Ilionx Devdays

“Modern webapps with Svelte”

Peter Kassenaar –
info@kassenaar.com


Search Google Maps
See travel times, traffic and nearby places
Zutphen, NL
Amsterdam
Rotterdam
The Hague
Utrecht
Arnhem
Nijmegen
Apeldoorn
Deventer
Zwolle
Enschede
Münster
Dortmund
Essen
Duisburg
Düsseldorf
Wuppertal
Antwerp
Ghent
Bruges
Eindhoven
Tilburg
Breda
Satellite
Google

Peter Kassenaar

- Sr. Frontend developer @Conclusion Confidential
- Specialty: *“Everything JavaScript”*
- JavaScript, ES6, Angular, NodeJS, TypeScript, jQuery, React, Vue, etc.

www.kassenaar.com

info@kassenaar.com

Twitter: @PeterKassenaar

ilionx

euricom
A DIMENSION DATA COMPANY

sanoma

ING

OHRA

delta lloyd

woonbron

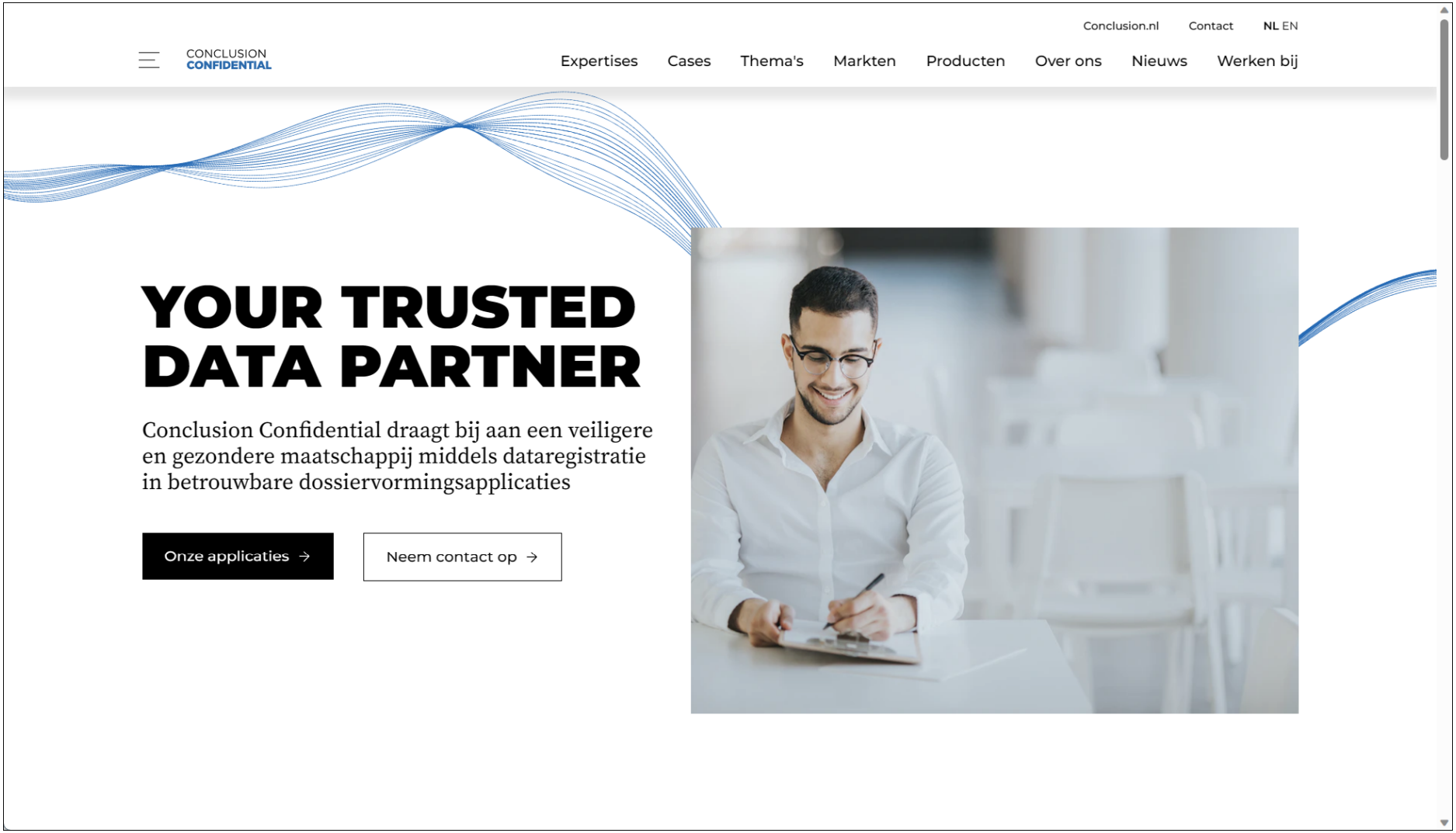
zenito
BETERE ZEKERHEID VOOR ONDERNEMERS

Atos

OBERON INTERACTIVE

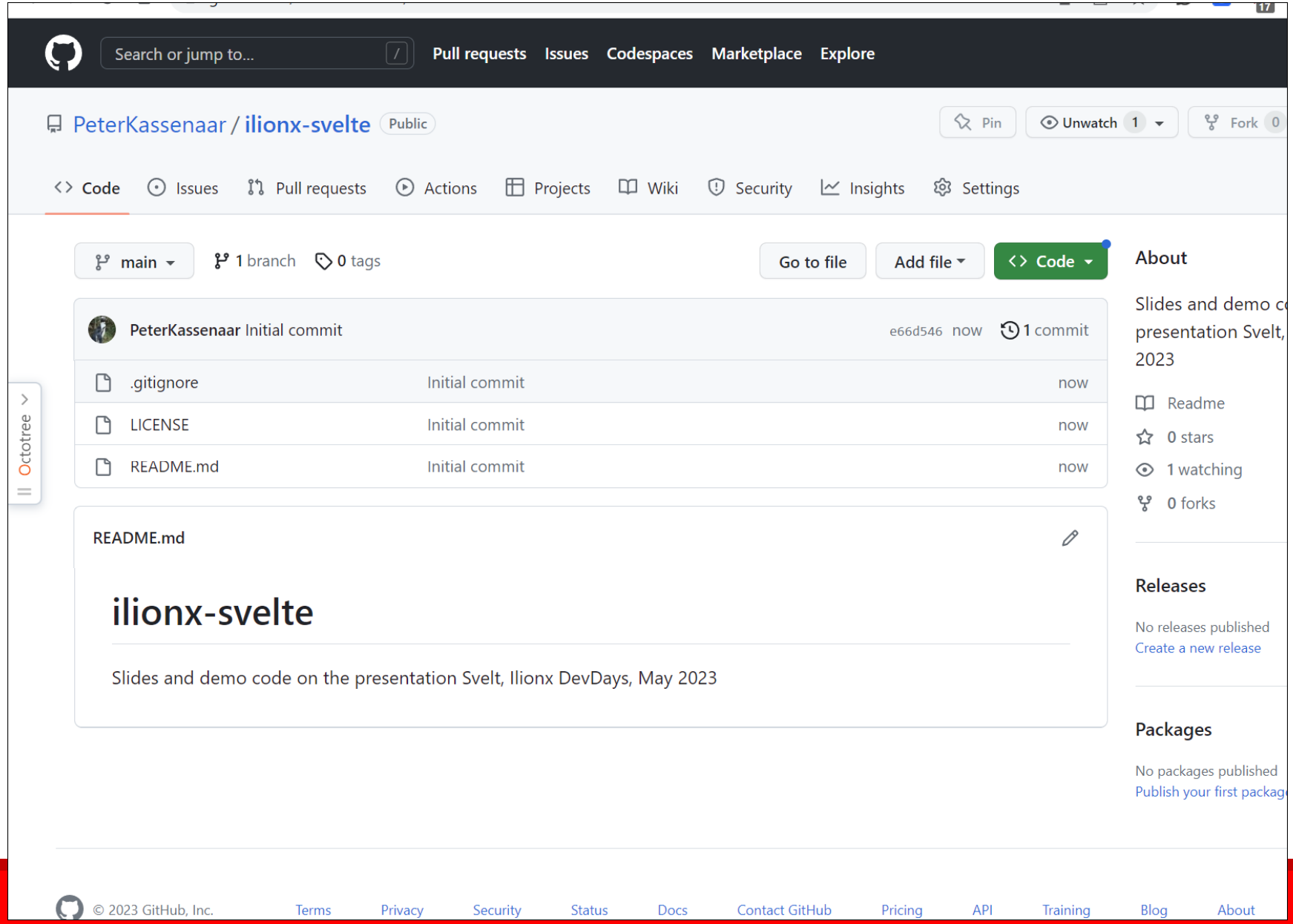
ROC West-Brabant

the eforum FACTORY



https://www.conclusion.nl/confidential

github.com/PeterKassenaar/ilionx-svelte



Agenda

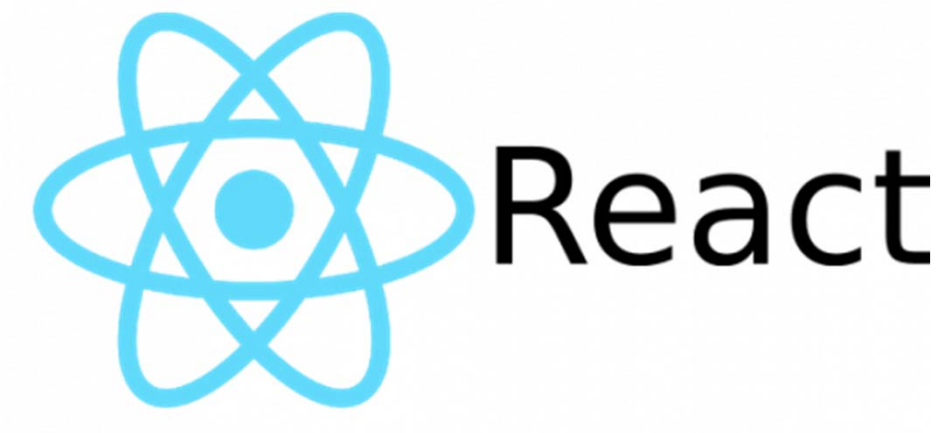
- Introduction – what is Svelte, comparison
- Svelte tooling – installation
- The structure and architecture of basic Svelte apps.
- Svelte building blocks/concepts

- *Live coding demo* - creating non-trivial applications (time permitting)

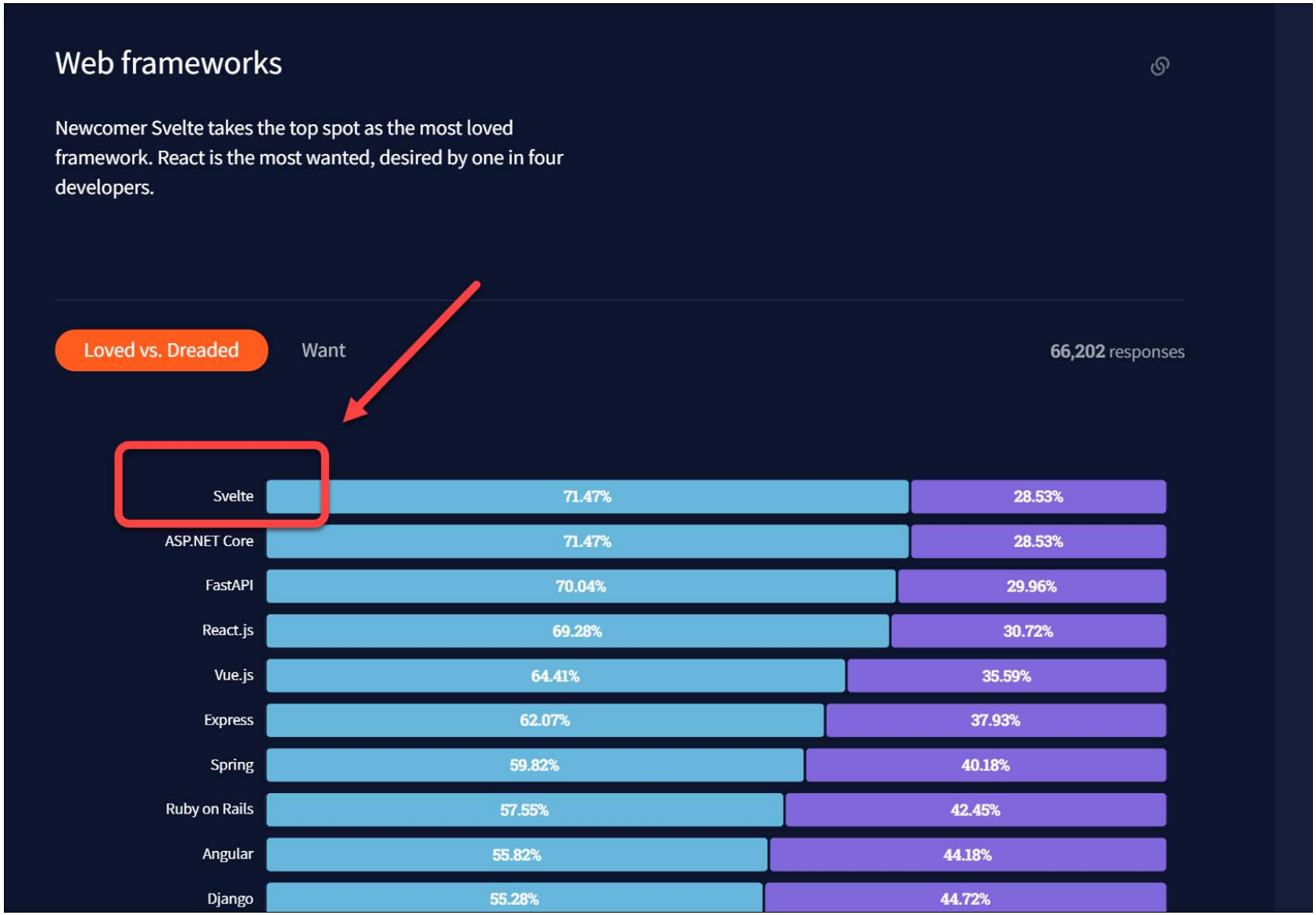
The Frontend Landscape

An overview of the frontend landscape

Front-end Frameworks – the big three



Svelte - “Most loved framework”

