
Svelte svelte.dev | 811,397 | ★★★★★ (38)
Svelte language support for VS Code

This extension is enabled globally.

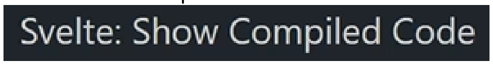
Extract code as component

Allows to extract code as component by highlighting the extract needed code and right click.



Show compiled code

Ctrl +shift + p



UI logic & Animations

Logic blocks

#Key

Svelte key will rebuild all children when the value on the right-side changes.

```
{#key $page.url}
    <Children />
{/key}
```

#if

```
{#if expression}...{/if}
{#if expression}...{:else if expression}...{/if}
{#if expression}...{:else}...{/if}
```

#each

```
{#each expression as name}...{/each}
{#each expression as name, index}...{/each}
{#each expression as name (key)}...{/each}
{#each expression as name, index (key)}...{/each}
{#each expression as name}...{:else}...{/each}
```

#await

```
{#await expression}...{:then name}...{:catch name}...{/await}
{#await expression}...{:then name}...{/await}
{#await expression then name}...{/await}
{#await expression catch name}...{/await}
```

Special tags

@html

Allow to unescape html tags.
`{@html expression}`

@debug

Console logs values.
`{@debug var1, var2, ..., varN}`

@const

Defines a local constant.
`{@const a = b * c}`

Animations

Svelte has 3 types of transitions, **in, out** and **transition**.

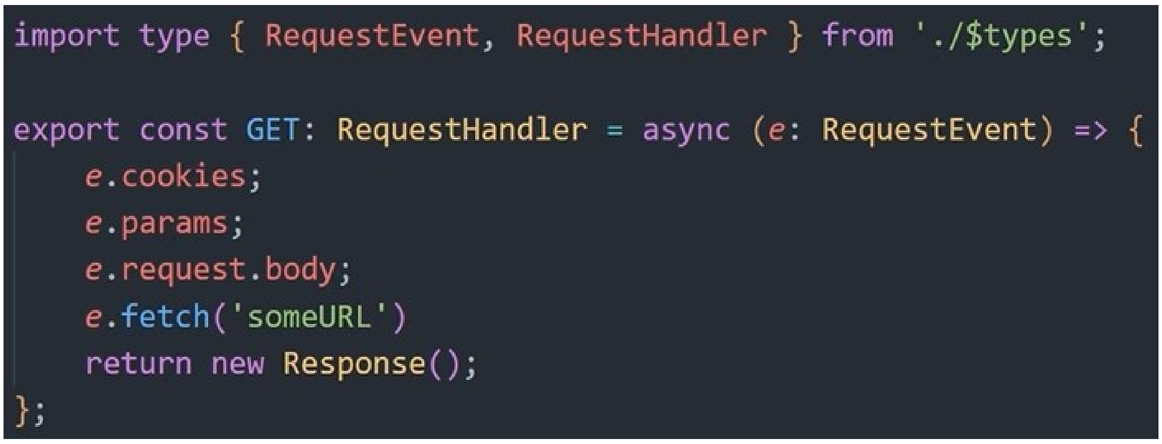
```
import {
    fade,
    blur,
    fly,
    slide,
    scale,
    draw,
    crossfade
} from 'svelte/transition';
```

Example:

```
<div transition:fade={{ delay: 250, duration: 300 }}>fades in and out</div>
```

