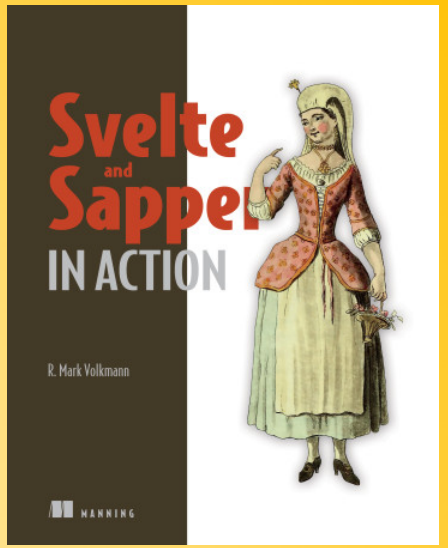


Svelte - Web App Development Reimagined

slides at https://github.com/mvolkmann/talks

R. Mark Volkmann
Object Computing, Inc.
http://objectcomputing.com
Email: mark@objectcomputing.com
Twitter: @mark_volkmann
GitHub: mvolkmann

OCI | TRAINING

What Is Svelte?

- Alternative to web frameworks like React, Vue, and Angular
- A web application **compiler**, not a runtime library
  - compiles `.svelte` files to a single JavaScript file
  - no Svelte runtime `dependencies`, only `devDependencies`
- Doesn't use a virtual DOM
- Developed by **Rich Harris**
  - formerly at "The Guardian"; currently at "The New York Times"
  - previously created **Ractive** web framework - https://ractive.js.org/
    - used at "The Guardian"
    - inspired parts of Vue
  - created **Rollup** module bundler - https://rollupjs.org/
    - alternative to Webpack and Parcel

What is SvelteKit?

- Framework on top of Svelte that replaces Sapper
- Like Next for React or Nuxt for Vue
- Features
  - file-based page routing
  - file-based endpoints (REST services)
  - layouts
    - ex. common page header, footer, and nav
  - error page
  - code splitting for JS and CSS
    - page visits only load the JS and CSS they need
  - hot module reloading (HMR)
    - provided by Vite; very fast!
  - static pages and sites
  - setup of TypeScript
  - setup of Sass or Less CSS preprocessors
  - setup of ESLint
  - setup of Prettier
  - adapters for deployment targets
    - currently node, static, begin, netlify, and vercel
    - to change, modify **`svelte.config.cjs`**

Important files and directories:
`src/app.html` - starting HTML file
`src/routes` - holds page components and endpoints
`src/lib` - holds other components and functions
`build` - holds files generated by **`npm run build`**

Config files:
`.eslintrc.cjs`
`.prettierignore`
`.prettierrc`
`jsconfig.json`
`package.json`
`svelte.config.cjs`

Creating a SvelteKit Project

- Install Node.js
- **`npm init svelte@next [`*`project-name`*`]`**
  - omit **`project-name`** to create in current directory
  - asks these questions
    - Use TypeScript in components? defaults to no
    - What do you want to use for writing Styles in Svelte components? CSS (default), Less, or SCSS
    - Add ESLint for code listing? defaults to no
    - Add Prettier for code formatting? defaults to no
  - outputs instructions for next steps
- **`cd `*`project-name`***
- **`npm install`**

sets **`useTabs`** to true and **`printWidth`** to 100

.prettierrc ← my preferred settings

```
{
  "arrowParens": "avoid",
  "bracketSpacing": false,
  "printWidth": 80,
  "singleQuote": true,
  "trailingComma": "none",
  "useTabs": false
}
```

