



Svelte

Web App Development Reimagined

R. Mark Volkmann

Object Computing, Inc.



<https://objectcomputing.com>



mark@objectcomputing.com



@mark_volkmann

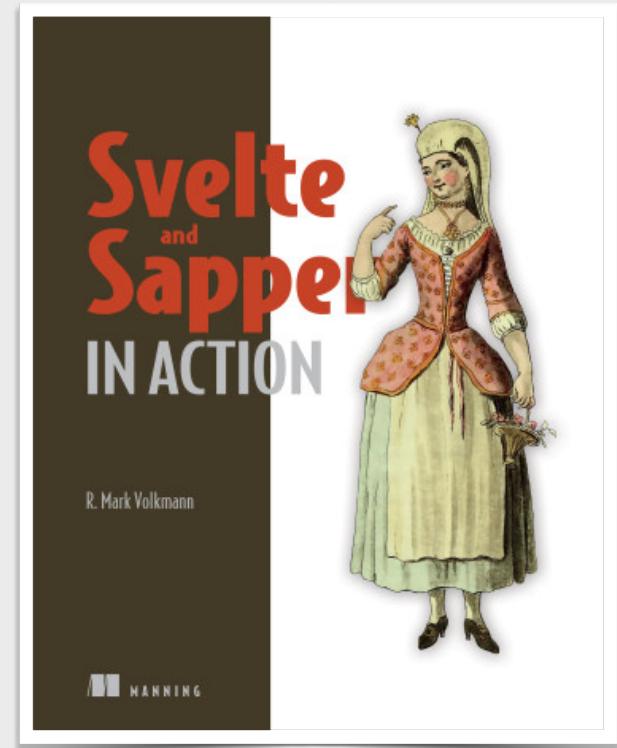


OBJECT COMPUTING
YOUR OUTCOMES ENGINEERED

Slides at <https://github.com/mvolkmann/talks/...svelte>

About Me

- Partner and Distinguished Software Engineer at Object Computing, Inc. in St. Louis, Missouri USA
- 42 years of professional software development experience
- Writer and teacher
- Blog at <https://mvolkmann.github.io/blog/>
- Author of Manning book “Svelte and Sapper in Action”
 - Sapper has been replaced by SvelteKit which has many of the same features
 - Much less expensive to buy directly through Manning when there is a sale rather than through Amazon
- Recently doing a lot of SwiftUI development



Object Computing, Inc.

- Provides software consulting and development
- Uses Svelte for nearly all web apps
- Almost always hiring ... talk to me later
- <https://objectcomputing.com/careers>



Svelte is ...

- An alternative to React, Vue, Angular, ...
- A compiler, not a runtime library
- Can use JavaScript or TypeScript
- Developed by Rich Harris
 - also created of Ractive and Rollup
 - worked in graphics departments at “The Guardian” and “The New York Times”
 - now at Vercel developing Svelte full-time along with Simon H. (@dummdidumm_) at Vercel and many contributors outside Vercel



<https://svelte.dev>

Being a Compiler

- Enables many things that cannot easily be done in other frameworks
 - eliminate use of virtual DOM
 - <https://svelte.dev/blog/virtual-dom-is-pure-overhead>
 - detect where component state and app state is used
 - so those parts of the DOM can be efficiently rebuilt when values change
 - support custom syntaxes
 - detect unused CSS
 - generate optimized JavaScript from `.svelte` files with no runtime library
 - only includes parts of Svelte framework that are used
- Can't just copy ideas from Svelte into other frameworks

Benefits



- Less code to write
- File-based component definitions - JS, HTML, and CSS
- CSS scoped by default
- Easy component state management - **reactivity**
- Easy app state management - **stores**
- Reactive statements (`$:`) - like spreadsheet formulas
- Two-way data bindings - ex. between variable and `<input>`
- Built-in animations
- Small bundle sizes

2021 State of JS Survey

- <https://2021.stateofjs.com/en-US/libraries/front-end-frameworks>
- Results from 2019, 2020, and 2021

Angular satisfaction - 38% 42% 45%

React satisfaction - 89% 88% 84%

Svelte satisfaction - 88% 89% 90%

Vue satisfaction - 87% 85% 80%

Svelte interest - 67% 66% 68%

Svelte Usage - 8% 15% 20%

Svelte awareness - 75% 87% 94%



.svelte File Contents

All parts are optional.