Handling Requests

RequestsHandlers / RequestEvents



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

export const GET: RequestHandler = async (e: RequestEvent) => {
    e.cookies;
    e.params;
    e.request.body;
    e.fetch('someURL')
    return new Response();
};
```

The fetch function will inherit cookies and authorization headers from the client.

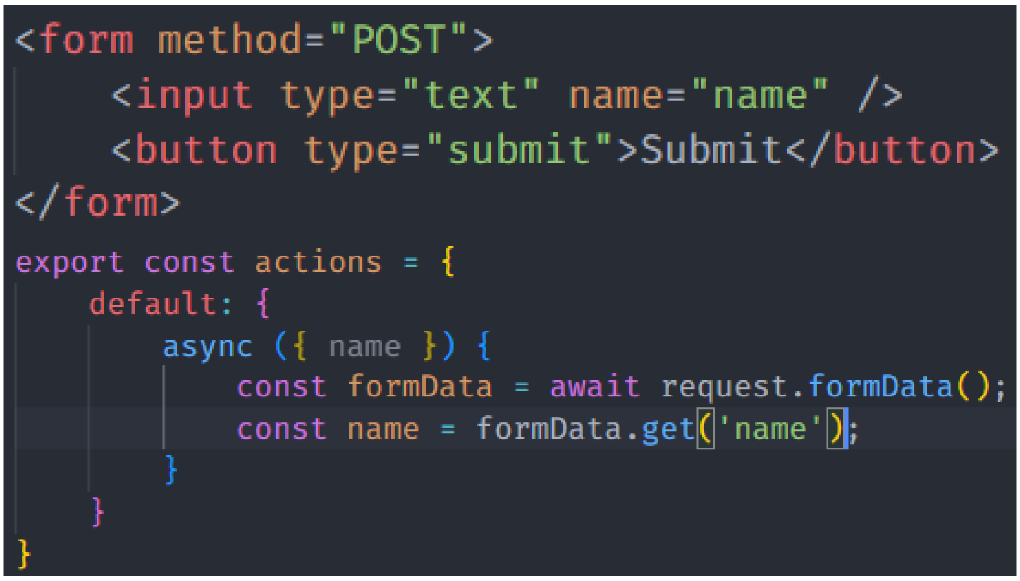
Svelte has some built-in functions like :

```
throw error(code, message)
return json(Object)
```

Actions

Default actions

Allows you to **make request**, **update the UI** and choose the **hydration** process easily.



```
<form method="POST">
    <input type="text" name="name" />
    <button type="submit">Submit</button>
</form>

export const actions = {
    default: {
        async ({ name }) {
            const formData = await request.formData();
            const name = formData.get('name');
        }
    }
}
```

Enhance

This performs a full page reload but with **enhance**, JavaScript **will** take over that form submission and **prevent a full page reload**.

```
import { enhance } from '$apps/forms';
<form method="POST" use:enhance>
```

Enhance also allows you to manage requests client side (access status, access form data, validation, …)

Named actions

```
export const actions = {
    rename: {
        async ({ name }) {
            const formData = await request.formData();
            const name = formData.get('name');
        }
    }
}
<form method="POST" action="?/rename" use:enhance>
```

Also works on form tags.