.eslintignore ← create this file

```
build/
```

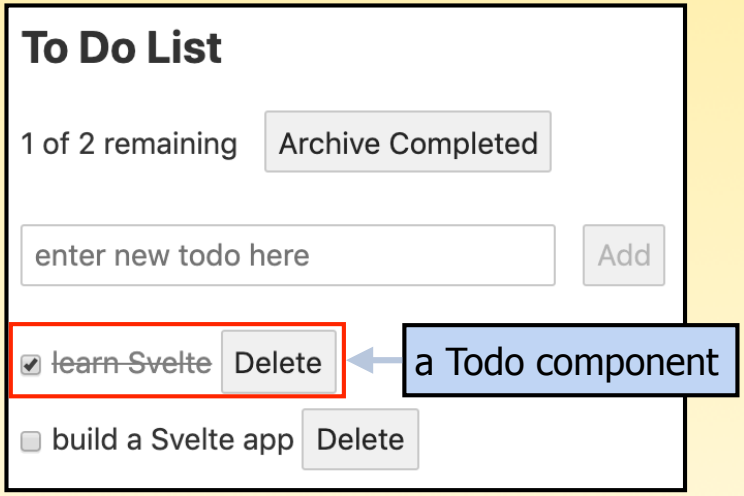
Running a SvelteKit Project

- **`npm run lint`** to run ESLint
- **`npm run format`** to run Prettier
- **`npm run dev`** to run in development mode
  - provides watch and live reload
  - options go after **`--`**
  - to open in default browser add **`--open`** or **`-o`**
  - to specify port add **`--port #`** or **`-p #`** (defaults to **`3000`**)
- **`npm build`** to build for deployment
  - creates files in **`build`** directory that should be deployed

An Example

- Since you are all experienced web developers,
  let's look at an example app
  so you can **compare** the look of the code
  to your current favorite web framework
- On to the classic ... todo app!
- Code at
  - https://github.com/mvolkmann/sveltekit-todo
  - https://github.com/mvolkmann/sveltekit-todo-w-endpoints

Todo App ...


To Do List
1 of 2 remaining
Archive Completed
enter new todo here
Add
learn Svelte
Delete
a Todo component
build a Svelte app
Delete

... Todo App ...

src/lib/Todo.svelte

```
<script>
  import {createEventDispatcher} from 'svelte';
  import {fade} from 'svelte/transition';
  const dispatch = createEventDispatcher();
  export let todo; // the only prop
</script>

<li transition:fade>
  <input
    type="checkbox"
    checked={todo.done}
    on:change={() => dispatch('toggleDone')}
  />
  <span class:done={todo.done}>{todo.text}</span>
  <button on:click={() => dispatch('delete')}>Delete</button>
</li>

<style>
  .done {
    color: gray;
    text-decoration: line-through;
  }
  li {
    margin-top: 5px;
  }
</style>
```

`export` makes it a prop

interpolation

`script` and `style` sections are optional

What is the name of this component? Can't tell.
Names are assigned when other components import this one.

... Todo App ...

src/routes/index.svelte

```
<script>
  import Todo from '../lib/Todo.svelte';

  let lastId = 0;
  const createTodo = (text, done = false) => ({id: ++lastId, text, done});

  let todoText = '';
  let todos = [
    createTodo('learn Svelte', true),
    createTodo('build a Svelte app')
  ];

  $: uncompletedCount = todos.filter(t => !t.done).length;
  $: status = `${uncompletedCount} of ${todos.length} remaining`;

  function addTodo() {
    todos = todos.concat(createTodo(todoText));
    todoText = '';
  }

  const archiveCompleted = () => todos = todos.filter(t => !t.done);

  const deleteTodo = todoId => todos = todos.filter(t => t.id !== todoId);

  function toggleDone(todo) {
    const {id} = todo;
    todos = todos.map(t => t.id === id ? {...t, done: !t.done} : t);
  }
</script>
```

Top-level variables ARE the **component state** if used in HTML! When state changes, only the relevant part of DOM are updated.

reactive declarations

No methods, just functions.

Not really archiving in this simple implementation, just deleting.

No `this` anywhere, just plain functions!