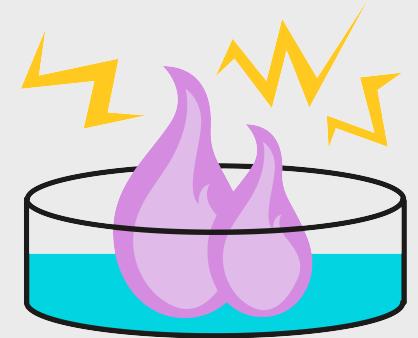
```
<script context="module">  
    think of as class-level; can export functions and constants; not often used  
</script>  
-----  
<script>    think of as instance-level; automatically exports component; cannot export anything else  
        export let name;    declares a prop to be passed in  
</script>    variables and functions are defined here  
-----  
<h1>Hello, {name}!</h1>    HTML can include Svelte logic and interpolations of expressions  
-----  
<style>  
    h1 {  
        color: red;    automatically scoped to this component  
    }  
</style>
```

Component State / Reactivity

- Just variables declared at top-level of `script` element
- Changes trigger updates to parts of DOM where used

```
<script>
  let count = 0;
</script>

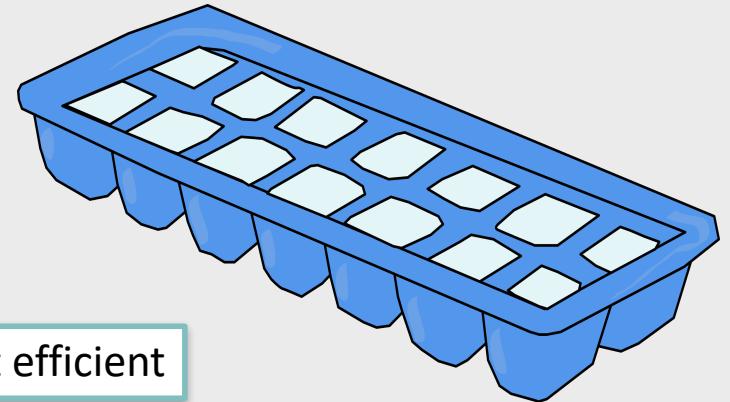
<div>
  <button disabled={count <= 0} on:click={() => count--}>-</button>
  <h1>{count}</h1>
  <button on:click={() => count++}>+</button>
</div>
```



- How much code would be required to implement this in other frameworks?

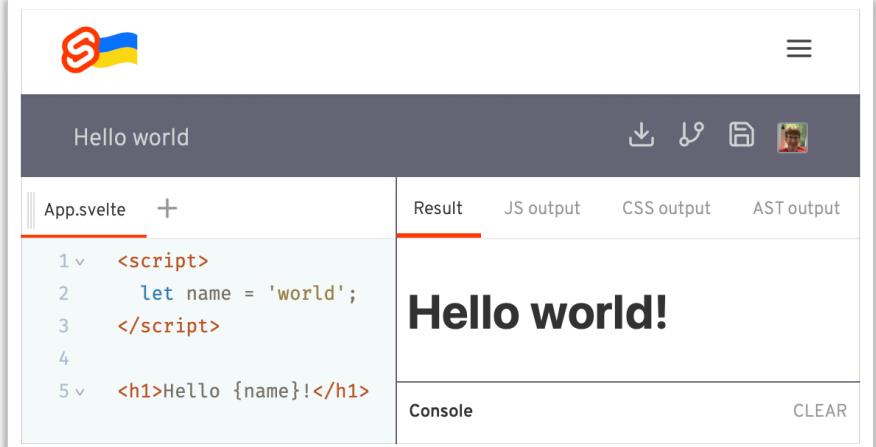
Reactivity of Arrays

- Changing variable “value” causes DOM updates
- But adding, removing, and changing array elements does not change array value
 - still refers to same array
- Three ways to handle
 - `myArr = myArr.concat(newValue);`
 - `myArr = [...myArr, newValue];`
 - `myArr.push(newValue); myArr = myArr;` most efficient