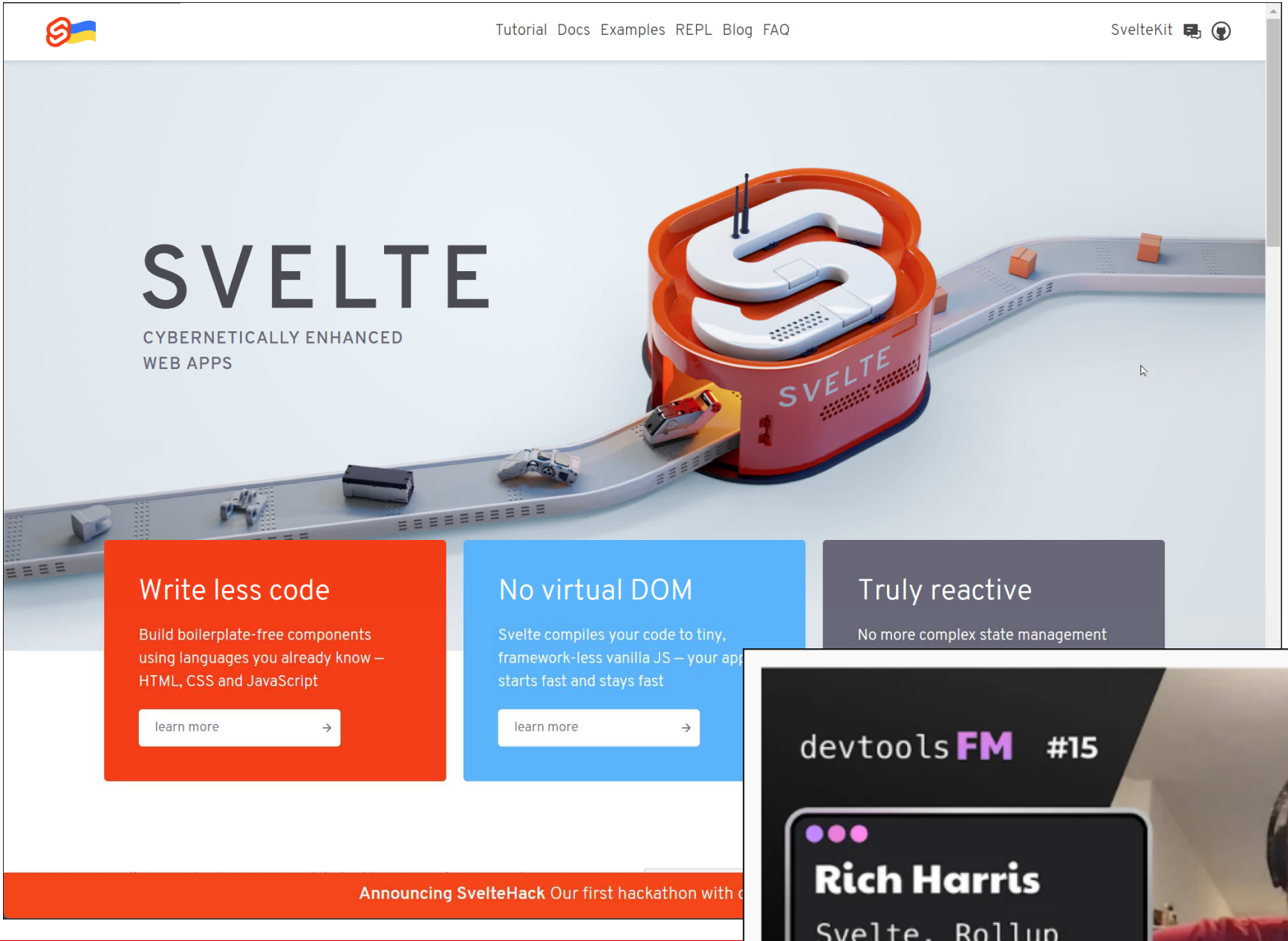


https://insights.stackoverflow.com/survey/2021#section-most-loved-dreaded-and-wanted-web-frameworks

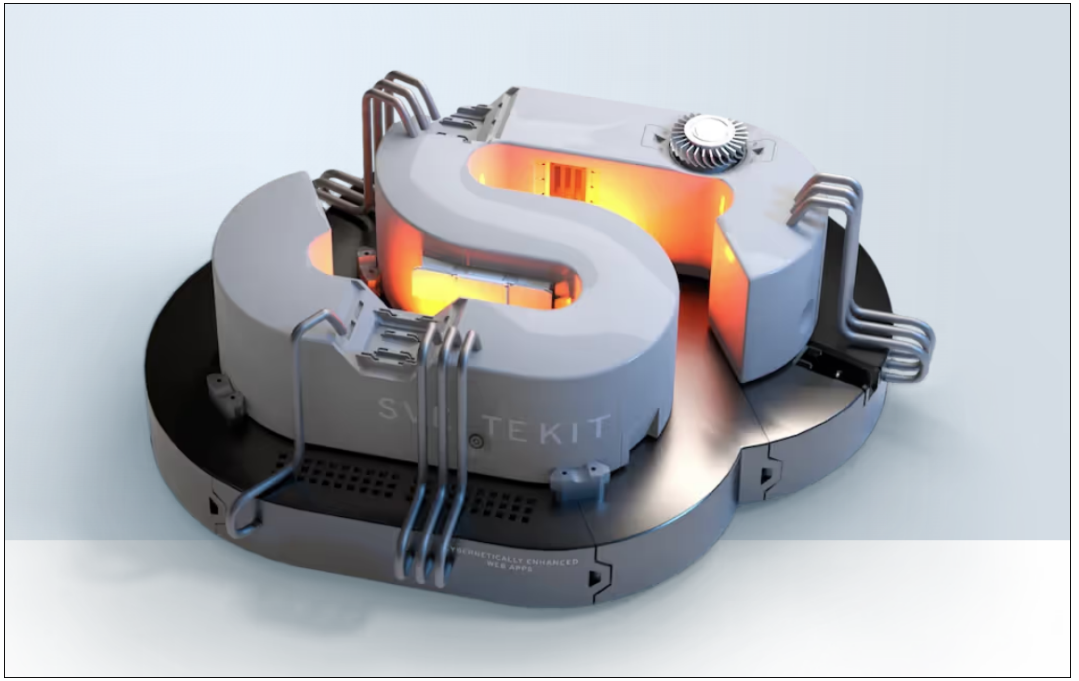
Why Svelte, What is it?

svelte.dev

Svelte – Created by Rich Harris



Svelte? Actually 2 main parts



Svelte

The framework, boilerplate, syntax, compiler, runtime, etc.

Svelte KIT

Application framework, comparable with *Next, NestJS* (React) and *Nuxt* (Vue)

Brief history of Svelte

2016 – Svelte 1 – JavaScript

2018 – Svelte 2 – also JavaScript, better reactivity

2019 – Svelte 3 – written in TypeScript

2021 – SvelteKit – Application Framework on top of Svelte

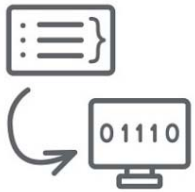
What is Svelte

- JavaScript/TypeScript tool for building UI

  <Components />

- Svelte is a Compiler.
  - No framework is send to the browser
  - `.svelte` code is compiled to JavaScript
  - Directly executed by the browser
  - No Shadow DOM or Virtual DOM

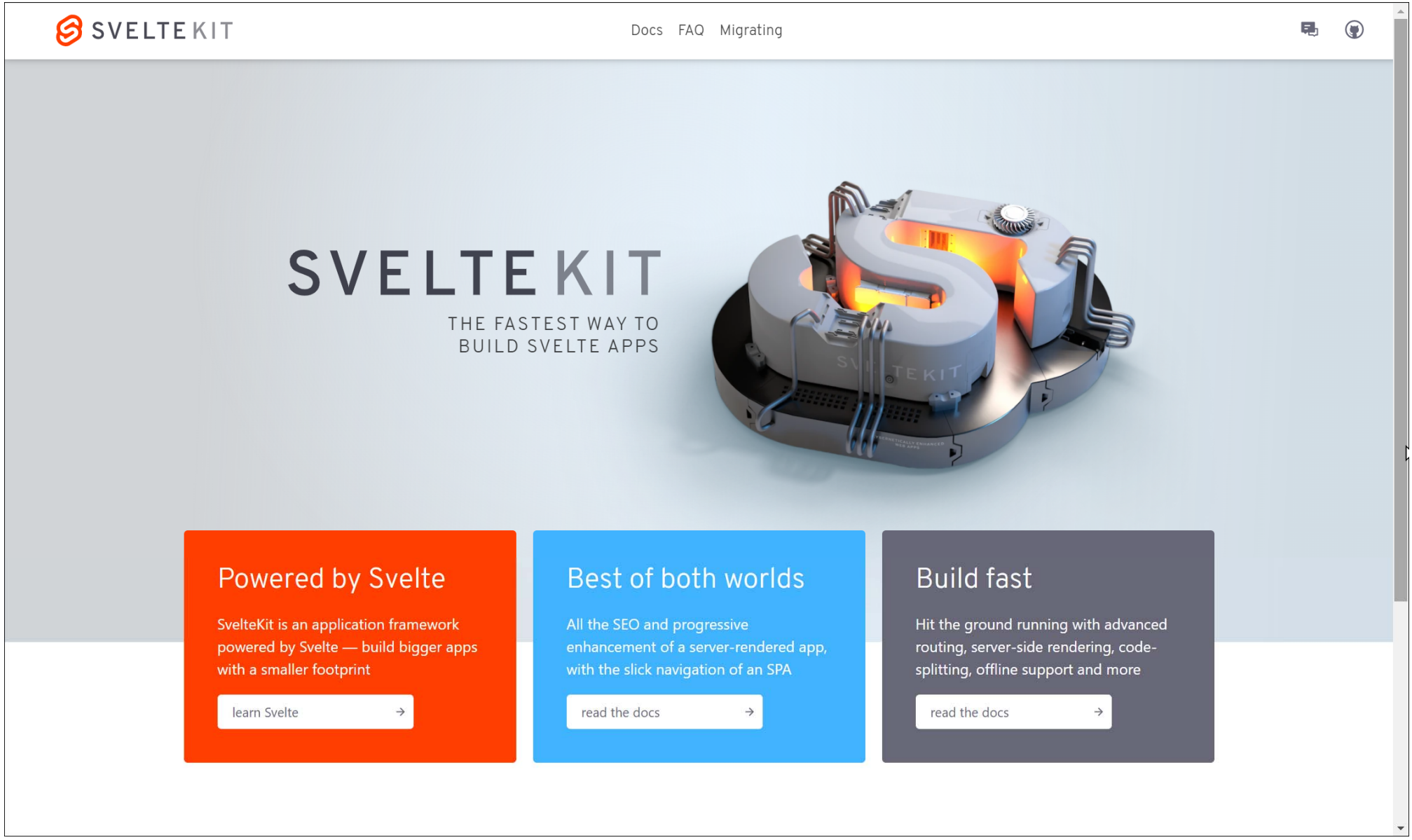
- High performance, small footprint
- So, Svelte runs on the server



COMPILER

"Instead of using techniques like virtual DOM diffing, Svelte writes code that surgically updates the DOM when the state of your app changes."

https://svelte.io/



kit.svelte.dev

What is SvelteKit

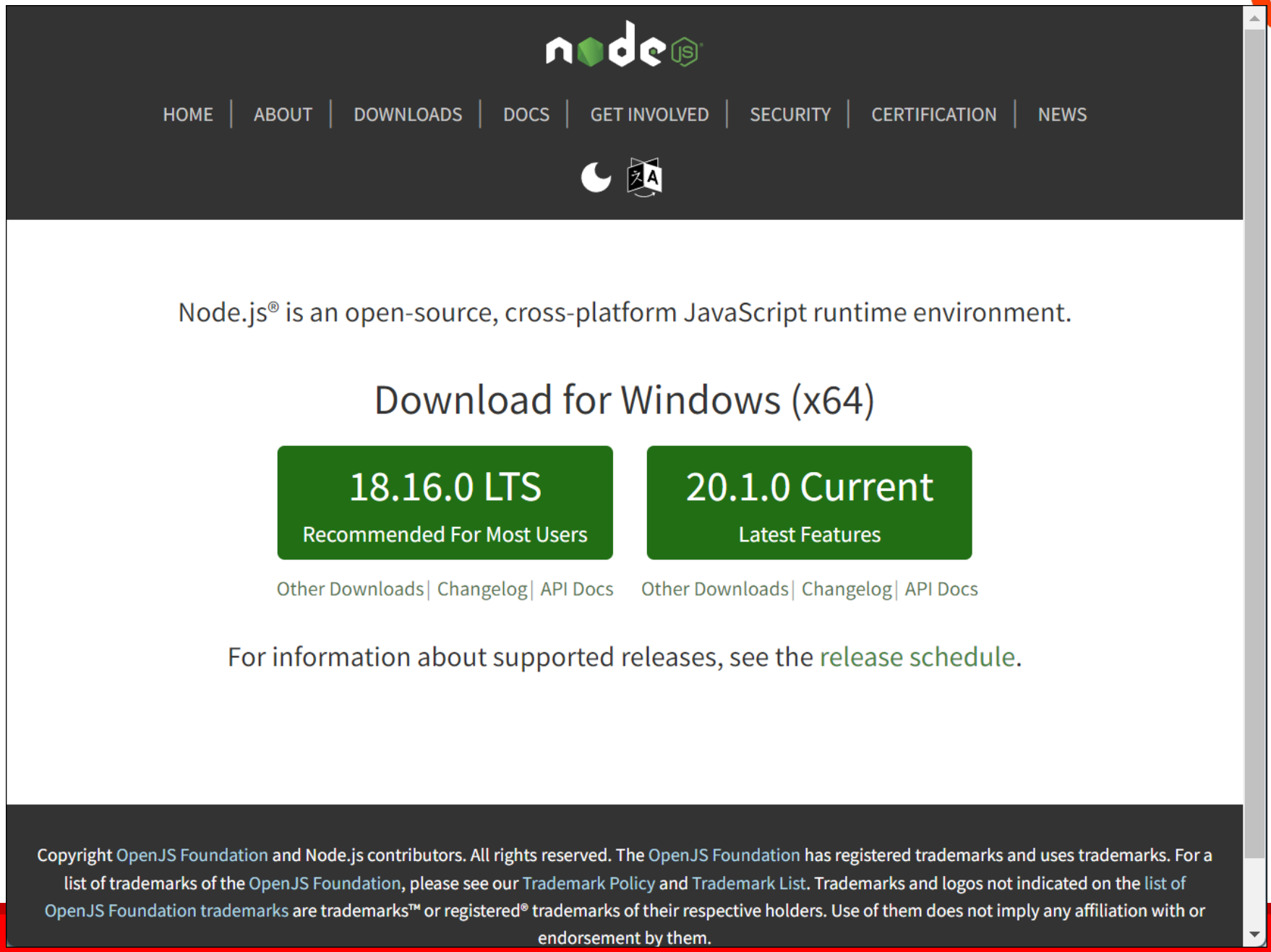
- Built on top of Svelte
- For creating complete Svelte applications
- SEO, SSR, Routing, Code splitting built-in
  - (like Next.js for React, Nuxt for Vue)

“THE way to build apps or websites with Svelte”

Tooling and your first app

What do you need to create Svelte applications?

NodeJS 16.14+ (check your version!)