... Todo App

src/routes/index.svelte

```
1  <div>
2    <h2>To Do List</h2>
3    <div>
4      {status}
5      <button on:click={archiveCompleted}>Archive Completed</button>
6    </div>
7    <form on:submit|preventDefault={addTodo}>
8      <input
9        type="text"
10       size="30"
11       autofocus
12       placeholder="enter new todo here"
13       bind:value={todoText}
14     />
15     <button disabled={!todoText}>Add</button>
16   </form>
17   <ul>
18     {#each todos as todo}
19       <Todo
20         {todo}
21         on:delete={() => deleteTodo(todo.id)}
22         on:toggleDone={() => toggleDone(todo)}
23       />
24     {/each}
25   </ul>
26 </div>
```

binds value of form element to a variable;
simulates two-way data binding;
provides current value and
event handling for updating variable
when user changes value

Mustache-style markup

... Todo App ...

src/routes/index.svelte

```
<style>
  button {
    margin-left: 10px;
  }

  h2 {
    margin-top: 0;
  }

  /* This removes bullets from a bulleted list. */
  ul {
    list-style: none;
    margin-left: 0;
    padding-left: 0;
  }
</style>
```

Logic in Markup

- Three approaches for conditional and iteration logic
- **React**
  - uses JSX where logic is implemented with JavaScript code in curly braces
- **Angular** and **Vue**
  - support framework-specific attributes for logic
  - ex. `ngIf`, `ngFor`, `v-if`, `v-for`, …
- **Svelte**
  - supports mustache-like custom syntax that **wraps** elements
  - ex. `{#if}` and `{#each}`
  - can wrap multiple elements without introducing a new, common parent

Why does it make sense to specify conditional and iteration logic INSIDE elements using attributes?

Imagine if you could do that with JavaScript functions.

```
doSomething(
  arg1,
  arg2,
  if (arg1 > 10),
  for (arg1 in someCollection));
```

Isn't that weird?

Top Svelte Features

- It's fast!
  - see https://krausest.github.io/js-framework-benchmark/current.html
  - can select frameworks to compare
- Small bundle sizes
- File-based component definitions
- CSS scoped by default
- Clear place to put global CSS
- Easy component state management (reactivity)
- Reactive statements (`$:`)
- Two-way data bindings
- Built-in animations
- Easy app state management (stores) We haven't seen this yet.

Small Bundle Sizes

- Delivered code is much smaller,
  so loads faster in browsers
- Create production build with **`npm run build`**
- A RealWorld Comparison of Front-End Frameworks with Benchmarks
  - https://medium.com/dailyjs/a-realworld-comparison-of-front-end-frameworks-**2020**-4e50655fe4c1

Gzipped App Size in KBs

Angular+ngrx: 694
React+Redux: 193
Vue: 71
Svelte: 15

Lines of Code

Angular+ngrx: 4210
React+Redux: 2050
Vue: 2076
Svelte: 1057

File-based Component Defs

- **Angular** uses classes
- **React** uses functions or classes
- **Vue** uses object literals
- **Svelte** doesn't use any JavaScript container
  - JavaScript, CSS, and HTML in source files
    are combined to form the component definition
    which automatically becomes the default export
  - name is associated when imported and must start uppercase
  - lowercase names are reserved
    - for predefined elements like those in HTML and SVG

CSS

- Scoped by default
  - CSS specified in a component **`style`** element
    is automatically scoped to the component
  - achieved by adding the same generated CSS class name, **`svelte-`*`hash`***,
    to each rendered element of the component affected by these CSS rules
- Clear place for global CSS
  - in Svelte see **`public/global.css`**; linked by **`public/index.html`**
  - in SvelteKit see **`src/app.css`**; imported by **`src/routes/$layout.svelte`**

Easy Component State Mgmt.

("reactivity")

- Changes to top-level variables referenced in interpolations automatically cause those interpolations to be reevaluated
- Example

```
<script>
  let count = 0;
  const increment = () => count++;
</script>

<div>count = {count}</div>
<button on:click={increment}>+</button>
```

- Must assign a new value to trigger
  - pushing new elements onto an array doesn't do this

```
myArr = myArr.concat(newValue);
```
works

```
myArr = [...myArr, newValue];
```
works

```
// Alternative trick
myArr.push(newValue);
myArr = myArr;
```
works

Reactive Statements

a.k.a. "destiny operator"

- `$:` is a "labeled statement" with label name "`$`" that Svelte treats as a "reactive statement"

Labeled statements can be used as targets of `break` and `continue` statements.
It is not an error in JavaScript to use same label more than once in same scope.