```
<button formaction="?/upperCaseName">Upper case</button>
```

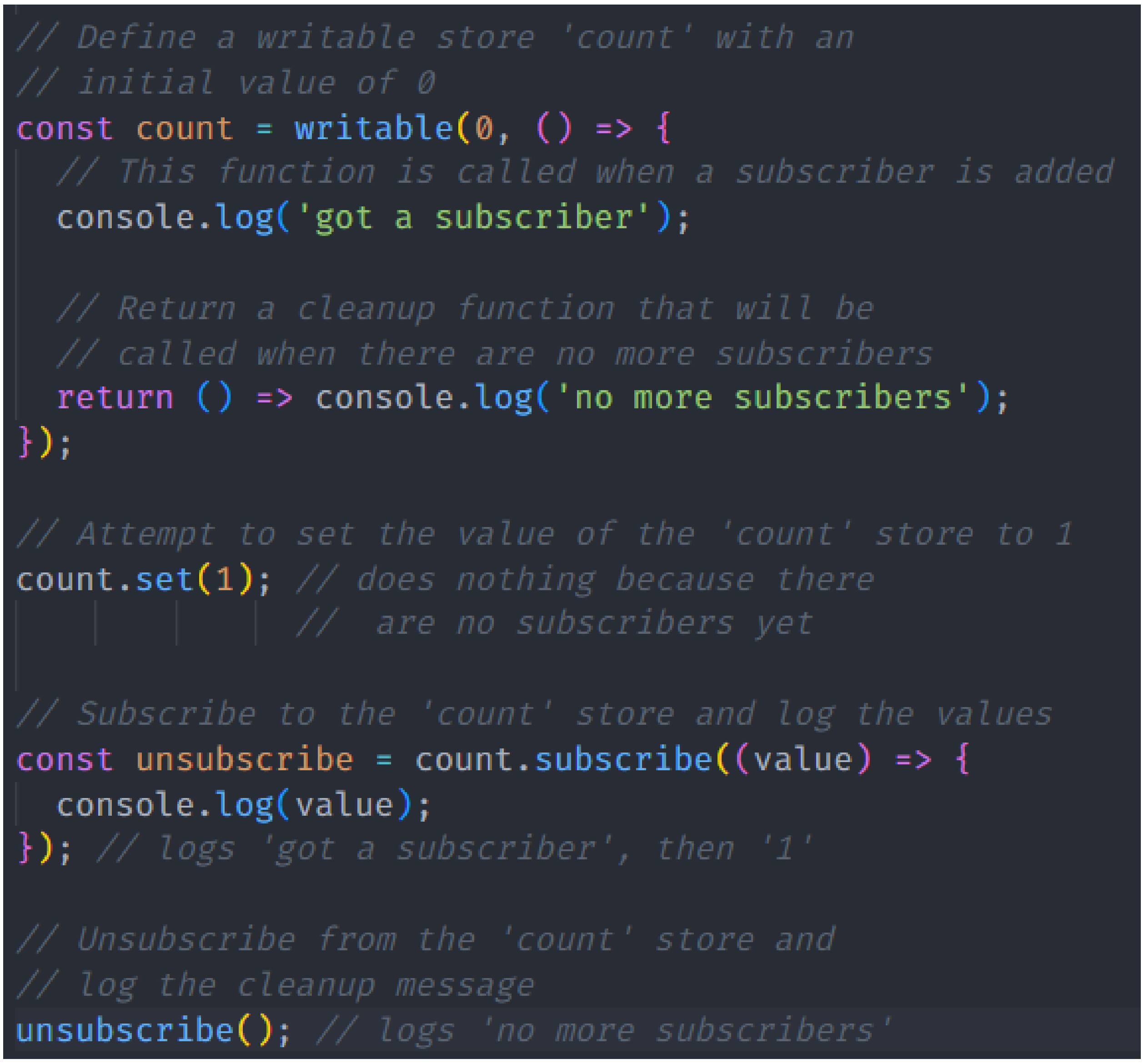
Cache control header

Can be set to **limit number of request or to address certain behaviors**.