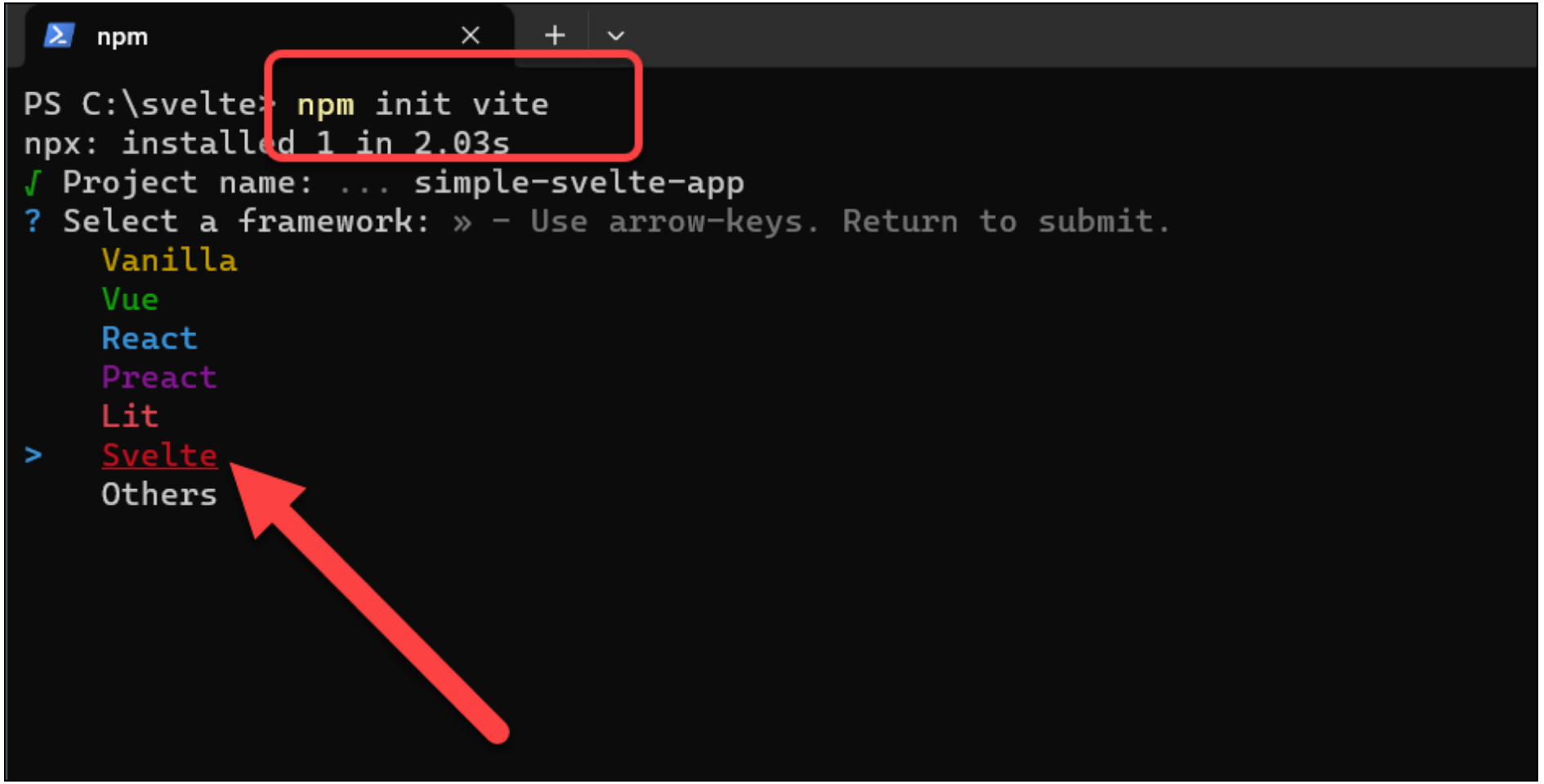


Again, two ways of building an app

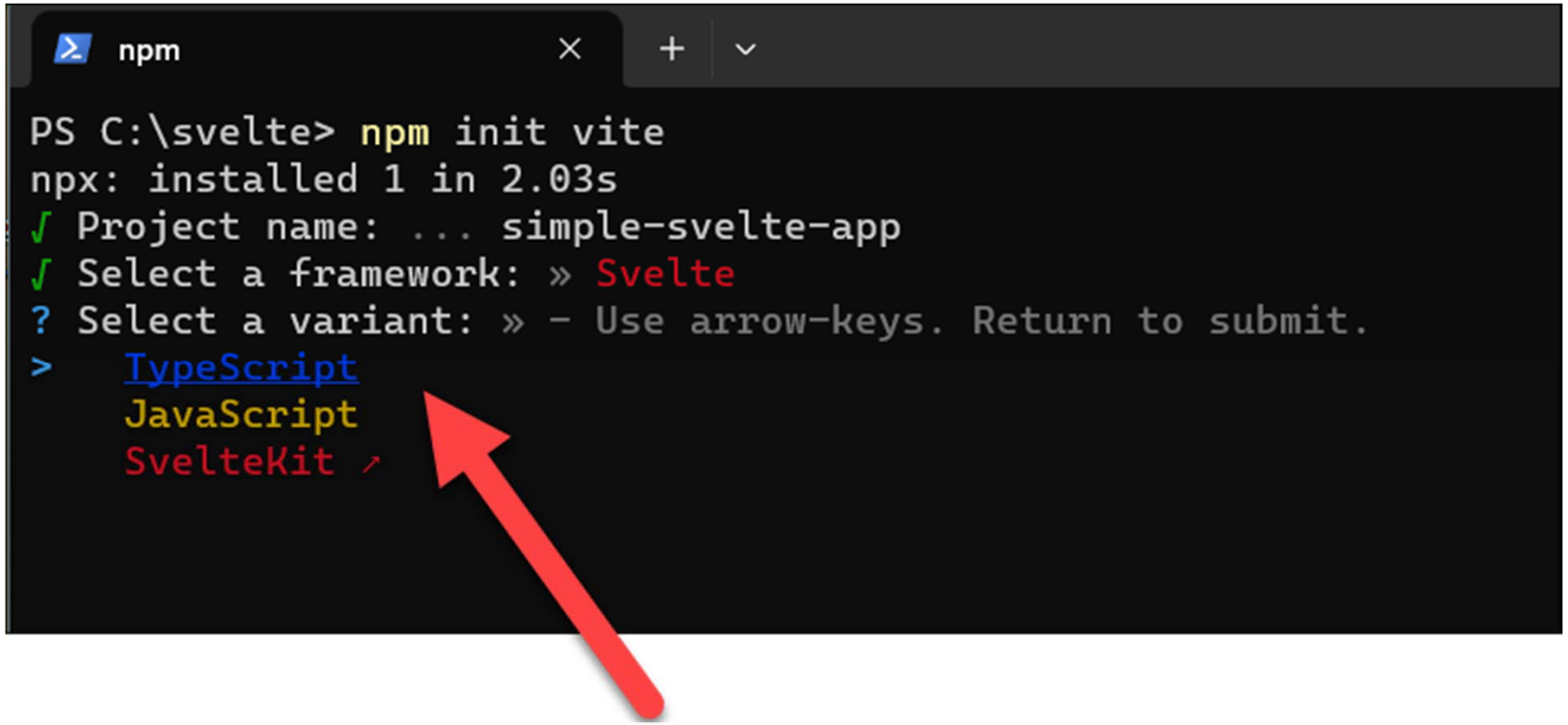
1. `npm init vite` – select `svelte` option
   - build a simple, frontend only site/app
   - `npm run build` will generate HTML, CSS, JS-files in `/dist` directory
   - Mostly just like Angular, React, Vue
2. `npm create svelte@latest my-svelte-app`
   - Build a Svelte KIT app
   - Including *development server*, *routing*, *SSR* and more