- Add as a prefix on top-level statements that should be repeated whenever any referenced variables change
- Examples

```
$: average = total / count;
$: console.log('count =', count);
```

like "computed properties" in Vue

great for debugging

When applied to an assignment to an undeclared variable it is called a "reactive declaration" and the `let` keyword is not allowed.

- Can apply to a block

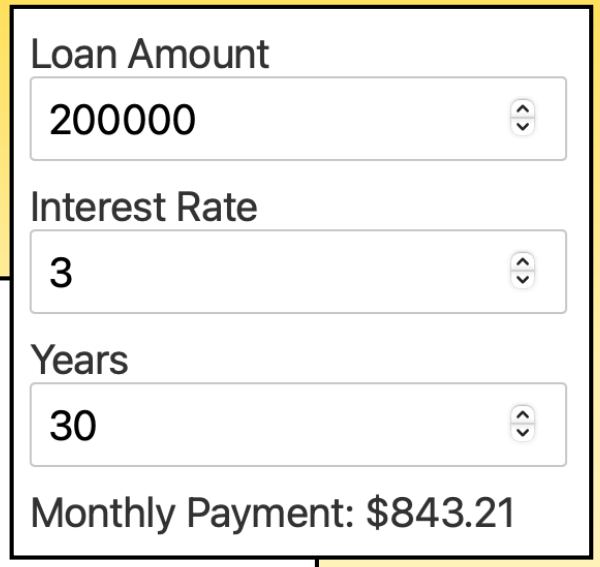
```
$: {
  // statements to repeat go here
}
```

- Can apply to multiline statements like `if` statements

```
$: if (someCondition) {
  // body statements
}
```

re-evaluates condition if any variables it references change, and executes body only when true

Loan Example




```
<script>
  let interestRate = 3;
  let loanAmount = 200000;
  let years = 30;
  const MONTHS_PER_YEAR = 12;

  $: months = years * MONTHS_PER_YEAR;
  $: monthlyInterestRate = interestRate / 100 / MONTHS_PER_YEAR;
  $: numerator = loanAmount * monthlyInterestRate;
  $: denominator = 1 - (1 + monthlyInterestRate) ** -months;
  $: payment =
    !loanAmount || !months ? 0 :
    interestRate ? numerator / denominator :
    loanAmount / months; // no interest
</script>

<label for="loan">Loan Amount</label>
<input id="loan" type="number" bind:value={loanAmount} />

<label for="interest">Interest Rate</label>
<input id="interest" type="number" bind:value={interestRate} />

<label for="years">Years</label>
<input id="years" type="number" bind:value={years} />

<div>
  Monthly Payment: ${payment.toFixed(2)}
</div>
```

Easy App State Mgmt.

- "Stores" hold application state outside any component
- Alternative to using props or "context"
  to make data available in components
- Where to define?
  - for stores that should be available to any component,
    define and export them in a file like `src/stores.js`
    and import them from that file wherever needed
  - for stores that should only be available to descendants of a given component,
    define them in that component and
    pass them to descendants using props or context

Kinds of Stores

- **Writable**
  - only kind that can be modified by components
  - methods
    - `set(newValue)`
    - `update(currentValue => newValue)` calculates new value from current value
- **Readable**
  - handle computing their data, perhaps from a REST call
  - components cannot modify
- **Derived**
  - derive data from current values of other stores
- **Custom**
  - must implement `subscribe` method
  - can provide custom methods to update state and not expose `set` and `update` methods

