#1: Getting Started

Links to **REPL examples** can be found at https://mvolkmann.github.io/blog/topics/#/blog/svelte/repls/.

What is Svelte? ...

- Alternative to other web frameworks
  - like React, Vue, and Angular
- Web application compiler, not a runtime library
- Doesn’t use virtual DOM
  - https://svelte.dev/blog/virtual-dom-is-pure-overhead
- Developed by Rich Harris
  - “The Guardian” and “The New York Times”
  - created Ractive and Rollup

Svelte Requirements

- Need to know fundamentals - HTML, CSS, & JS/TS
  - can use JavaScript or TypeScript
- Need to install Node.js
  - we'll see how later

Svelte Pros

- Fast
- Small bundle sizes often half size of other frameworks
- File-based component definitions
- CSS scoped by default
- Easy component state management (reactivity)
- Reactive statements (`?:`)
- Two-way data bindings
- Easy app state management (stores)
- Built-in animations

Svelte Cons

- Not yet as popular as alternatives
  - fewer developers, libraries, and other resources
- Can’t generate HTML in JS functions
  - in React, functions can return JSX
  - Svelte encourages creating more components

Svelte Compared to Other Options

- Svelte requires less code than Angular, React, and Vue to accomplish the same things
- Angular has a steep learning curve
- React uses JSX, hooks, and optionally Redux
- React and Vue use a virtual DOM
  - results in larger bundle sizes
  - slower than Svelte approach of using compiler-generated code to update DOM

Selected Svelte Resources

- Home page - https://svelte.dev

- “Rethinking Reactivity” talk by Rich Harris
- “Svelte and Sapper in Action” book

What is SvelteNative?

- Combines Svelte and NativeScript to build native mobile applications (Android and iOS)
- Thin layer over NativeScript API
- Renders native components, not web views
- Plugins allow access to all native device APIs
- Similar to ReactNative for React
- https://svelte-native.technology

What is Sapper?

- Framework built on Svelte
  - similar to Next for React or Nuxt for Vue
- Being replaced by SvelteKit
  - provides same features and more
- https://sapper.svelte.dev

What is SvelteKit? ...

- Replacement for Sapper
- https://kit.svelte.dev
- Provides
  - **`svelte-kit`** CLI tool
    - commands include **`build`**, **`dev`**, **`package`**, and **`preview`**
  - file-based page routing
  - file-based endpoints (REST services)
  - layouts
  - error page

npm scripts in **`package.json`** uses these

can provide a common header, footer, and nav for a set of pages

… What is SvelteKit?

- And provides
  - code splitting for JS and CSS
    - page visits only load JS and CSS they need
  - fast hot module reloading (HMR) provided by Vite https://vitejs.dev
  - Server-Side Rendering (SSR) for first page visited by default
    - remaining pages rendered in browser
  - support for static sites and individual static pages
    - generated at build time
  - adapters for specific deployment targets
    - deno, netlify, node, static, vercel, and more

Vite

- Build tool
  - alternative to webpack, parcel, and Snowpack
  - uses **esbuild** to prebundle dependencies - https://esbuild.github.io
    - "extremely fast JavaScript bundler" implemented in Go
  - uses **Rollup** to bundle app code - https://rollupjs.org
    - also a module bundler
- Development server
  - provides fast hot module reloading (HMR)
  - uses WebSockets
- https://vitejs.dev

"The Vite team is planning to shift completely to esbuild in the future."

Svelte REPL

- Stands for **R**ead **E**valuate **P**rint **L**oop
- Great for experimenting
- Write and run Svelte apps without installing anything
- Save apps to recall later - requires GitHub account
- Export apps to continue development outside REPL
- Limited set of libraries can be imported
- Cannot create/edit CSS files
- Can view generated JS and CSS

REPL Demonstration

- Let's write some code in the REPL!
  - create **`Counter`** component
  - create **`Greet`** component
- Review "Todo App" REPL

Page Routing Option #1

- Manual routing
  - use variable **`component`** to hold current page component
  - render a component specified in a variable with
    `<svelte:component this={component} `*`props`*` />`
  - use event handling to change **`component`**
    - ex. button or link click
  - URL in browser address bar never changes,
    so can't bookmark pages

Page Routing Option #2

- Hash Routing
  - also use `<svelte:component this={component} props />`
  - select new component with
    `<svelte:window on:hashchange={handleHashChange} />`
    - `location.hash` holds current hash value
  - URL in browser address bar does change (only hash portion), so can bookmark pages

Page Routing Option #3

- Use a library like page.js at https://github.com/visionmedia/page.js
  - “tiny Express-inspired client-side router”
  - not specific to Svelte

Page Routing Option #4

- Use SvelteKit file-based routing
- Best option in my opinion
- Described in section #3 “Deeper Into SvelteKit”

Component Source Files ...