https://svelte.dev/docs#getting-started

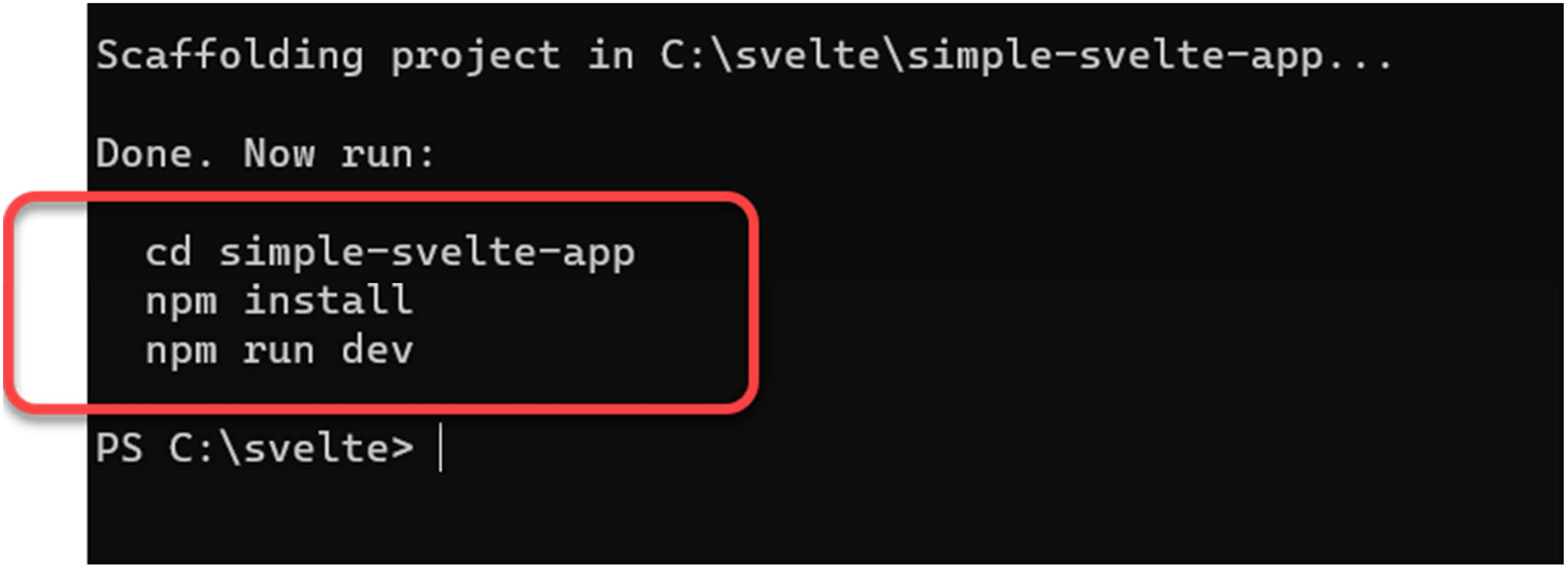
1. Simple Svelte app



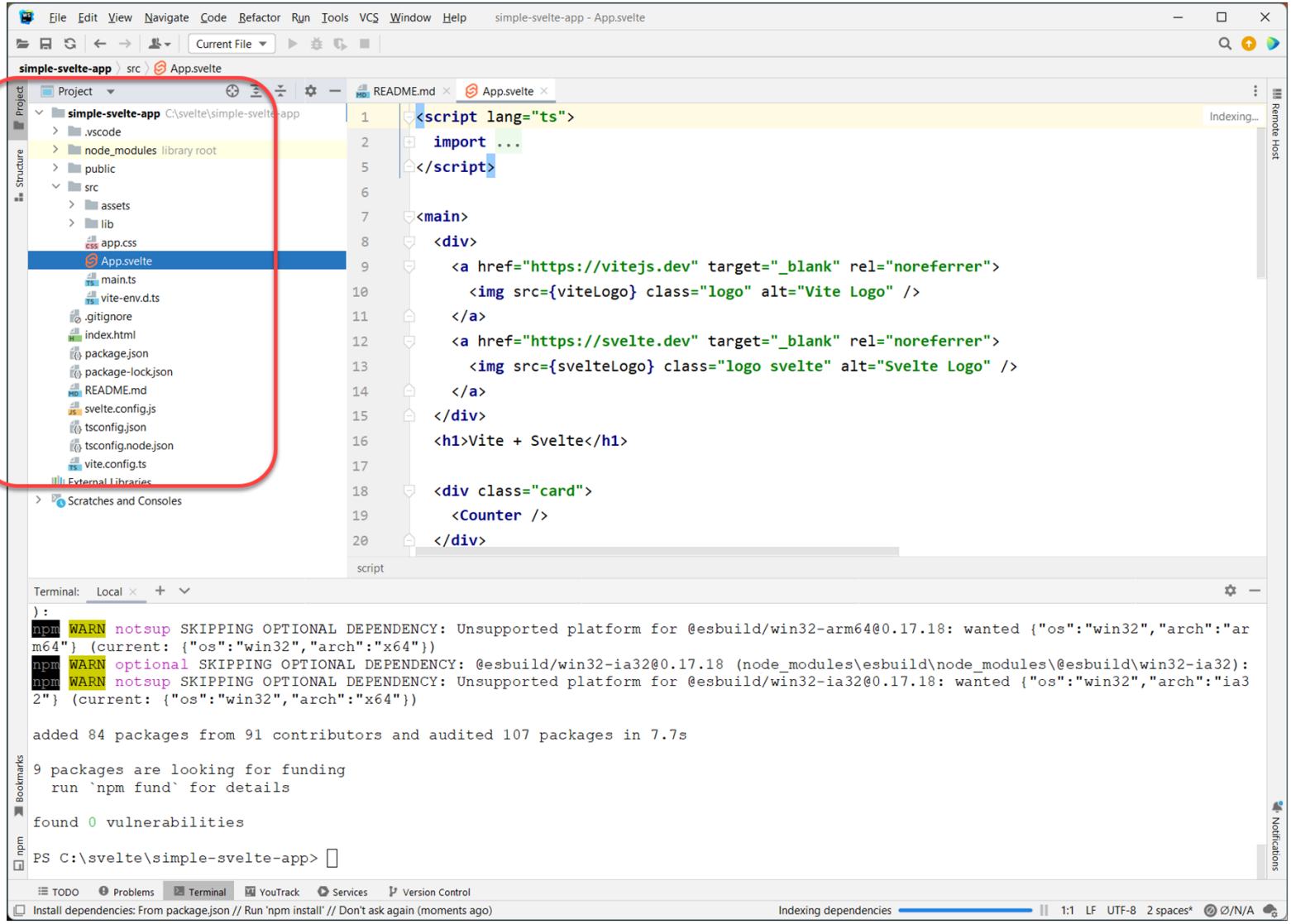
Modern option: pick TypeScript



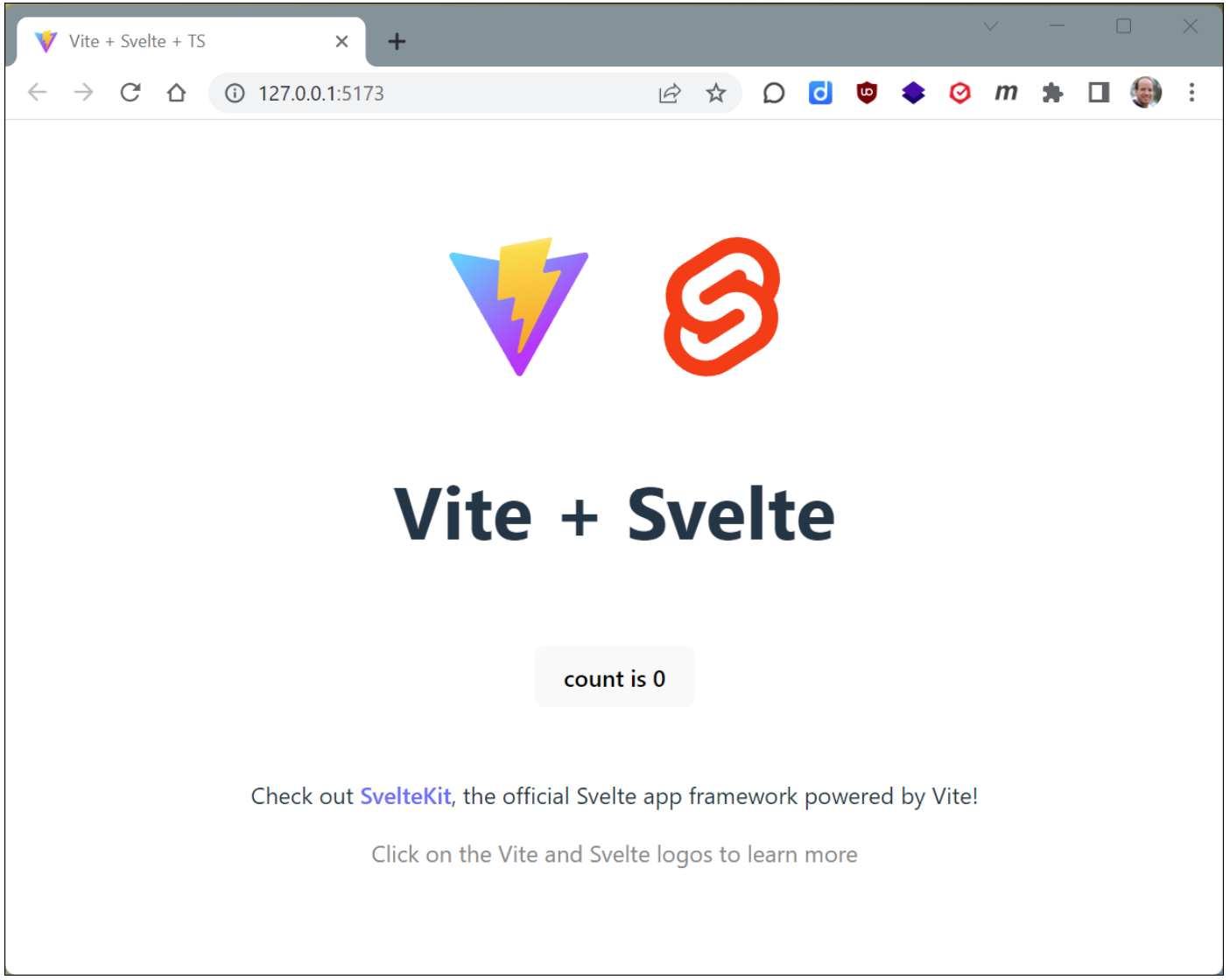
cd into directory and run app



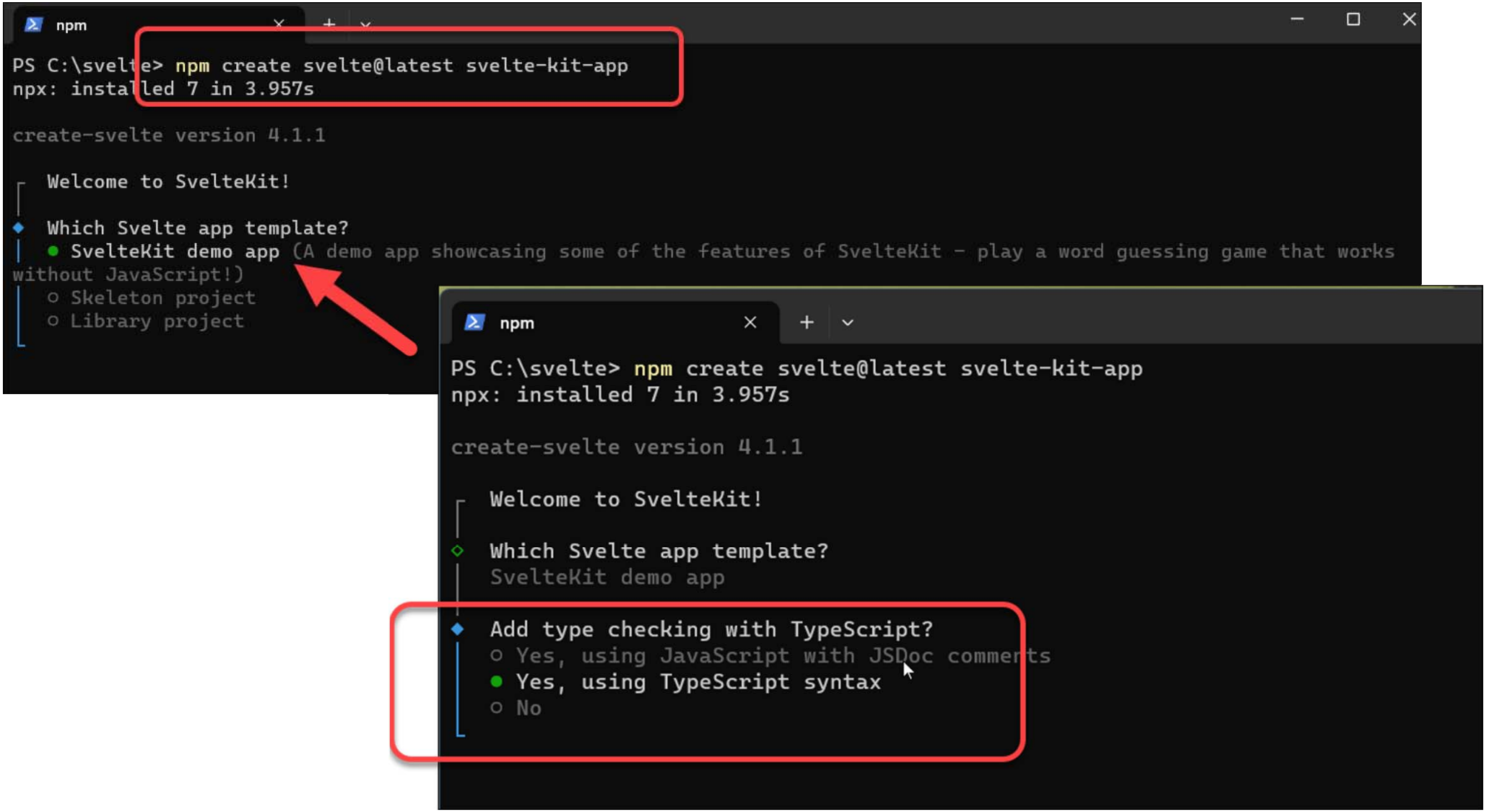
npm install && npm run dev



Open localhost http://127.0.0.1:5173/



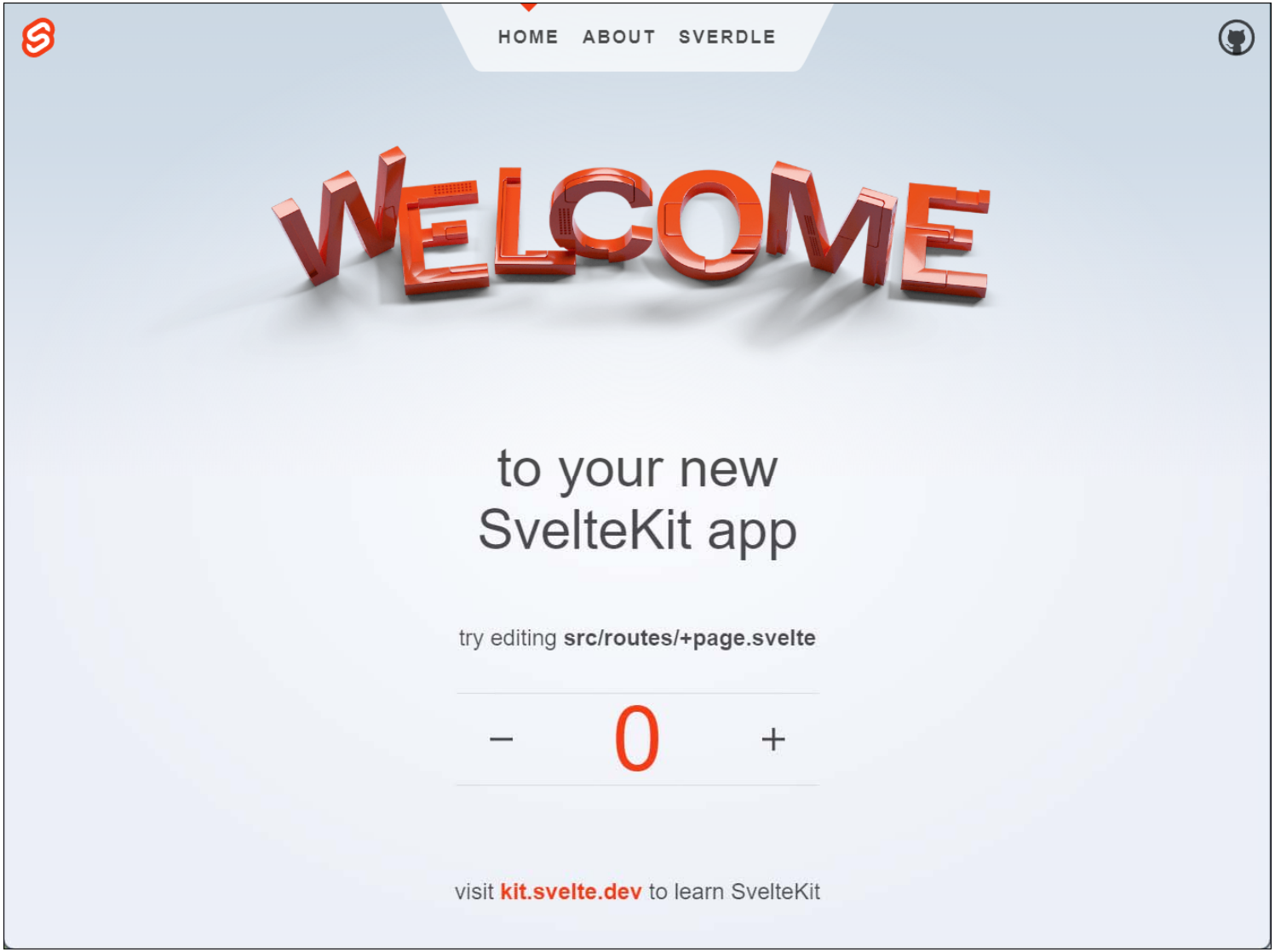
2. Using Svelte KIT



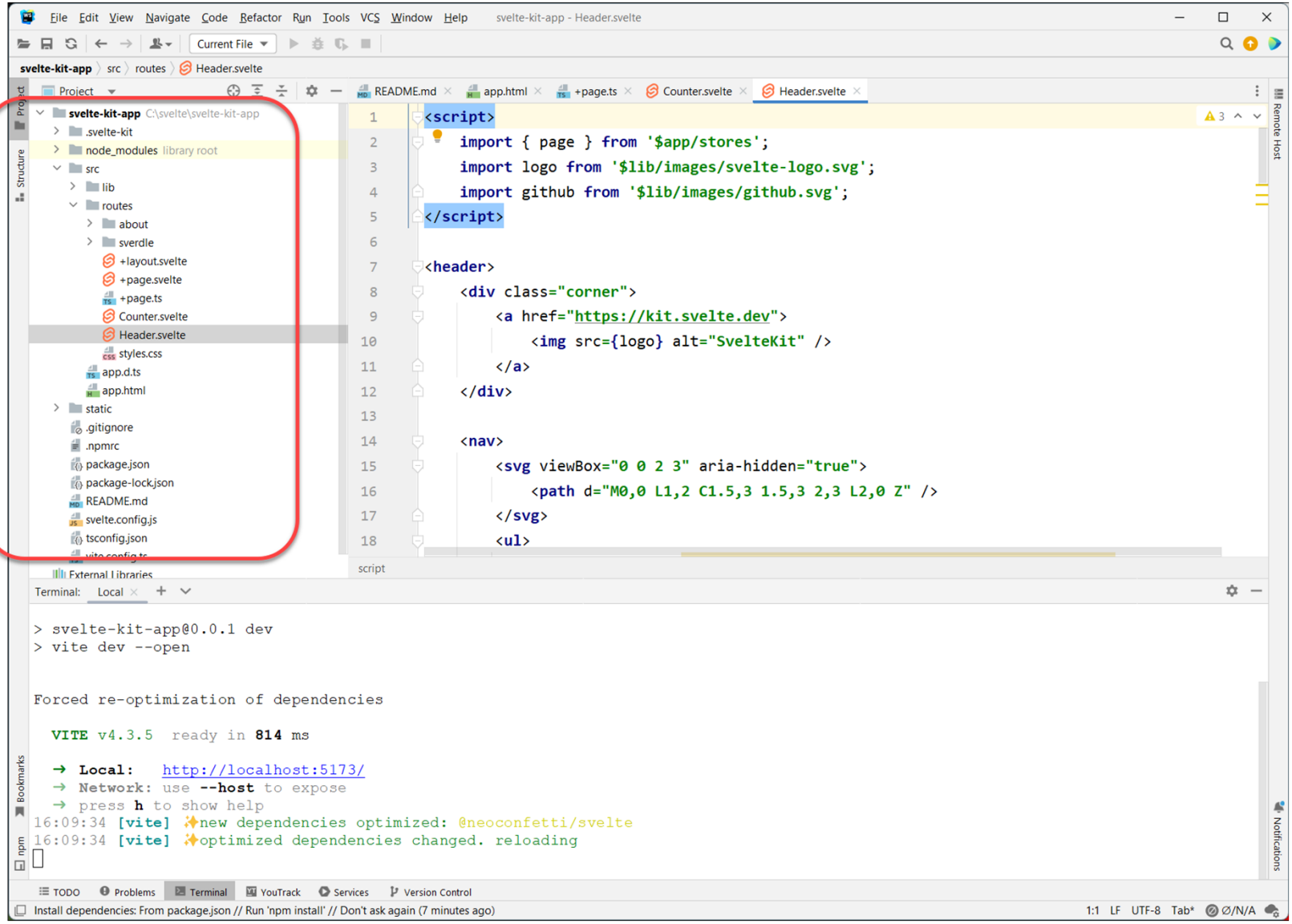
cd into directory and npm install

```
Install community-maintained integrations:
  https://github.com/svelte-add/svelte-add

Next steps:
  1: cd svelte-kit-app
  2: npm install (or pnpm install, etc)
  3: git init && git add -A && git commit -m "Initial commit" (optional)
  4: npm run dev -- --open

To close the dev server, hit Ctrl-C

Stuck? Visit us at https://svelte.dev/chat
PS C:\svelte>
```