```
(async ({ setHeaders }) =>
setHeaders({
    "cache-control": "max-age=60",
});
setHeaders({
    age: res.headers.get('age'),
    'cache-control': res.headers.get('cache-control')
});
```

Stores

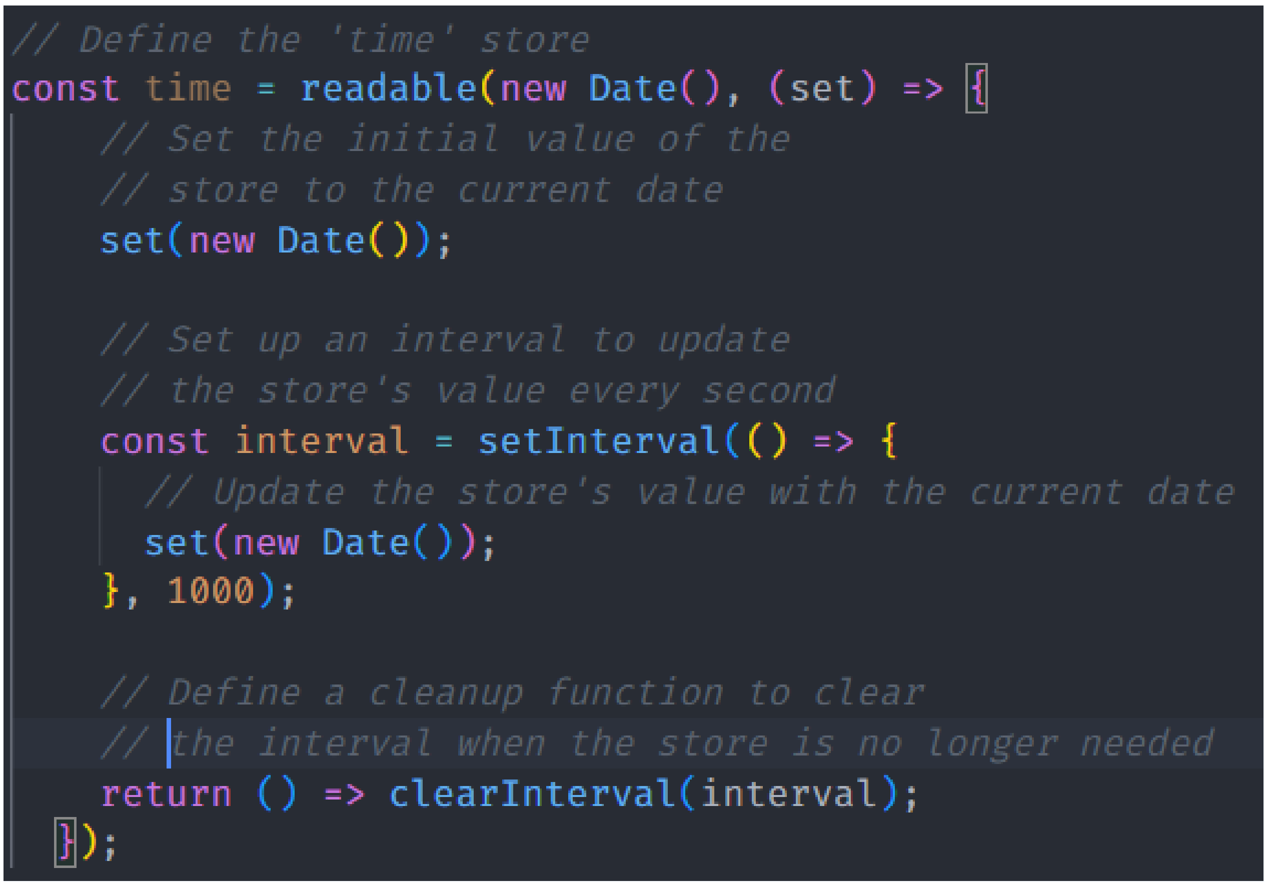
Writable



```
// Define a writable store 'count' with an
// initial value of 0
const count = writable(0, () => {
  // This function is called when a subscriber is added
  console.log('got a subscriber');

  // Return a cleanup function that will be
  // called when there are no more subscribers
  return () => console.log('no more subscribers');
});

// Attempt to set the value of the 'count' store to 1
count.set(1); // does nothing because there
              //  are no subscribers yet

// Subscribe to the 'count' store and log the values
const unsubscribe = count.subscribe((value) => {
  console.log(value);
}); // logs 'got a subscriber', then '1'