Svelte REPL

- Online Read Eval Print Loop at <https://svelte.dev/repl/>
- Write and run Svelte apps without installing anything
- View generated JS and CSS
- Save to recall later
- Export to continue development outside REPL
- See my examples at <https://mvolkmann.github.io/blog/> ... Svelte ... REPLS
- Demo Counter app in REPL



The screenshot shows the Svelte REPL interface. At the top, there's a logo with a stylized 'S' and a flag. Below it, the title 'Hello world' is displayed. On the left, the code editor shows 'App.svelte' with the following content:

```
1 <script>
2   let name = 'world';
3 </script>
4
5 <h1>Hello {name}!</h1>
```

On the right, the 'Result' tab is selected, showing the output 'Hello world!'. Other tabs include 'JS output', 'CSS output', and 'AST output'. At the bottom, there are buttons for 'Console' and 'CLEAR'.

Reactive Statements

- Svelte interprets the JS label \$ to be a “reactive statement”
- Similar to a spreadsheet formula - ex. =B3+C3
- Code is re-executed every time a referenced variable changes
- Examples
 - `$: total = scores.reduce((acc, s) => acc + s, 0);`
 - `$: console.log('total =', total);`
 - `$: evaluateCart(cart, taxRate);`
 - `$: { ... } block of code`

Loan Calculator

- Calculates monthly payment from loan amount, interest rate, and number of years
- Great demo of using reactive statements
- Review code in REPL
- How much code would be required to implement this in other frameworks?

The image shows a screenshot of a loan calculator application. It features three input fields with spinners for adjusting values: 'Loan Amount' (set to 200000), 'Interest Rate' (set to 3), and 'Years' (set to 30). Below these inputs, the calculated 'Monthly Payment' is displayed as \$843.21.

Input	Value
Loan Amount	200000
Interest Rate	3
Years	30
Monthly Payment:	\$843.21

Conditional Logic in HTML

- Uses Mustache-like syntax
- `{#if some-condition}` opens with #
 HTML to render
- `{:else if other-condition}` continues with :
 other HTML to render
- `{:else}`
 more HTML to render
- ends with /

Iteration in HTML

- `{#each arrExpr as elemName, index (keyExpr)}`
 - HTML to render for each element
- `{:else}`
 - HTML to render if array is empty
- `{/each}`
- , `index` is optional
- `(keyExpr)` is optional
 - include when array elements will be added, deleted, or reordered
- `{:else}` block is optional

Animations

- Supported by four packages
 - `svelte/animate` - `flip` function animates changes to x/y position from old to new
 - `svelte/easing` - easing functions controls rate of change through an animation
 - `svelte/motion` - `spring` and `tweeted` functions return a `writable` store used to animate changes to a value
 - `svelte/transition` - many directives and `crossfade` function directives include `blur`, `draw`, `fade`, `fly`, `scale`, and `slide`
- All are CSS-based rather than JS-based
 - good performance because main thread is not blocked
- Can define custom transitions

Animation Demos

- See <https://mvolkmann.github.io/blog/...> Svelte ... REPLS
- Transition Animations →
- Toast →
- Pie Chart (svelte/motion) →
- Crossfade Demo →
- Flip Animation →
- Custom Transition (spin) →
- Draw Animation →

Events

- Events go from child component to parent
- Events have a name and optional data
- To dispatch from child
- To listen in parent

```
<script>  
  import {createEventDispatcher} from 'svelte';  
  const dispatch = createEventDispatcher();  
  dispatch('my-event', someData);  
</script>
```