`npm run dev -- --open`



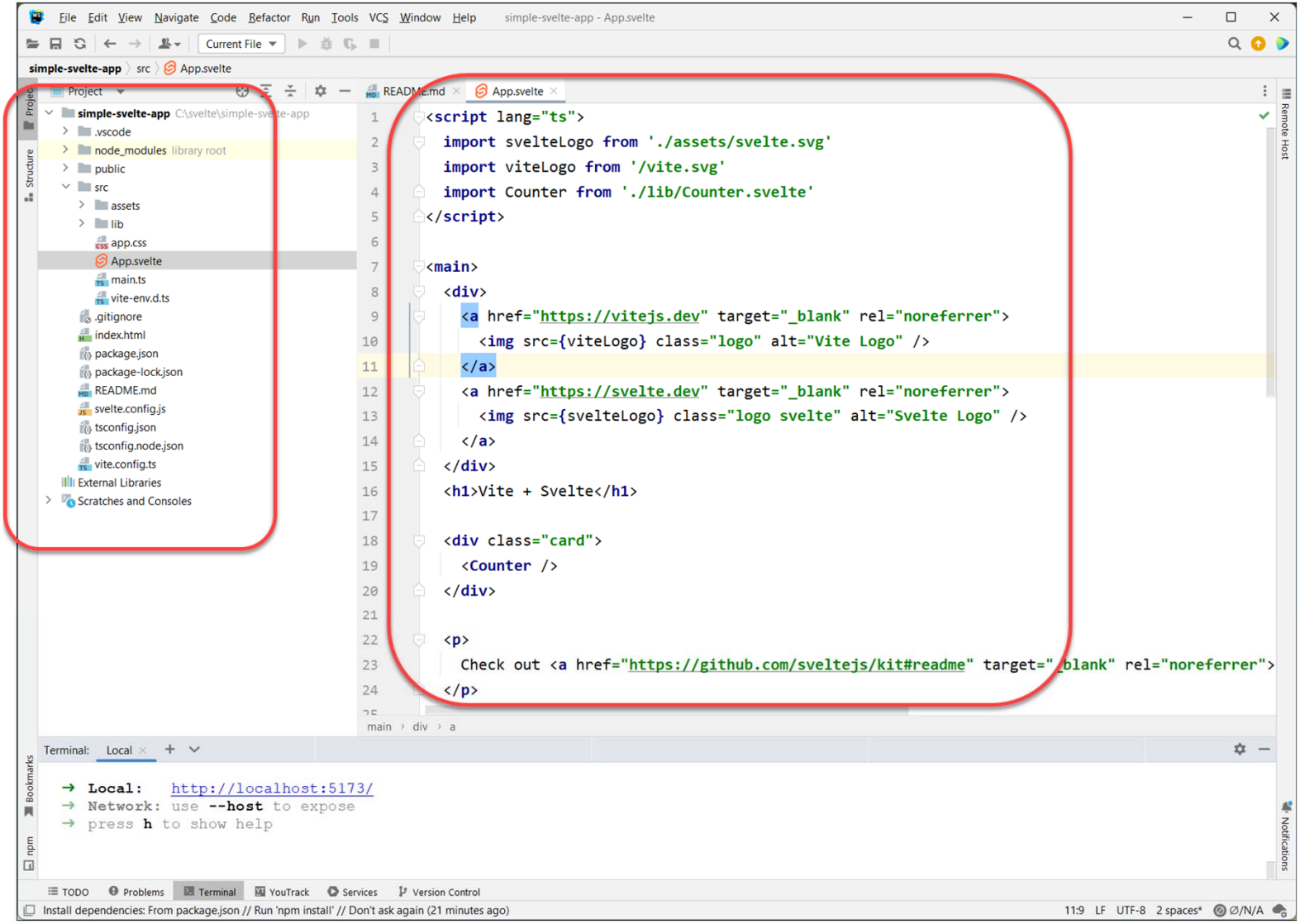
More complex application – study the files



...looking at pure Svelte framework

App structure, file structure, directives

Overall File structure



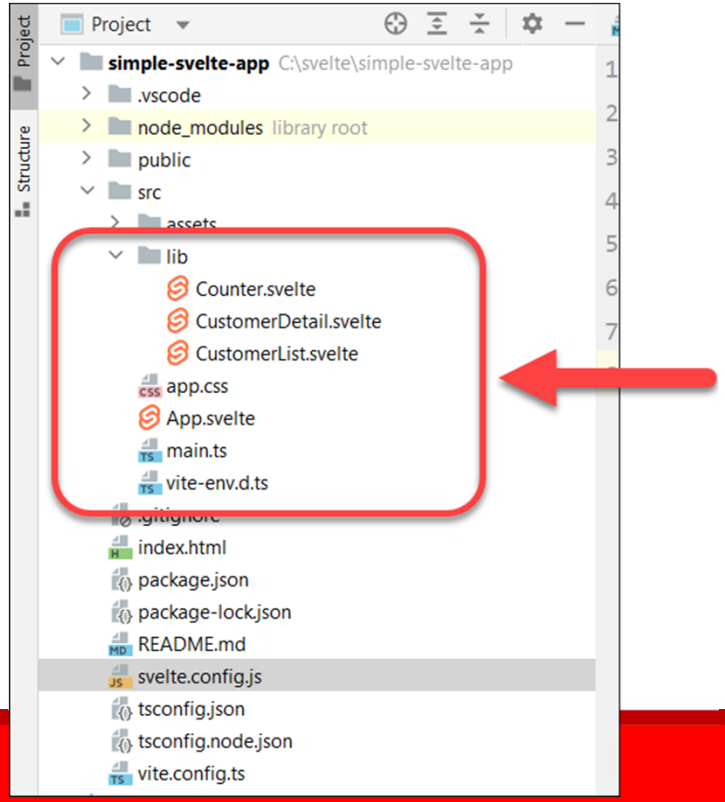
src/App.svelte

Components and architecture

What does your app look like, how do you work on it?

Svelte Components

- Components have the `.svelte` extension
- Are compiled and bundled by vite by default
  - (you can configure Rollup or WebPack if you want to)



Anatomy of a component

```
<script>
    // All JavaScript and/or TypeScript here.
    // This acts as the 'controller' for the component.
    // language attributes are optional.
</script>

<style lang="scss">
    /* all CSS or SASS, LESS, SCSS here*/
</style>

<!--All other HTML here-->
<h2>Counter Component</h2>
<button>Increment</button>
...
```

Creating variables in your components

```
<script>
    // local variable
    let name='Peter';

    // exported variable, i.e. a 'prop' for this component
    export let count = 0;

</script>
```

```
<script>
   // using the variable and importing other components
   import Counter from "./components/Counter.svelte";
   export let name = 'Peter';
</script>

<main>
   <h1>Hello {name}!</h1>
   <Counter />
</main>
```

Componenten importeren

Creating functions to update state

- Just add `functions` in your script to update variables
  - or use syntax like `const=()=> {…}`

```
<script>
    // variables…
    export let count = 0;

    // increment counter
    const increment = () =>{
        count++;
    }
    // decrement counter
    const decrement = () =>{
        count--;
    }

    // reset counter
    const reset = () =>{
        count = 0;
    }
</script>
```

Reactivity

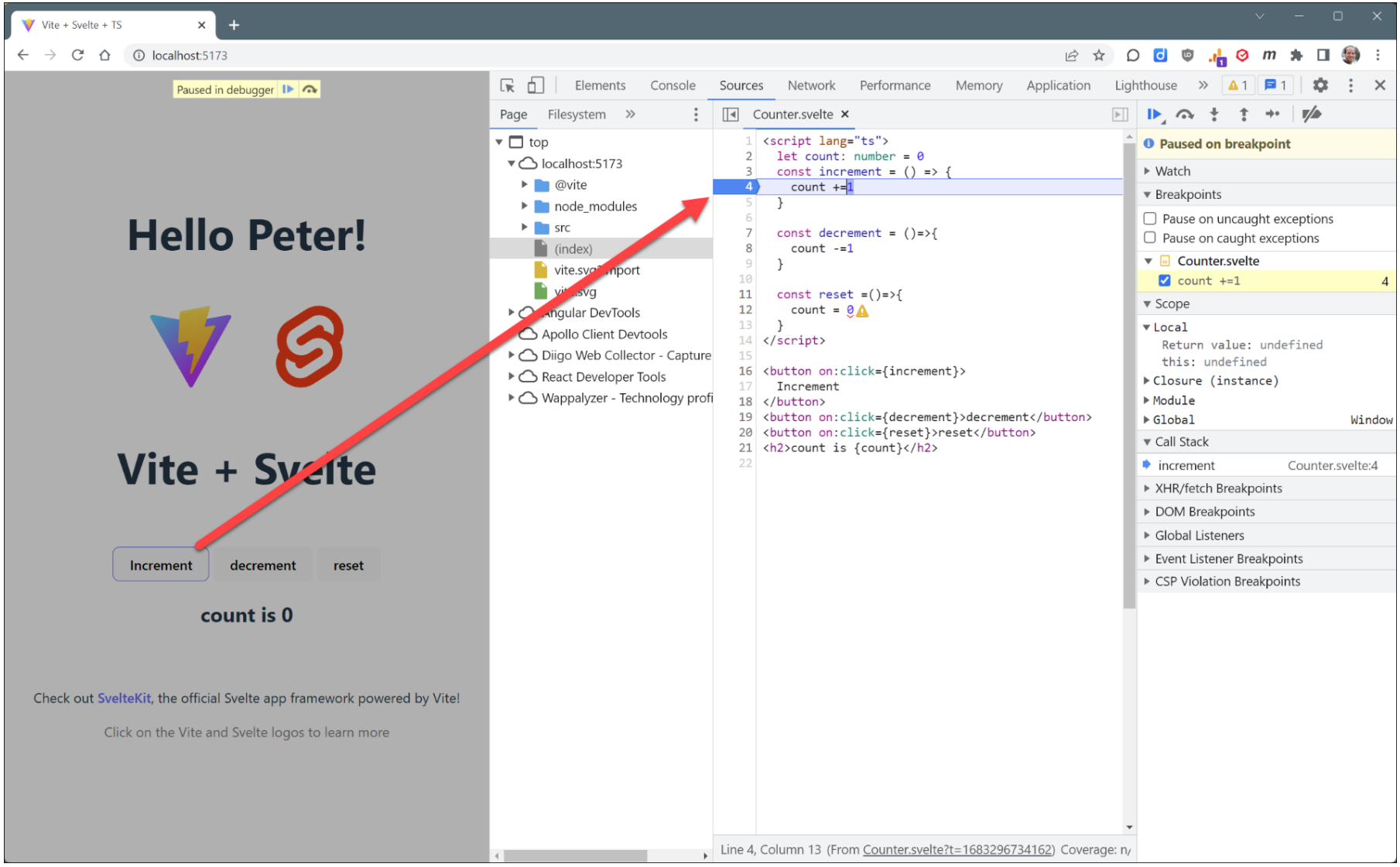
- Change state by listening to events in the DOM with the `on:<eventName>` syntax
  - `on:click={functionName}`
- Component will rerender to show updated state

```
<h1>Counter Component</h1>
<h2>{count}</h2>
<button on:click={increment}>Increment</button>
<button on:click={decrement}>Decrement</button>
<button on:click={reset}>Reset</button>
```

Result


Increment
decrement
reset
count is 3
Check out SvelteKit, the official Svelte app framework powered by Vite!
Click on the Vite and Svelte logos to learn more

Debugging – use Chrome DevTools as normal

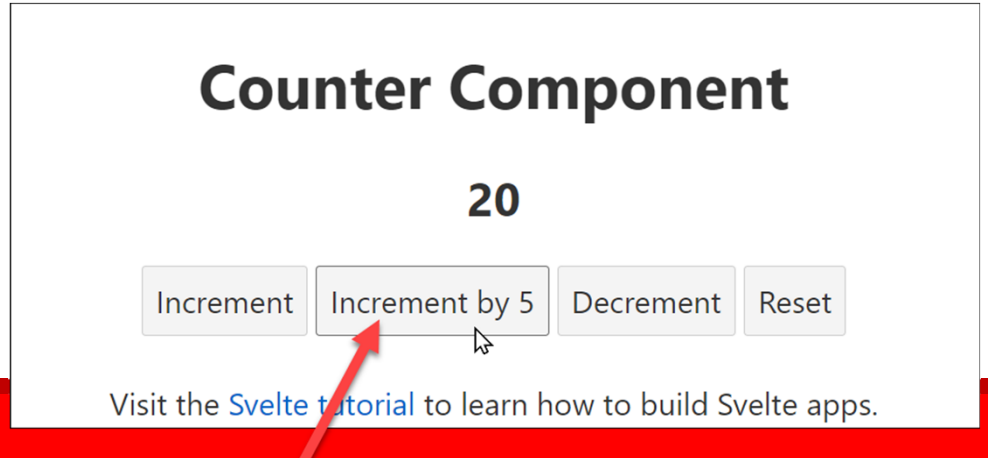


Passing arguments to a function

Use the inline function syntax (like in React)

```
// Update counter with an arbitrary value
const update = val => {
    count += val;
}
```

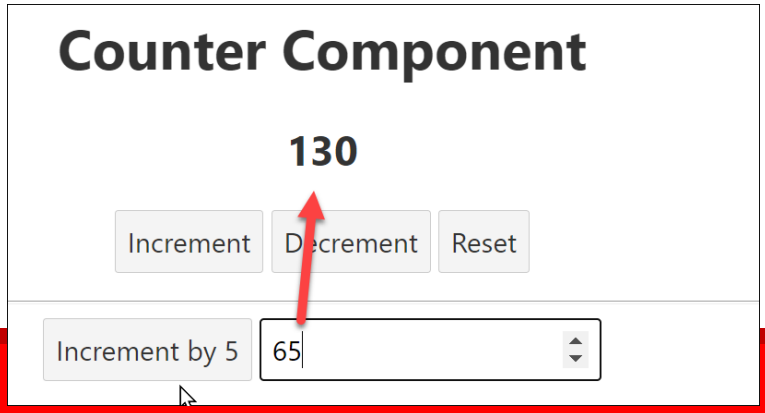
```
<button on:click={()=>update(5)}>Increment by 5</button>
```



Reading values from an input field

- Use the `bind:` directive
- `bind` can be used in multiple ways and on multiple elements

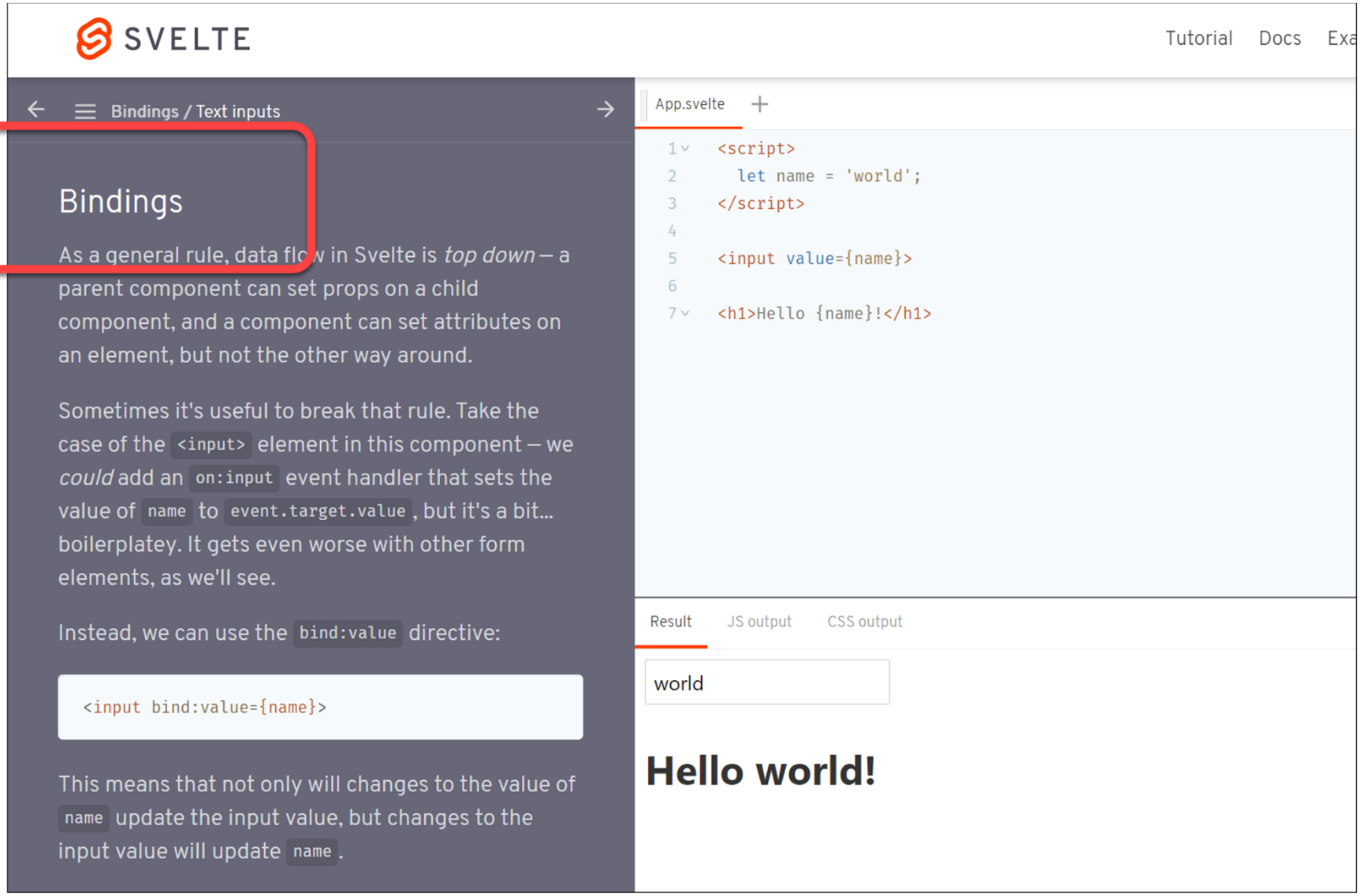
```
<!--Reading values from a text field-->
<input type="number" placeholder="value..."
       bind:value={val}
       on:keyup={e => e.key==='Enter' && update(val)} >
```