Defining Writable Stores

stores.js

```
import {writable} from 'svelte/store';

export const dogStore = writable([]);
```

initial value

```
export const fancyStore = writable(
  initialValue,
  async set => {
    // Called when subscriber count goes from 0 to 1.
    // Compute initial value and pass to set function.
    const res = await fetch('/some/url');
    const data = await res.json();
    set(data);

    return () => {
      // Place cleanup code here.
      // Called when subscriber count goes to 0.
    }
  }
);
```

using optional second argument

add `try/catch` for error handling

Using Stores

- Option #1 - **subscribe** method - very verbose
- Option #2 - **$** auto-subscription shorthand - much better
  - variables whose names begin with **$** must be stores
  - automatically subscribes when first used
    and unsubscribes when removed from DOM

uses **subscribe** method

```
<script>
  import {onDestroy} from 'svelte';
  import {dogStore} from './stores.js';
  let dogs;
  const unsubscribe = dogStore.subscribe(value => dogs = value);
  onDestroy(unsubscribe);
</script>

<!-- Use dogs in HTML. -->
```

uses auto-subscription

```
<script>
  import {dogStore} from './stores.js';
</script>

<!-- Use $dogStore in HTML. -->
```

Issues to Consider

- **Popularity**
  - perhaps Svelte is now considered the #4 most popular approach for building web apps
  - isn't yet easy to find developers that already know it
  - but it's very easy to learn and there is less to learn than other approaches
- **Component libraries**
  - fewer available than for other frameworks, but perhaps enough for your app
  - just a matter of time for more to arrive
- **Cannot generate HTML in functions**
  - encourages creating additional `.svelte` files in cases where React would use functions that return JSX

Related Tools

- **Svelte VS Code extension**
- **SvelteKit** - https://kit.svelte.dev
  - "a framework for building web applications of all sizes,
    with a beautiful development experience and flexible filesystem-based routing"
  - provides routing, server-side rendering, code splitting, and building static sites
  - uses Vite "Next Generation Frontend Tooling" which provides
    "instant server start", "lightning fast HMR", and "optimized builds"
- **Svelte Testing Library**
  - https://testing-library.com/docs/svelte-testing-library/intro/
- **Storybook** with Svelte
  - https://storybook.js.org/docs/svelte/get-started/introduction
  - https://mvolkmann.github.io/blog/topics/#/blog/svelte/storybook/
- **Svelte Native** - https://svelte-native.technology/
  - for implementing native mobile apps
  - builds on top of NativeScript
  - community-driven project

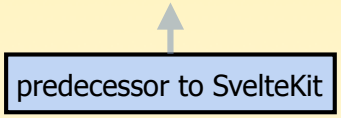
Topics Not Covered Here

but covered in my book

- Two-way data bindings
  - more options than shown here
- Easy animations built-in
- Inserting HTML
- Slots
  - for passing child elements to a component
- Event details
  - handling, modifiers, dispatching
- Lifecycle functions
  - `onMount`, `beforeUpdate`, `afterUpdate`, and `onDestroy`
- Actions
  - register a function to be called when a specific element is added to DOM
  - ex. moving focus
- Routing
  - use SvelteKit file-based routing or `page` on npm
- Context
  - to share static data with descendant components
- Module Context
  - to run JavaScript code in a component source file only once instead of once for each component instance created
- Special Elements
  - `<svelte:name ...>`
- Debugging with `{@debug}`
  - debugger breaks on state changes
- Unit tests
  - with Jest and Svelte Testing Library
- End-to-end tests
  - with Cypress
- Compiling to custom elements
  - can be used with any framework