- `.svelte` files with four optional parts
  - `<script context="module">…</script>`
    - think of as class level
    - can export multiple values including constants and functions
  - `<script>…</script>`
    - think of as instance level
    - automatically exports the component and cannot export anything else
  - HTML elements
    - can include Svelte logic syntax and `{expression}` interpolations
  - `<style>…</style>`
    - automatically scoped to component

Sections can appear in any order, but this is the **recommended order**.

... Component Source Files

```
<script context="module">
  not used often
</script>
<script>
  export let name;  declares a prop
</script>
<h1>Hello, {name}!</h1>  uses a prop
<style>
  h1 {  scoped to this component
    color: red;
  }
</style>
```

Props

- Primary way to pass data from parent components to child components
- Parent component example
  - `<SomeChild propA="some value" propB={7} />`
- Values can be any kind of JS value
  - boolean, number, string, object, array, function
- Shorthand for prop values in variable with same name
  - `<SomeChild {propA} {propB} />`
- Can spread object properties: `<SomeChild {...obj} />`

Accepting Props

- `.svelte` files are not JS files
- Svelte uses JS `export` keyword to identify props
- For example
  - `export let name;` required
  - `export let score = 0;` optional
- Use in JS code and interpolations
  just like any other variable
- Parent components can `bind` to props to get updates
  - more on this later

Interpolation in HTML

- Expressions in curly braces
  - including function calls
- Examples
  - `<h1>{title}</h1>` variable
  - ```

      {amount * (taxRate + 1)}

    ```
    expression
  - `<p>{getDescription(product)}</p>` function call
    ```

Component State

- Variables declared at top level of **`<script>`** are state if used in HTML
- Changes to these variables trigger updates to DOM that depends on them
- Referred to as “reactivity”
- We saw this in REPL code for **`Counter`**, **`Greet`**, and **`Todo App`**

Reactivity

- DOM updates in response to changing top-level variables if ...
  - value is changed
  - value is an object and one of its properties is modified
- When changing an array, trigger reactivity by creating new array or assigning variable to itself
  - `myArr = myArr.concat(newValue);`
  - `myArr = [...myArr, newValue];`
  - `myArr.push(newValue); myArr = myArr;` most efficient

Reactive Statements ...

- `.svelte` files are not JS files
- Svelte uses label name `$` to mark “reactive statements”
- JS labels are followed by a colon
- Reactive statements are executed
  - initially
  - again when value of any referenced variable changes
- Like spreadsheet cells that contain formulas

... Reactive Statements ...

- Can be assignment to a variable not yet declared
  - `$: total = scores.reduce((acc, s) => acc + s, 0);`
- Can use for debugging
  - `$: console.log('total =', total);`
- Can call a function every time an argument changes
  - `$: evaluateCart(cart, taxRate);`

… Reactive Statements

- Can be a block containing any # of statements that are re-executed every time any referenced variable changes
- Example
  - ```
    let total;
    $: {
      total = scores.reduce(
        (acc, s) => acc + s, 0);
      console.log('total =', total);
    }
    ```
- See “Loan Calculator” REPL

re-executed every time **scores** array changes
    ```

Conditional Logic in HTML ...

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

opens with #

continues with :

ends with /

... Conditional Logic in HTML

```
{#if temperature > 80}
  <p>It's hot.</p>
{:else if temperature < 40}
  <p>It's cold.</p>
{:else}
  <p>Go for a run!</p>
{/if}
```

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

… Iteration in HTML

more on **bind** later

```
<select bind:value={favoriteColor}>
  <option value="">Select a color</option>
  {#each colors as color}
    <option>{color}</option>
  {/each}
</select>
```

assumes **colors** is an array of strings

Handling Promises in HTML ...