If necessary – cast your value!

```
// update value with a random number.
// Parse it explicitly to an integer (there ARE other ways)
const update = val =>{
   count= parseInt(count) +  parseInt(val);
}
```

svelte.dev/tutorial/text-inputs



https://svelte.io/tutorial/text-inputs

Template syntax

Lots of directives / template syntax available

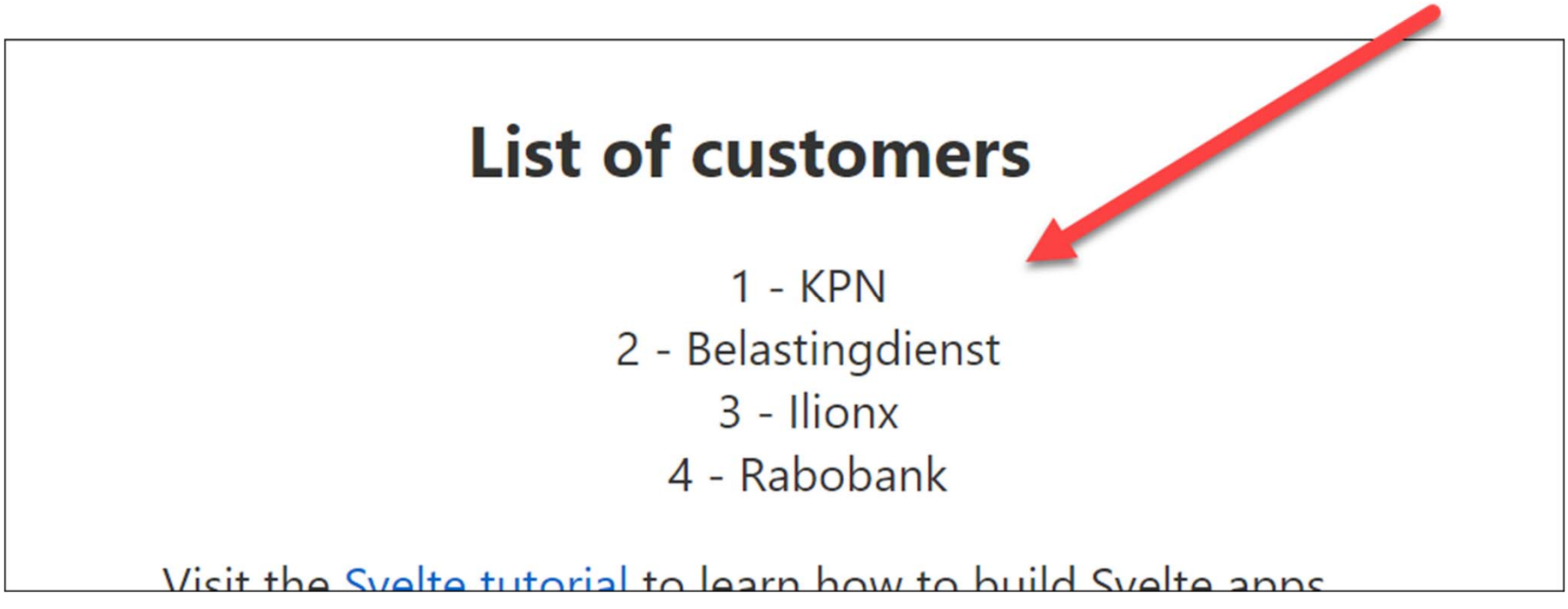
https://svelte.io/docs#Template_syntax

Iterating over a data set {#each}

```
<script>
    // Define array of data
    let customers = [
        {id: 1, name: 'KPN', city: 'Amsterdam'},
        {id: 2, name: 'Belastingdienst', city: 'Apeldoorn'},
        {id: 3, name: 'Ilionx', city: 'Maastricht'},
        {id: 4, name: 'Rabobank', city: 'Utrecht'},
    ]
</script>
```

```
<h2>List of customers</h2>
<ul>
    <!--    Loop over data in array-->
    {#each customers as customer}
        <li>{customer.id} - { customer.name }</li>
    {/each}
</ul>
```

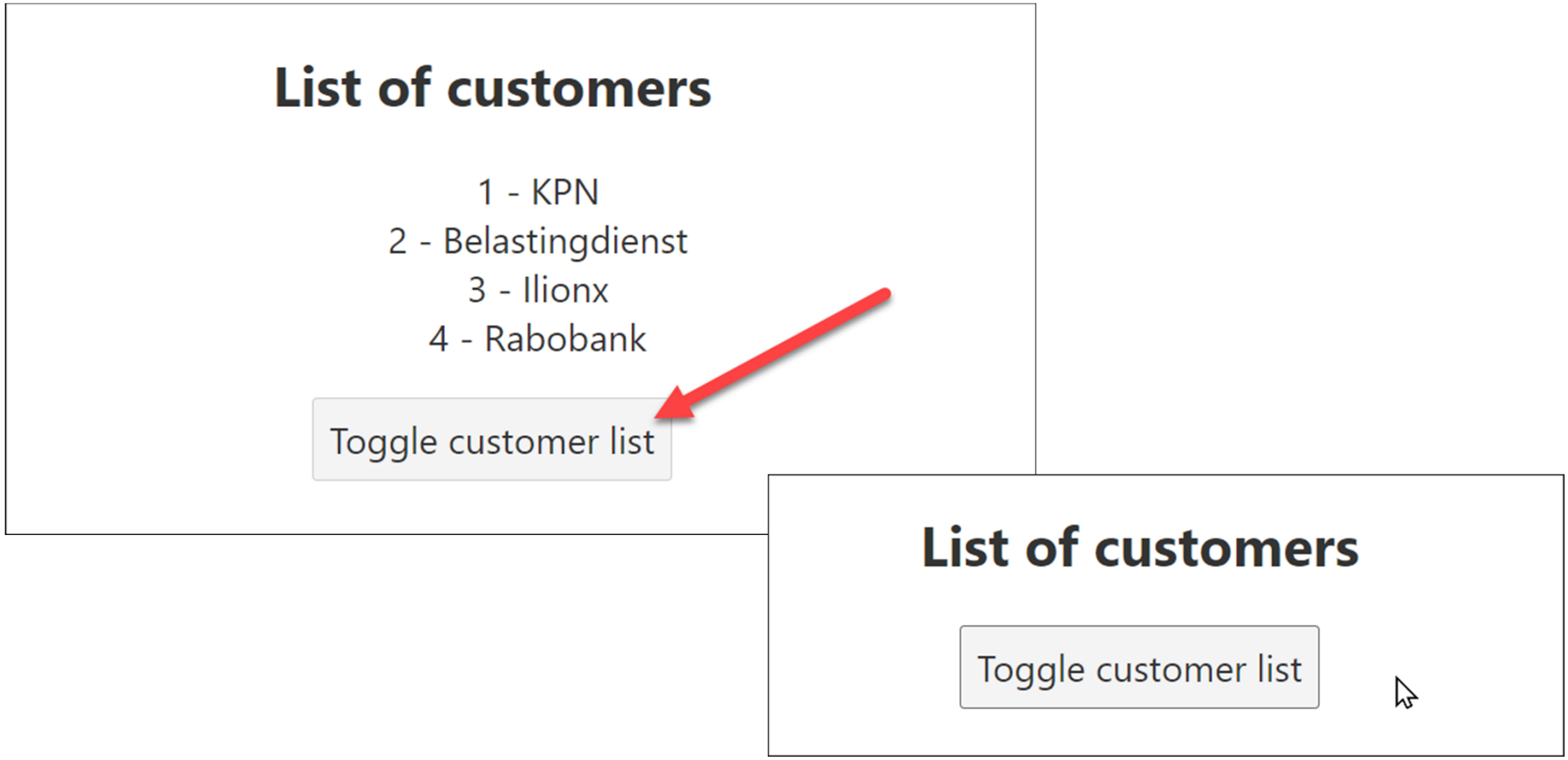
Result


List of customers
1 - KPN
2 - Belastingdienst
3 - Ilionx
4 - Rabobank
Visit the Svelte tutorial to learn how to build Svelte apps

Conditional Logic – {#if}

```
<script>
    // boolean value, used in the UI
    let showCustomers = true;
</script>
<!--html-->
<h2>List of customers</h2>
{#if showCustomers}
    <ul>
        …
    </ul>
{/if}

<button on:click={()=> showCustomers = !showCustomers}>
    Toggle customer list
</button>
```

Result


List of customers
1 - KPN
2 - Belastingdienst
3 - Ilionx
4 - Rabobank
Toggle customer list
List of customers
Toggle customer list

More Svelte Concepts

See documentation for extensive descriptions

https://svelte.dev/docs

Reactive assignments

- To trigger re-render, just (re-)assign local variable

2. Assignments are 'reactive'

To change component state and trigger a re-render, just assign to a locally declared variable.

Update expressions (`count += 1`) and property assignments (`obj.x = y`) have the same effect.

```
<script>
  let count = 0;

  function handleClick () {
    // calling this function will trigger an
    // update if the markup references `count`
    count = count + 1;
  }
</script>
```

Because Svelte's reactivity is based on assignments, using array methods like `.push()` and `.splice()` won't automatically trigger updates. A subsequent assignment is required to trigger the update. This and more details can also be found in the tutorial.

Reactive statements with $

3. $: marks a statement as reactive

Any top-level statement (i.e. not inside a block or a function) can be made reactive by prefixing it with the `$:` JS label syntax. Reactive statements run after other script code and before the component markup is rendered, whenever the values that they depend on have changed.

```
<script>
  export let title;
  export let person;

  // this will update `document.title` whenever
  // the `title` prop changes
  $: document.title = title;

  $: {
    console.log(`multiple statements can be combined`);
    console.log(`the current title is ${title}`);
  }

  // this will update `name` when 'person' changes
  $: ({ name } = person);

  // don't do this. it will run before the previous line
  let name2 = name;
</script>
```

(Compare: Angular does this now also in Angular V16, with `signal()` functions)

Built-in state management

- Store
- Observable (`import {writable} from svelte/store`), `$subscribe`

4. Prefix stores with $ to access their values

A *store* is an object that allows reactive access to a value via a simple *store contract*. The `svelte/store` module contains minimal store implementations which fulfil this contract.

Any time you have a reference to a store, you can access its value inside a component by prefixing it with the `$` character. This causes Svelte to declare the prefixed variable, subscribe to the store at component initialization and unsubscribe when appropriate.

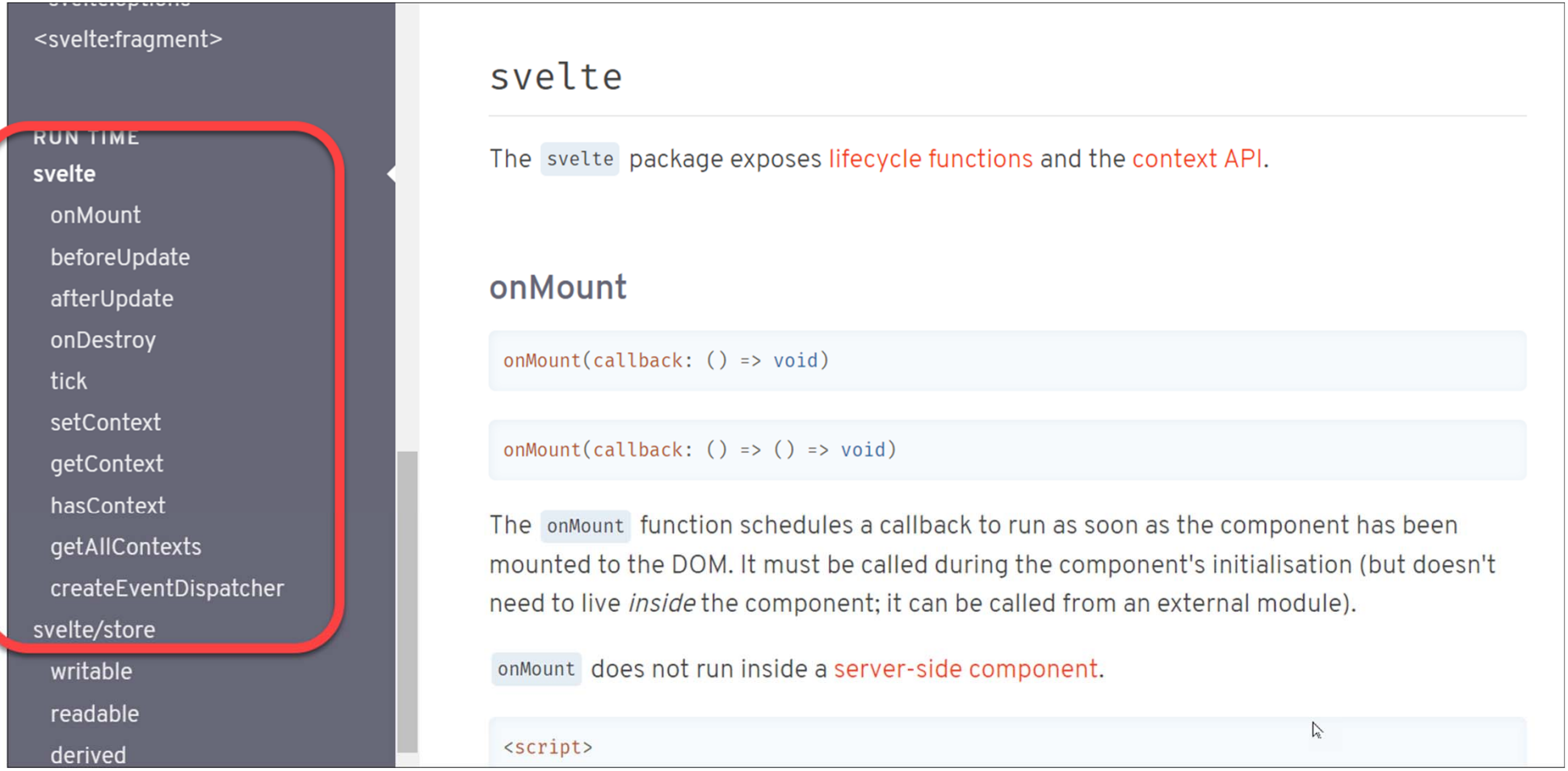
Assignments to `$`-prefixed variables require that the variable be a writable store, and will result in a call to the store's `.set` method.