MyChild.svelte

typically called when some user interaction occurs

```
<script>  
  import MyChild from './MyChild.svelte';  
  function handleMyEvent(event) {  
    const data = event.detail;  
    ...  
  }  
</script>  
<MyChild on:my-event={handleMyEvent}>
```

MyParent.svelte

Demonstrated in TODO app ahead

if no handler function is specified, the event propagates up

Scoped CSS

- To achieve CSS scoping, Svelte ...
 - computes hash of all CSS in component
 - adds CSS class with name **svelte-hash** to all elements rendered by the component that are targeted by CSS
 - ex. `<button>` becomes
`<button class="svelte-4vhgyu">`
 - adds same generated CSS class name to all CSS rules
 - ex. `button { ... }` becomes
`button.svelte-4vhgyu { ... }`

You don't need to know this, but it is what happens behind the scenes.

Global CSS

- Can define in a `.css` file and import into top component `.svelte` file
- Can avoid scoping in `<style>` of a component using`:global(selector) { properties }`
 - typically used to style nested components

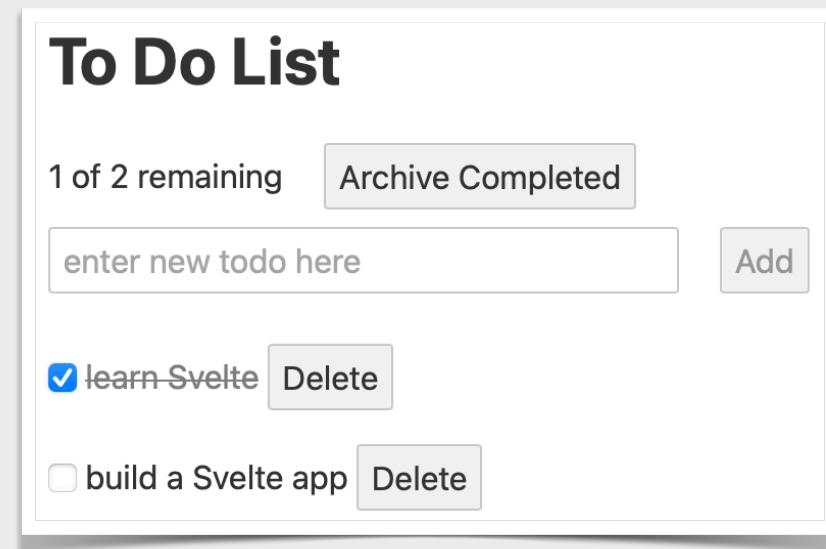
```
<div class="my-component">
  <OtherComponent />
</div>

<style>
  .my-component :global(.other-component h1) {
    color: red;
  }
</style>
```

goal here is to change `h1` styling only for those rendered by `OtherComponent`

Todo App

- Classic app that demonstrates many Svelte features including
 - importing components
 - passing props
 - interpolation and `#each` in HTML
 - two-way data binding
 - reactivity
 - events
 - animation
 - scoped CSS
- Review code in REPL



Actions & Slots

- **Actions** are functions that are called when an element is added to the DOM
 - often used to modify DOM after an element appears
 - can also specify code to run when element is removed from DOM
- **Slots** allow parent components to pass content to child components
 - unnamed (default) and named slots
- Not related to each other,
but both are demonstrated next

Actions & Slots Example

```
<script>
  import Box from './Box.svelte';
  let showBox = false
</script>

<div>
  <input type="checkbox" bind:checked={showBox} />
  Show Box
</div>

{#if showBox}
  <Box bgColor="red" fgColor="yellow">
    Some <i>italic</i> text
  </Box>
{/if}
```