// Unsubscribe from the 'count' store and
// log the cleanup message
unsubscribe(); // logs 'no more subscribers'
```

Writable object methods are set and update.
Writable values can be set from outside components.

Readable

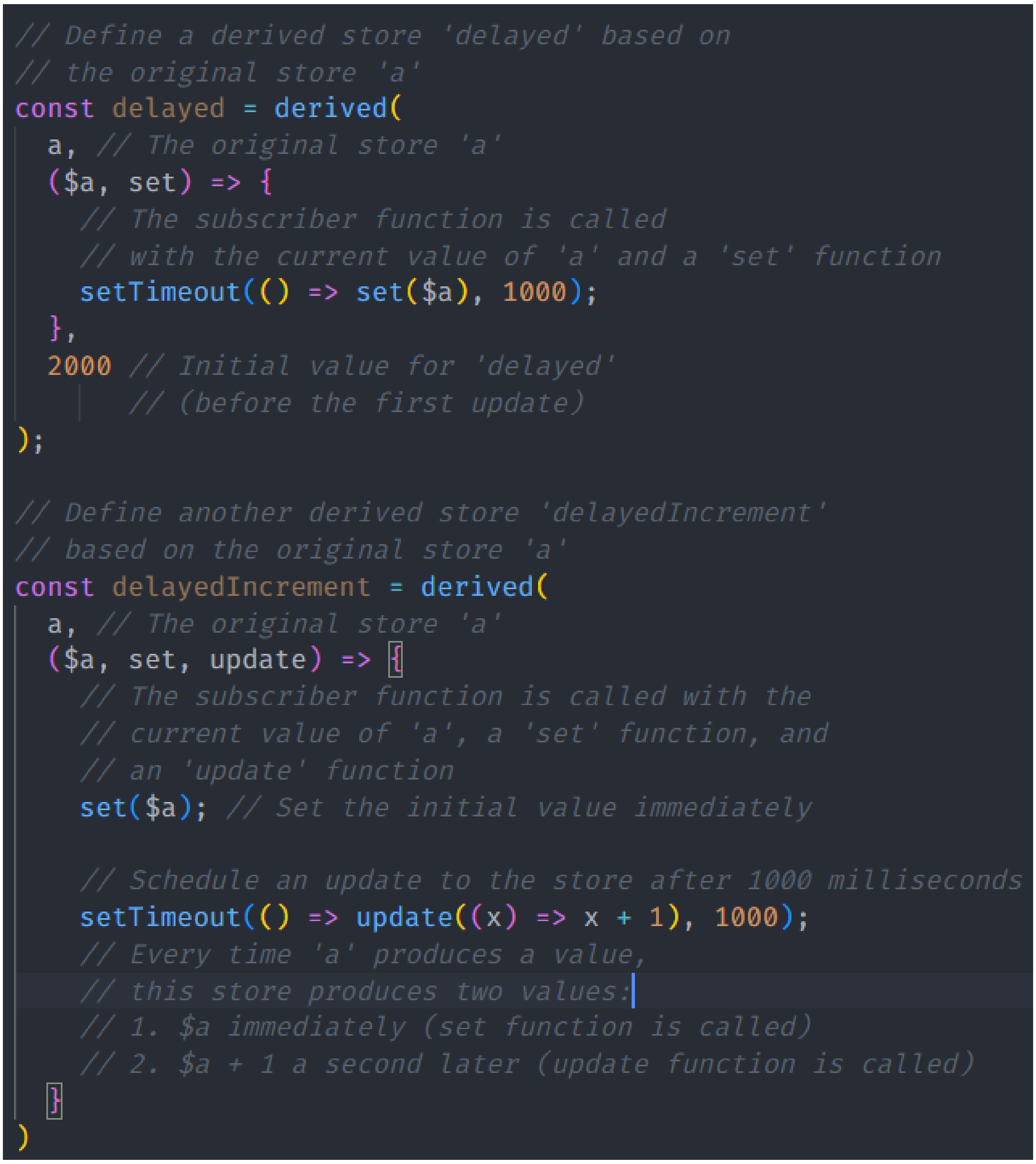


```
// Define the 'time' store
const time = readable(new Date(), (set) => {
    // Set the initial value of the
    // store to the current date
    set(new Date());

    // Set up an interval to update
    // the store's value every second
    const interval = setInterval(() => {
      // Update the store's value with the current date
      set(new Date());
    }, 1000);

    // Define a cleanup function to clear
    // the interval when the store is no longer needed
    return () => clearInterval(interval);
  });
```

Creates a store whose value cannot be set from 'outside'.

Derived stores

Derived stores take multiples stores and combines them in a single value.
Derives a store from one or more other stores.