Note that the store must be declared at the top level of the component — not inside an `if` block or a function, for example.

Extensive Template Syntax

https://svelte.dev/docs#template-syntax-comments

Lifecyle events



https://svelte.dev/docs#run-time-svelte

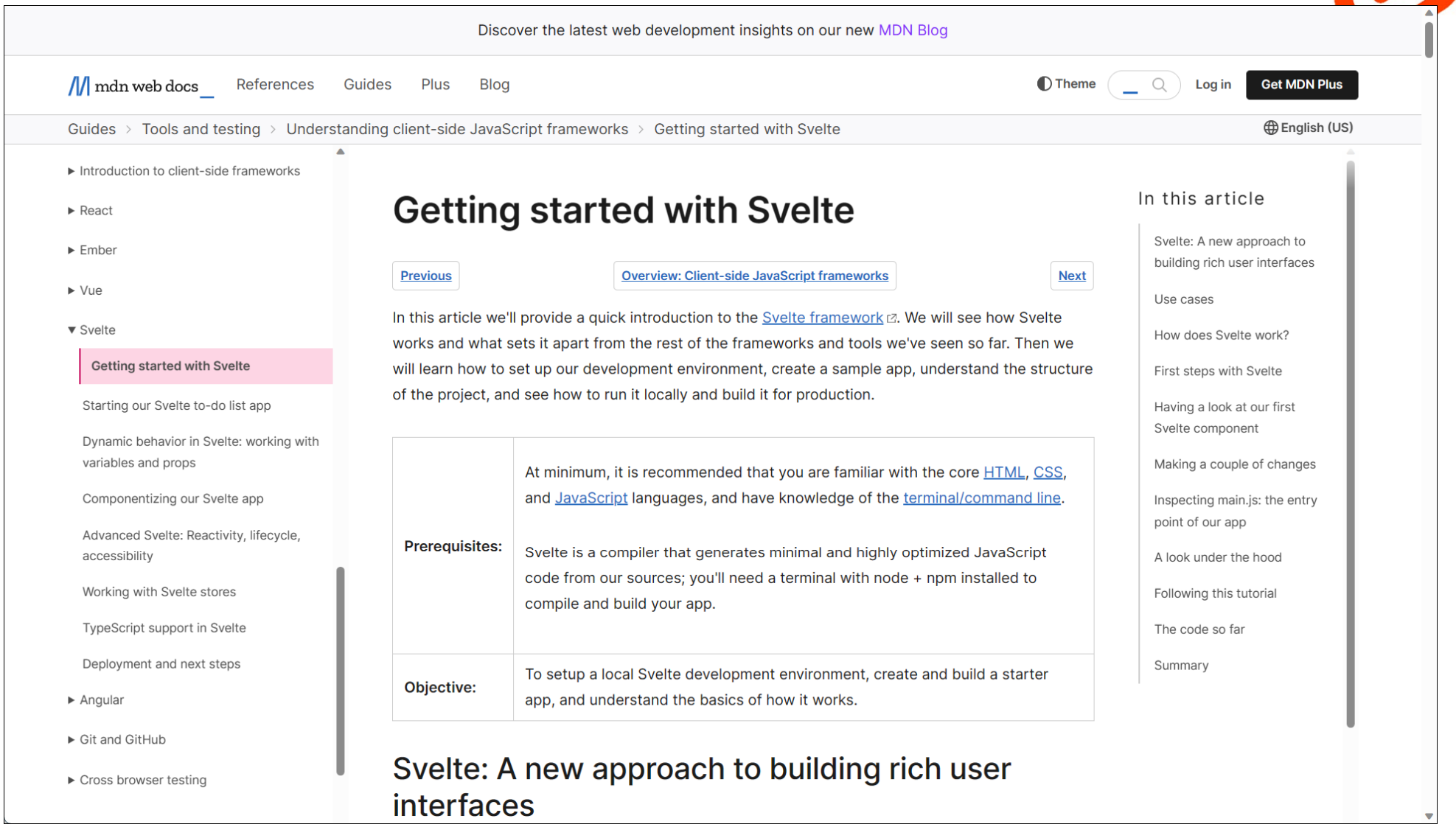
So, PRO's of using Svelte

- Small bundle sizes
- Fast(er) performance
- (very) good documentation and tutorials
- Built-in State Management
- Integration of TypeScript, Sass, PostCSS
- SvelteKit as first-class citizen

CON's, downsides of using Svelte?

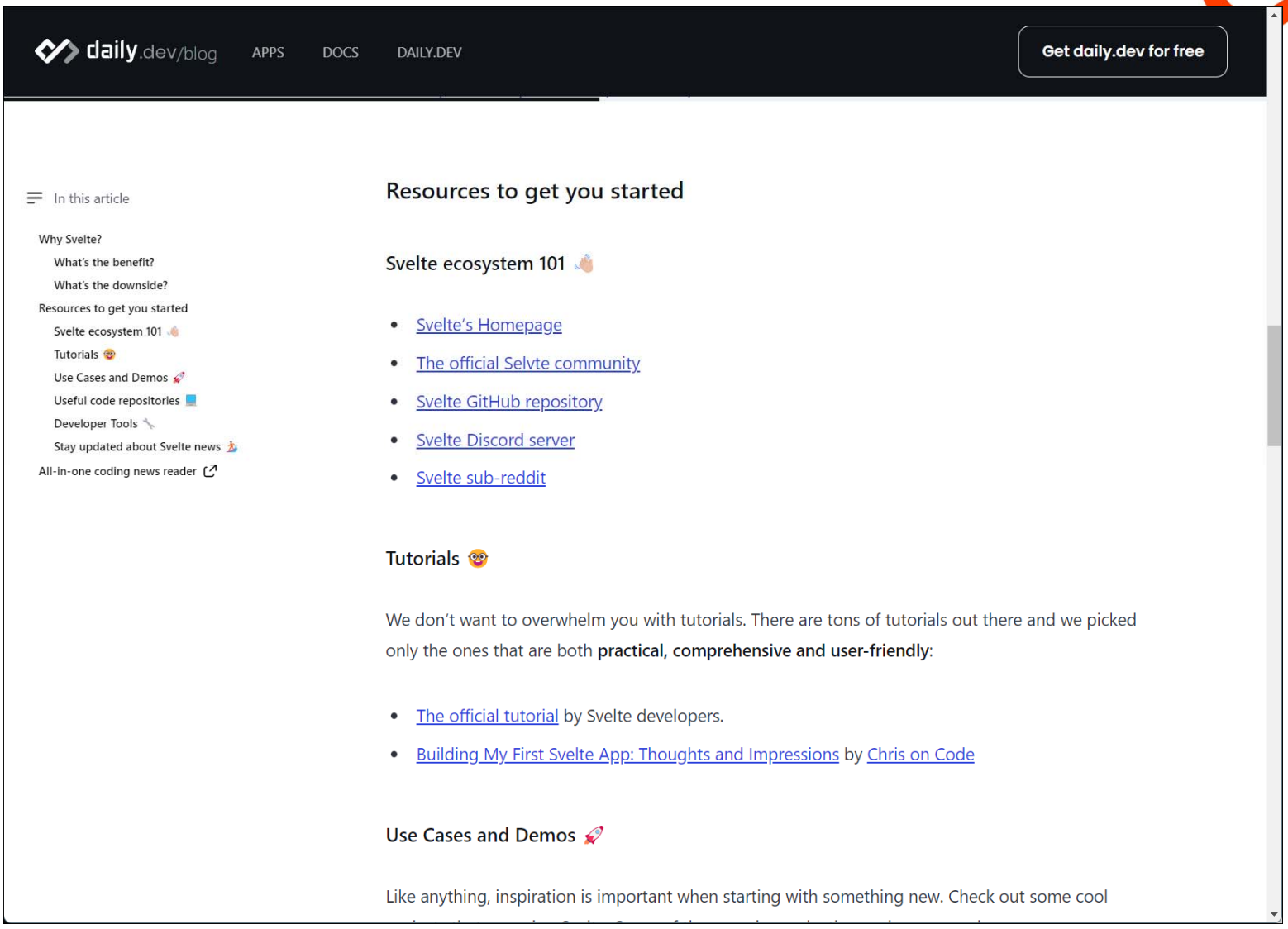
- Small(er) Open Source Svelte-ecosystem
  - UI libraries, Testing support, Stack Overflow, etc.
- More difficult to find experienced programmers for your company
- Early stages – a lot of `build your own`
- You are on the vanguard / you might not like that
- Forwarding events to child components not fluent
- https://www.reddit.com/r/sveltejs/comments/oxrv1r/svelte_pros_and_cons/

More info on Svelte on the interwebzzz



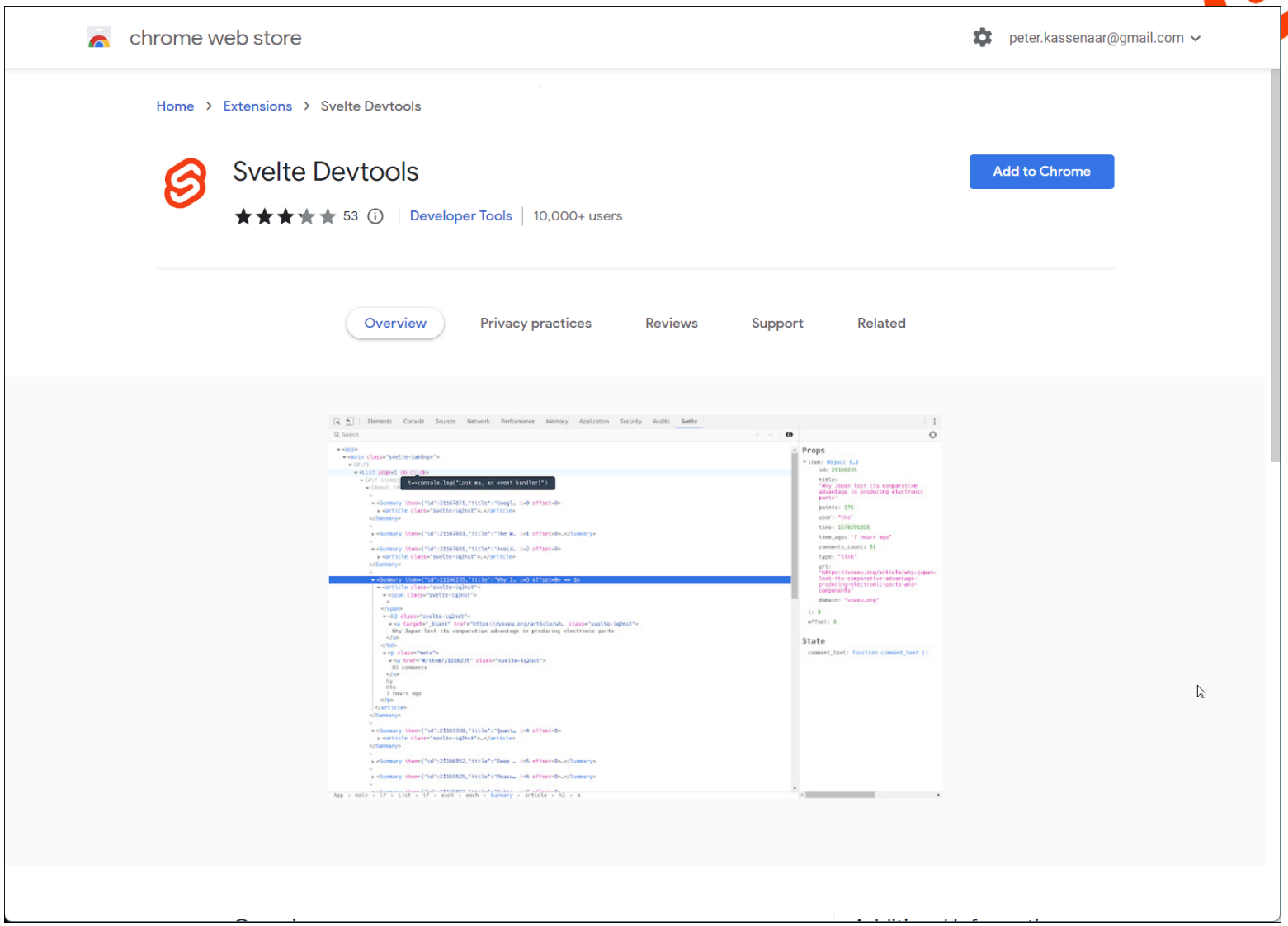
https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Svelte_getting_started

daily.dev



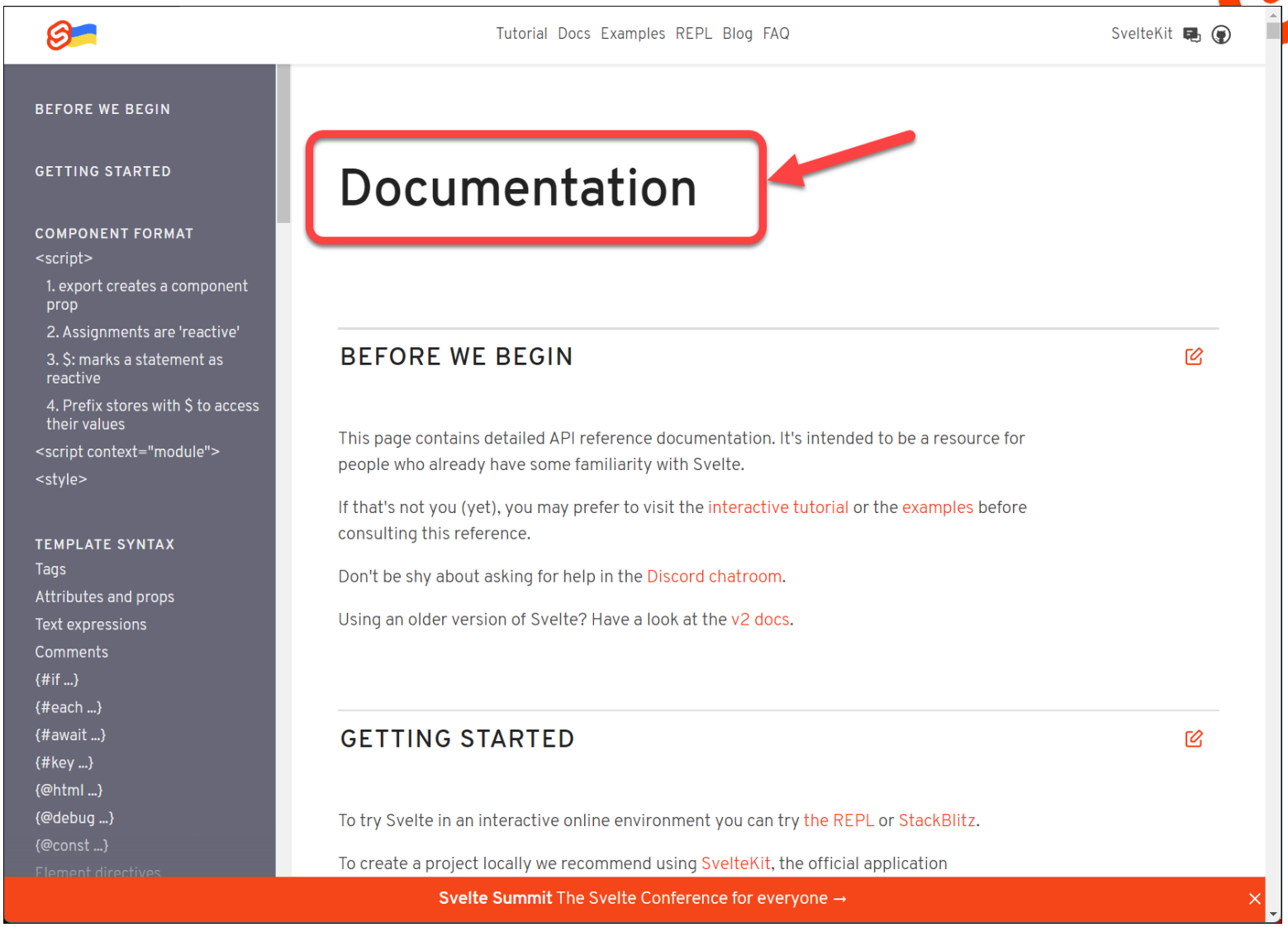
https://daily.dev/blog/building-with-svelte-all-you-need-to-know-before-you-start

Chrome extension, Svelte DevTools



https://chrome.google.com/webstore/detail/svelte-devtools/ckolcbmkjpjmangdbmnkpjigpkddpogn

Don’t forget: Svelte documentation!