App.svelte

```
<script>
  export let bgColor; // applied with an action
  export let fgColor = 'white'; // applied with style attribute

  function setBgColor(element) {
    console.log("added to DOM");
    element.style.backgroundColor = bgColor;
    return {
      destroy() {
        console.log("removed from DOM");
      }
    };
  }
</script>

<!-- Shows two approaches to applying style-related props. -->
<div class="box" style="color: {fgColor}" use:setBgColor>
  <slot /> <!-- content goes here -->
</div>

<style>
  .box {
    display: inline-block;
    padding: 1rem;
  }
</style>
```

Box.svelte

see “Action and Slot Demo” REPL

App State / Stores

- Stores use publish/subscribe to share data between components
- Four kinds: **writable**, **readable**, **derived**, and **custom**
- Easiest approach is to define and export all in **stores.js**
- Can import in any components
- Subscribing and unsubscribing is automatic
when store names are preceded by \$

Writable Store Example

```
import { writable } from 'svelte/store';
export const user = writable({
  firstName: '',
  lastName: ''
});
```

stores.js

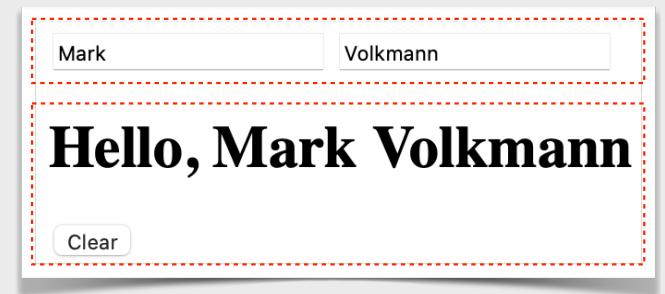
How much code would be required to implement this in other frameworks?

```
<script>
  import { user } from '../stores.js';
  import Report from '../Report.svelte';
</script>
<input placeholder="First Name" bind:value={$user.firstName} />
<input placeholder="Last Name" bind:value={$user.lastName} />
<Report />
```

routes/index.svelte

```
<script>
  import { user } from './stores.js';
  function clear() {
    $user = {firstName: '', lastName: ''};
  }
</script>
<h1>Hello, {$user.firstName} {$user.lastName}.</h1>
<button on:click={clear}>Clear</button>
```

Report.svelte



see "Writable Store" REPL

Readable Stores

- Cannot be changed from outside, but can change themselves
- Example
 - holds an array of objects describing dogs that come from a REST API
 - updated every five minutes

```
import {readable} from 'svelte/store';           stores.js

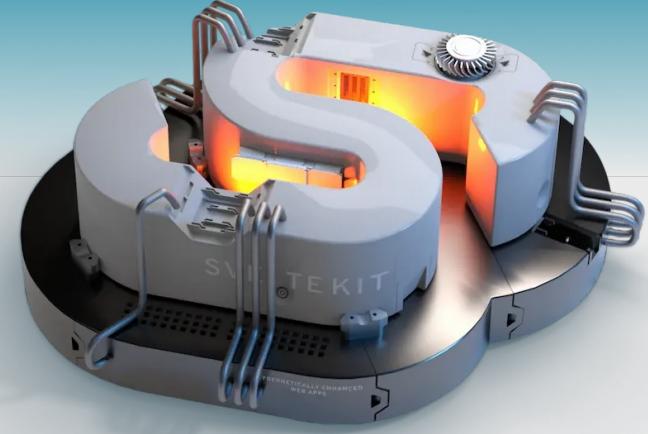
export const dogStore = readable([], async set => {
    const token = setInterval(() => {
        const res = await fetch('/dogs');
        const dogs = await res.json();
        set(dogs);
    }, 5 * 60 * 1000); // every 5 minutes

    return () => clearInterval(token);           ← called when there are
});                                              no more subscribers
```

Component Communication Options

Need	Solution
parent passes data to child	props
parent passes HTML and components to child	slots
child notifies parent, optionally including data	events
ancestor makes static data available to descendants	context
component shares data between all instances	module context
any component subscribes to and publishes data	stores