- `{#await promiseExpr}`
  `HTML to render before Promise resolves or rejects`
  `{:then result}`
  `HTML to render if Promise resolves`
  `{:catch error}`
  `HTML to render if Promise rejects`
  `{/await}`
- Not used often
  - more typical to handle with JS in `<script>`

could render a spinner

... Handling Promises in HTML

```
{#await getWeatherForecast(zipCode)}
  <p>... loading forecast ...</p>
{:then result}
  <p>
    high temperature today is
    {result.high}
  </p>
{:catch error}
  <p class="error">
    error getting forecast: {error}
  </p>
{/await}
```

Key Blocks ...

- Can surround HTML and components with a key block
- Causes corresponding DOM to be destroyed and recreated when value of expression changes
-
```
{#key expression}
  HTML and/or components
{/key}
```

… Key Blocks

- Example
  - ```
    {#key languageCode}
      <ContactInfo {person} />
    {/key}
    ```

assumes `ContactInfo` renders differently when `languageCode` changes

- Not used often
  - useful when HTML doesn't directly depend on a variable that has changed
    - perhaps variable is used in a function called in an interpolation
  - one use is to cause a CSS animation to re-run
    ```

Importing Other Components

- Components don’t specify their own name
  - source file names imply their name
    and typically those names are used
- Parent components choose name in import
  - `import OtherName from './SomeName.svelte';`

Global vs. Scoped Styles ...

- CSS rules in **`<style>`** elements in **`.svelte`** files are scoped to the component
  - achieved by adding a generated CSS class name to all elements and CSS selectors (**`svelte-hash`**)
  - computed from hash of **`<style>`** contents
- Can include a CSS file for global styling
  - ex. create **`src/global.css`** and add following in **`src/routes/index.svelte`**
  - **`import '../global.css';`**

in Counter REPL examine “CSS output”

assumes use of SvelteKit file-based routing

... Global vs. Scoped Styles

- To prevent scoping, use `:global(selector)`
- Example
  - suppose this component
    - renders `<div class="my-class">...</div>`
    - renders a component inside that `div` that renders an element with CSS class `other-class`
  - `.my-class :global(.other-class) { ... }`
- **TIP**: Starting with selector matching HTML in this component prevents global part from affecting components outside this one

Conditional Styles

- `<div class="c1"`
  `class="c2 c3"`
  `class:c4={`***`expression`***`}`
  `class:c5>`
- `c1`, `c2`, and `c3` will always be applied
- `c4` will only be applied if
  `expression` evaluates to `true`
- `c5` will only be applied if
  there is a variable `c5` that evaluates to `true`

Installing Node.js

- Browse https://nodejs.org
  - click big, green button for LTS or Current version to download installer
  - double-click downloaded installer
- Can also use a tool like Node Version Manager (nvm)
  - installs multiple versions
  - can easily switch between them
  - https://github.com/nvm-sh/nvm

Creating a SvelteKit App ...

- **`npm init svelte@next project-name`**
  - uses npm package **`create-svelte`**
- Answer questions
  - Which Svelte app template?
    - SvelteKit demo app or Skeleton project (prefer this)
  - Use TypeScript? (prefer yes)
  - Add ESLint for code linting? (prefer yes)
  - Add Prettier for code formatting? (prefer yes)

… Creating a SvelteKit App

- Follow instructions that are output to install dependencies and run locally
  - **`cd project-name`**
  - **`npm install`**
  - **`npm run dev -- --open`** listens on port 3000 by default
- Recommended additional step
  - so ESLint doesn't run on generated files, create file **`.eslintignore`** containing line **`build/`**

Special Files and Directories

- **`src/app.html`** file
  - starting HTML file
- **`src/routes`** directory
  - holds page components and endpoints
- **`src/lib`** directory
  - holds other components and functions
- **`.svelte-kit`** directory
  - holds files generated by **`npm run dev`** and **`npm run build`** which create development and production versions of app

in **`build`** and **`dev`** subdirectories

Using ESLint

- Checks code for many issues
- To run, enter **`npm run lint`**

Default ESLint Config File Using TS

.eslintrc.cjs