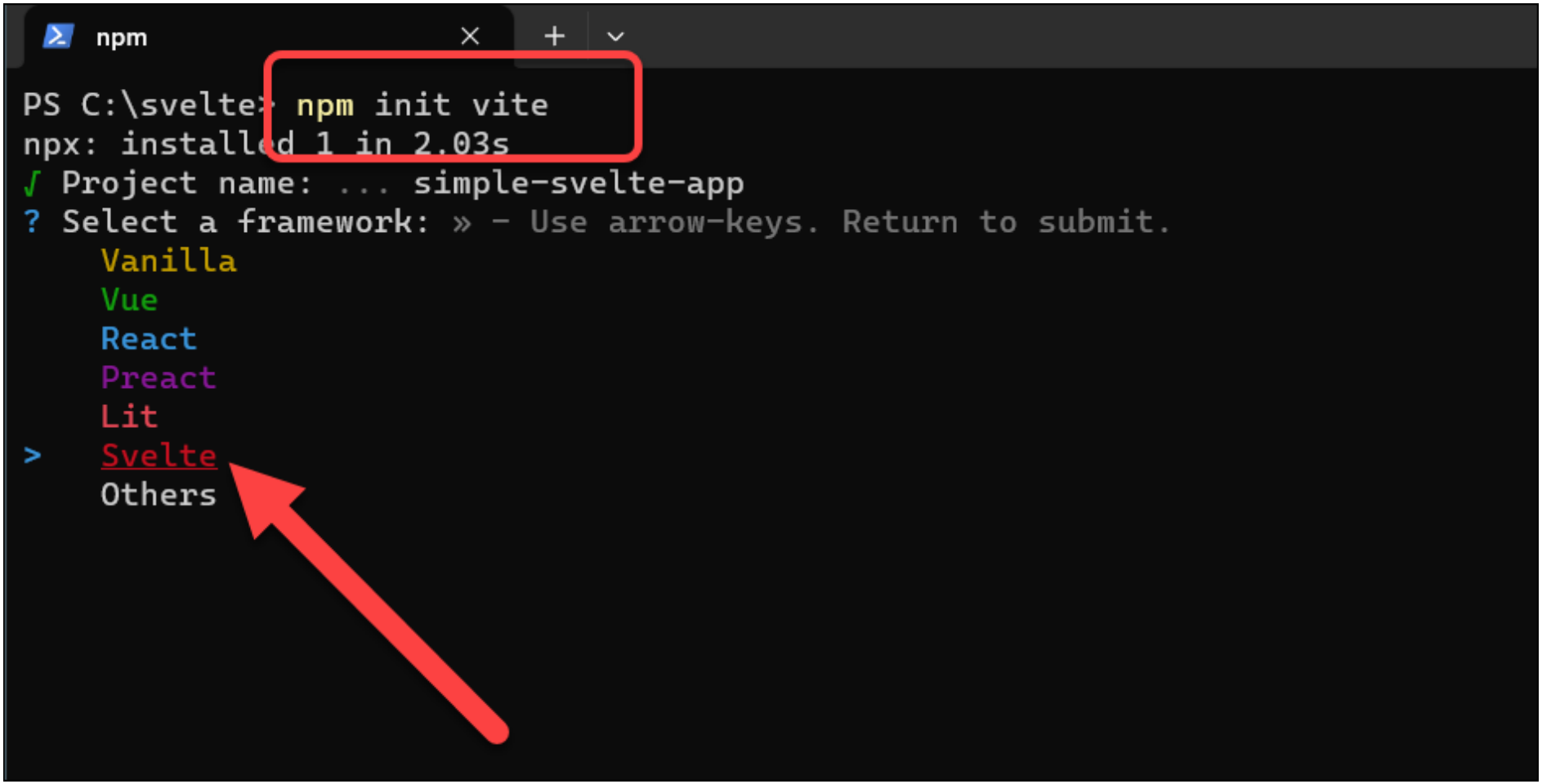
https://svelte.io/docs

Let’s create an app

Creating a non-trivial app

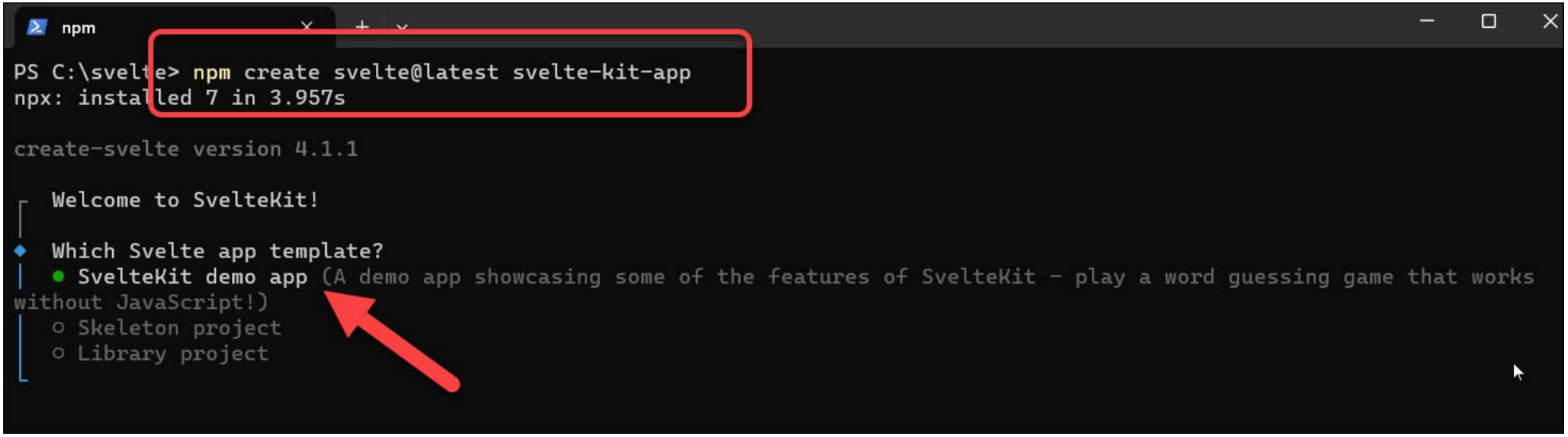
1. Simple Svelte app

`npm init vite` – select `svelte` option



2. Using Svelte KIT

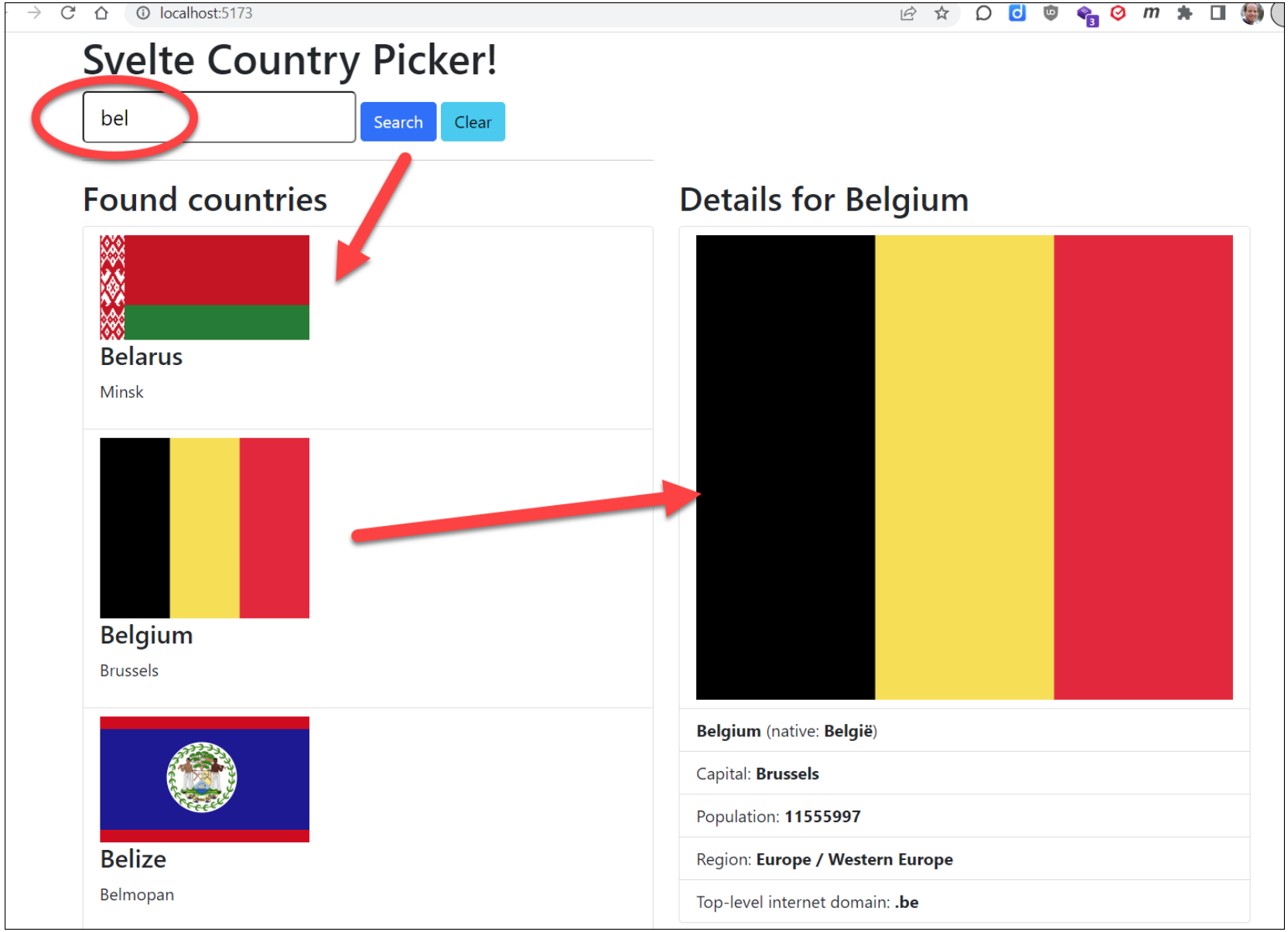
```
npm create @svelte/latest svelte-kit-app
```



3. Non-trivial app

- Show UI
  - Collect form data / user information
- Talk to backend/API
- Process results
- Handle user interaction

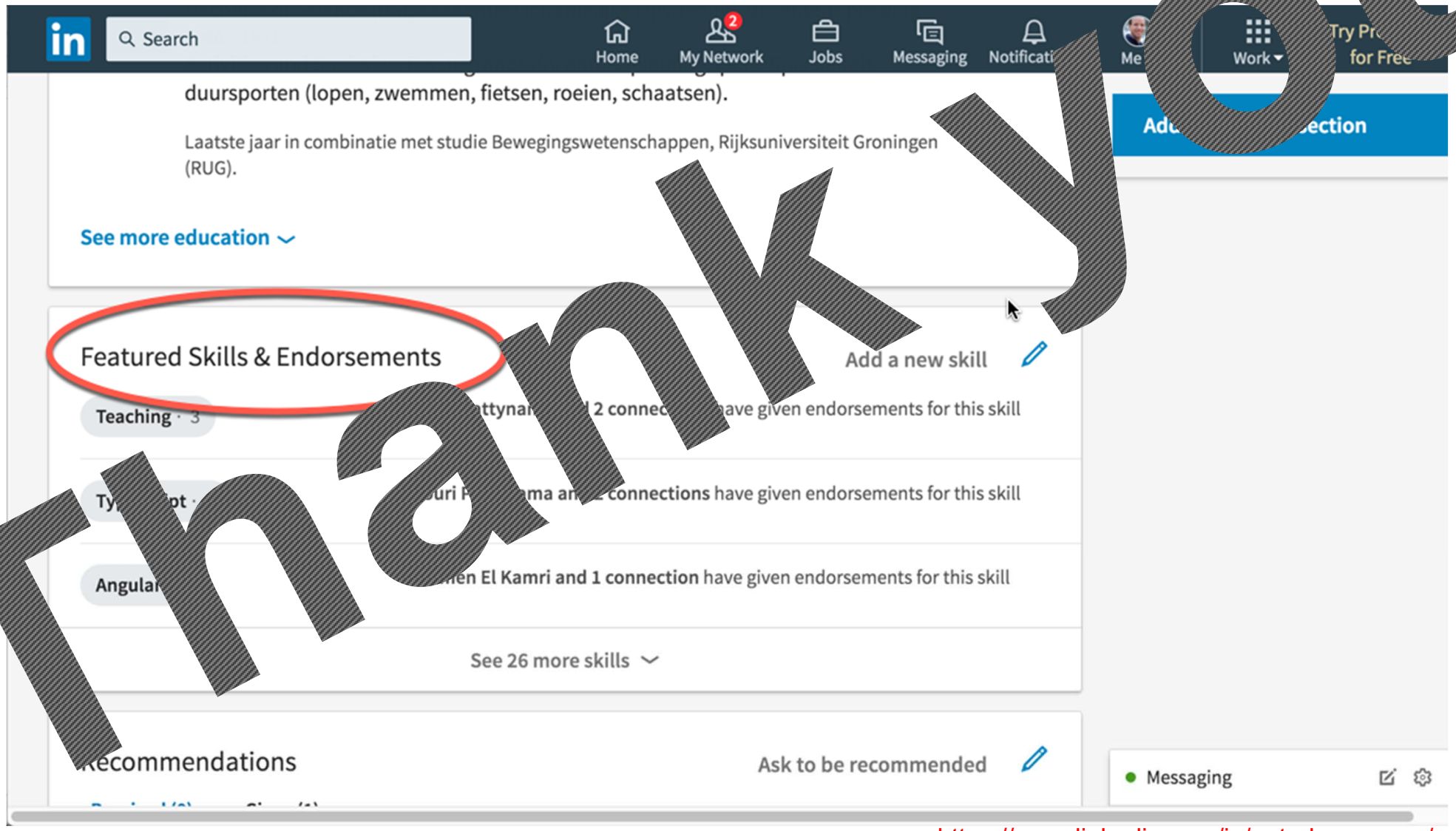
Svelte Country picker



Summary

- Introduction – what is Svelte, comparison
- Svelte tooling – installation
- The structure and architecture of basic Svelte apps.
- Svelte building blocks/concepts

- *Live coding demo* - creating non-trivial applications (time permitting)



Thank you

https://www.linkedin.com/in/peterkassenaar/