Svelte Resources

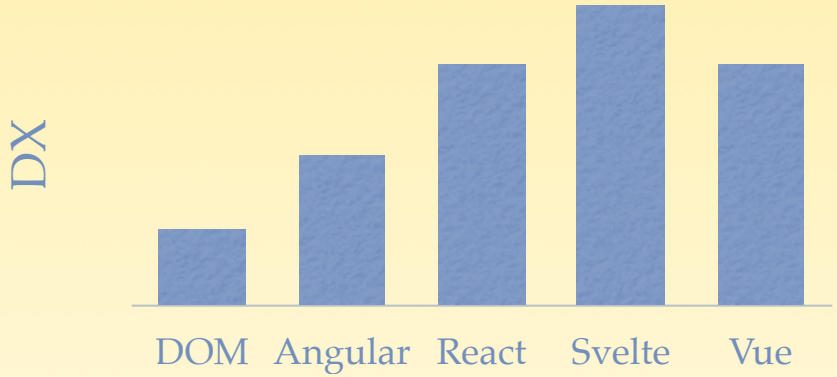
- **"Rethinking Reactivity" talk by Rich Harris**
  - delivered multiple times, most recently at "Shift Conference" June 20, 2019
  - explains issues with using virtual DOM (like React and Vue) and motivation for Svelte
- **Home page** - https://svelte.dev
  - contains **Tutorial**, **API Docs**, **Examples**, online **REPL**, **Blog**, and **Sapper** link
    - REPL is great for trying small amounts of Svelte code
    - REPL can save for sharing and submitting issues



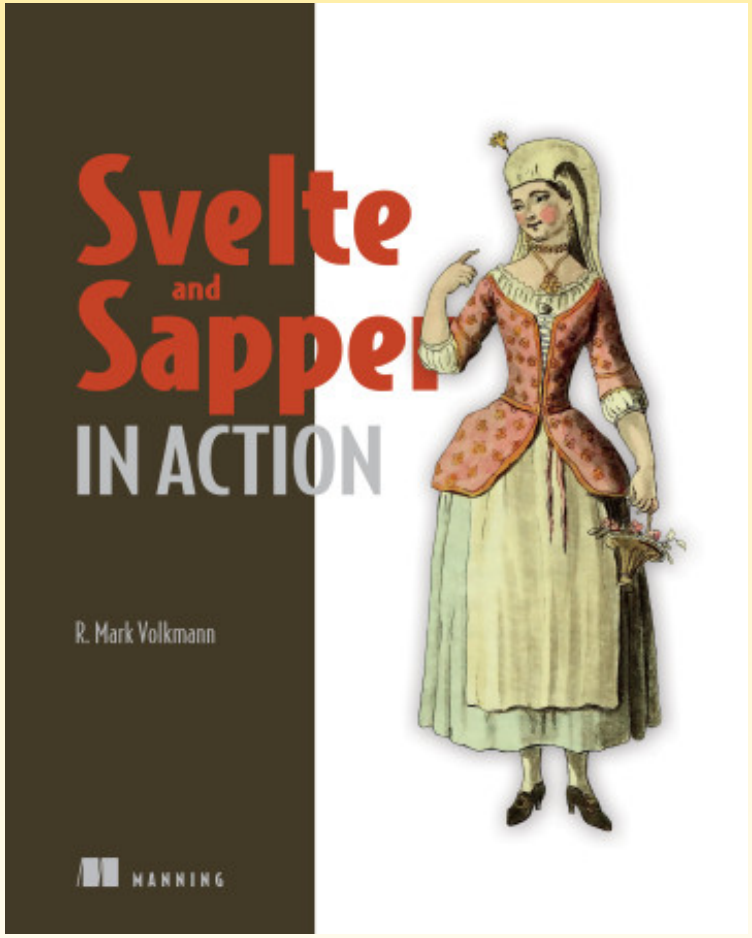
- **SvelteKit** - https://kit.svelte.dev
- **GitHub** - https://github.com/sveltejs/svelte
- **Svelte Community** - https://github.com/sveltejs/community
  - "contains data for Svelte meetups, packages, resources, recipes, and showcase websites"
- **Discord chat room** - https://discordapp.com/invite/yy75DKs

Conclusion

- **Svelte is a worthy alternative** to React, Vue, and Angular



UX is similar for all,
but built-in animations in Svelte may encourage their use.



- Check out **my book**
  - https://www.manning.com/books/svelte-and-sapper-in-action