SvelteKit



- Framework on top of Svelte that replaces Sapper
- Like Next for React and Nuxt for Vue
- Features
 - file-based page routing
 - file-based endpoints (APIs)
 - layouts - common header/footer/nav
 - error page
 - code splitting for JS and CSS
 - prefetching based on hover and focus
 - static sites and pages
 - hot module reloading using Vite
 - TypeScript, ESLint, and Prettier setup
 - adapters for deployment targets
 - official: static, Node, Netlify, Vercel
 - community: Begin, Cloudflare Workers, Deno, Firebase, and more

<https://kit.svelte.dev>

Creating a New Project

- `npm init svelte@next project-name`
- Answer questions
 - Which Svelte app **template**?
 - SvelteKit demo app or Skeleton project (preferred)
 - Use **TypeScript**?
 - Add **ESLint** for code linting?
 - Add **Prettier** for code formatting?
 - Add **Playwright** for browser testing?

alternative to Cypress

```
[I] volkmannm@OCI1689MBP15INCH ~/D/p/svelte> npm init svelte@next demo
create-svelte version 2.0.0-next.124
Welcome to SvelteKit!
This is beta software; expect bugs and missing features.
Problems? Open an issue on https://github.com/sveltejs/kit/issues if none exists already.

✓ Which Svelte app template? > Skeleton project
✓ Use TypeScript? ... No / Yes
✓ Add ESLint for code linting? ... No / Yes
✓ Add Prettier for code formatting? ... No / Yes
✓ Add Playwright for browser testing? ... No / Yes

Your project is ready!
✓ Typescript
  Inside Svelte components, use <script lang="ts">
✓ ESLint
  https://github.com/sveltejs/eslint-plugin-svelte3
✓ Prettier
  https://prettier.io/docs/en/options.html
  https://github.com/sveltejs/prettier-plugin-svelte#options

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

Next steps:
1: cd demo
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
```

Running a Project

- Follow instructions that are output to install dependencies and run locally
 - `cd project-name`
 - `npm install`
 - `npm run dev -- --open`

Welcome to SvelteKit

Visit kit.svelte.dev to read the documentation

SvelteKit Page Routing

- Pages and their URLs are described by directory and file names under `src/routes`
- File and directory names inside square brackets indicate that a path parameter will be captured

Page Routing Example

```
<script>  routes/index.svelte
  import '../global.css';
</script>
<nav>
  <a href="/page1">Page 1</a>
  <a href="/page2">Page 2</a>
</nav>
<main>
  <h1>Home Page</h1>
</main>
<style>
  a {
    color: white;
    font-size: 2rem;
    text-decoration: none;
  }
  main {
    padding: 1rem;
  }
  nav {
    display: flex;
    gap: 1rem;
    background-color: orange;
    padding: 1rem;
  }
</style>
```

```
body {           global.css
  font-family: sans-serif;
  margin: 0;
}

routes/page1.svelte
<h1>First Page</h1>

routes/page2.svelte
<h1>Second Page</h1>
```

Page 1 Page 2
Home Page

Wrap Up

- Developers like Svelte for many reasons, but the biggest reasons are that it reduces the amount of code that must be written and the code is easy to understand
- Delivering an excellent developer experience (DX) is not at the exclusion of good user experience (UX)
- Give Svelte a try and see if it simplifies web development for you!

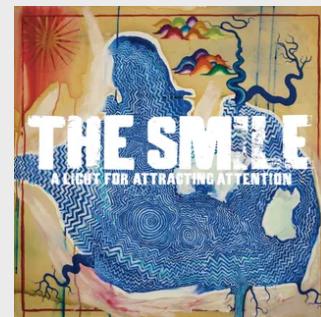


Music Recommendations

- **Florist -**
Florist



- **The Smile -**
A Light For
Attracting Attention



- **Regina Spektor -**
Home, Before and After



- **Julia Jacklin -**
Pre Pleasure