```
// Define a derived store 'delayed' based on
// the original store 'a'
const delayed = derived(
  a, // The original store 'a'
  ($a, set) => {
    // The subscriber function is called
    // with the current value of 'a' and a 'set' function
    setTimeout(() => set($a), 1000);
  },
  2000 // Initial value for 'delayed'
     // (before the first update)
);

// Define another derived store 'delayedIncrement'
// based on the original store 'a'
const delayedIncrement = derived(
  a, // The original store 'a'
  ($a, set, update) => {
    // The subscriber function is called with the
    // current value of 'a', a 'set' function, and
    // an 'update' function
    set($a); // Set the initial value immediately

    // Schedule an update to the store after 1000 milliseconds
    setTimeout(() => update((x) => x + 1), 1000);
    // Every time 'a' produces a value,
    // this store produces two values:
    // 1. $a immediately (set function is called)
    // 2. $a + 1 a second later (update function is called)
  }
)
```

If you return a function from the callback, it will be called when a) the callback runs again, or b) the last subscriber unsubscribes.

In both cases, an array of arguments can be passed as the first argument instead of a single store.

Forms

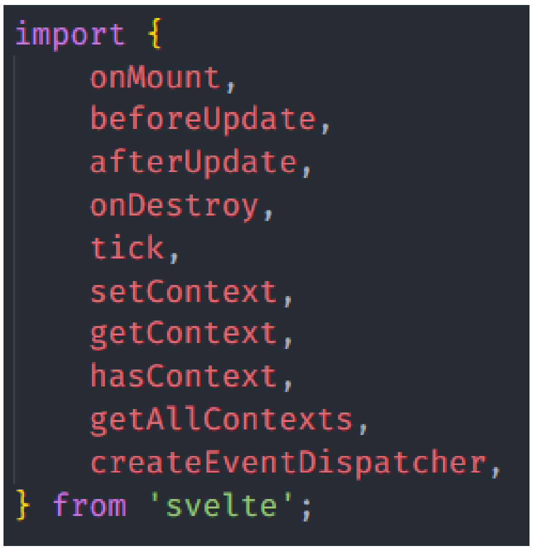
Bind form values

```
<input bind:value={username} />
```

Form modifiers

```
<form on:submit|preventDefault={submitFunction}></form>
```

Lifecycles



```
import {
    onMount,
    beforeUpdate,
    afterUpdate,
    onDestroy,
    tick,
    setContext,
    getContext,
    hasContext,
    getAllContexts,
    createEventDispatcher,
} from 'svelte';
```

onMount

If a function is returned from onMount, it will be called when the component is unmounted.

tick

Returns a promise that resolves once any pending state changes have been applied, or in the next microtask if there are none.

```
console.log('the component is about to update')
await tick()
console.log('the component just updated')
```

setContext

```
setContext('answer', 42)
```

createEventDispatcher

```
<button on:click={() => dispatch('notify', 'detail value')}>Fire
Event</button>
<Child on:notify={callbackFunction} />
```

Sources

- [Fireship - Learn to Code Faster]
- [Svelte • Cybernetically enhanced web apps]

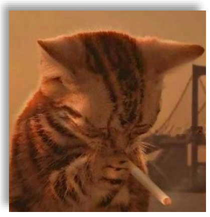
Made by :

Matthias Brat

Socials:

github.com/matthiasbrat

stackoverflow.com/users/17921879/matthias-b



matthiasbrat.dev