```
module.exports = {
  root: true,
  parser: '@typescript-eslint/parser',
  extends: ['eslint:recommended', 'plugin:@typescript-eslint/recommended', 'prettier'],
  plugins: ['svelte3', '@typescript-eslint'],
  ignorePatterns: ['*.cjs'],
  overrides: [{ files: ['*.svelte'], processor: 'svelte3/svelte3' }]
  settings: {
    'svelte3/typescript': () => require('typescript')
  },
  parserOptions: {
    sourceType: 'module',
    ecmaVersion: 2019
  },
  env: {
    browser: true,
    es2017: true,
    node: true
  }
};
```

Can add globals and override rules. For example:

```
  globals: {
    Cypress: 'readonly',
    cy: 'readonly',
    describe: 'readonly',
    it: 'readonly'
  },
  rules: {
    // Allow use of @ts-ignore.
    '@typescript-eslint/ban-ts-comment': 'off'
  }
```

Using `svelte-check`

- Finds unused CSS
- Detects some accessibility issues
- Outputs JS/TS compiler errors
- To run, enter **`npm run check`**

Using Prettier

- Formats HTML, CSS, JS, TS, and more
- To run, enter **`npm run format`**
- Can also configure editor/IDE to run Prettier when files are saved
  - in VS Code, after modifying **`.prettierrc`** run “Developer: Reload Window”

Default Prettier Config File

.prettierrc

```
{
  "useTabs": true,
  "singleQuote": true,
  "trailingComma": "none",
  "printWidth": 100
}
```

Recommended additions and changes:

```
"arrowParens": "avoid",
"bracketSpacing": false,
"printWidth": 80,
"useTabs": false
```

`npm run dev` Options

- `-o` or `--open`
  - opens tab for app in default browser
- `-p` or `--port` followed by a number
  - listen on port other than 3000
- `-h` or `--host`
  - makes server available to other devices in same network
- `-H` or `--https`
  - uses HTTPS with self-signed certificate

Lesson #1 Q & A

#2: Deeper into Svelte

Lifecycle Functions

- Each are passed a function to invoke when the lifecycle event occurs
- **`onMount`**
  - invoked when component is added to DOM
- **`beforeUpdate`** rarely used
  - invoked before every component update
- **`afterUpdate`** rarely used
  - invoked after every component update
- **`onDestroy`** rarely used
  - invoked when component is removed from DOM

Uses include manipulating DOM generated by Svelte and calling REST services to get data to render.

Directives ...

- **`bind`** - two-way binding
  - **`<input bind:value={variable} />`**
- **`bind:this`** - DOM element access
  - **`<div class="dialog" bind:this={dialog}>`**
  - sets variable to DOM element
  - can use to manipulate DOM in **`onMount`**
- **`class:name={condition}`** - conditional CSS class
  - we learned about conditional styles earlier

can also use with **`textarea`** and **`select`**

… Directives

- **`on:`*`name`*`={`*`function`*`}`** - event handling
  - ex. **`<button on:click={handleClick} />`**
  - can also use an arrow function
- **`use:`*`fnName`*** - action
  - function is invoked when element is added to DOM
  - see “Action Demo” REPL
- **`animate`**, **`transition`**, **`in`**, and **`out`**
  - covered soon on slides starting with “Animation”

Component Communication Options

Need	Solution	
parent passes data to child	props	already discussed
parent passes HTML and components to child	slots	discussed with SvelteKit layouts
child notifies parent, optionally including data	events	
ancestor makes static data available to descendants	context	not covered
component shares data between all instances	module context	already discussed
any component subscribes to and publishes data	stores	

Events ...

- Events go from child components to their parent
  - as seen in “Todo App” REPL which dispatches **`toggleDone`** and **`delete`** events
- To dispatch events from a component
  - ```
    import {createEventDispatcher} from 'svelte';
    const dispatch = createEventDispatcher();
    dispatch('my-event', data);
    ```
- To listen for events in parent component
  - ```
    <Child on:my-event={handleMyEvent} />
    ```
  - ```
    function handleMyEvent(event) {
      const data = event.detail;
      ...
    }
    ```

in **`script`** tag
    ```

… Events

- Events propagate up one level if no handler is specified
  - `<Child on:my-event />`

Stores ...

- Hold data that can be shared between components
- Uses publish/subscribe
- Four kinds of stores
  - **writable** - components can modify most commonly used
  - **readable** - only the store can change
    - could get data from REST services and periodically update
  - **derived** - compute value from one or more other stores
  - **custom** - can provide a custom API to control use
    - typically built from a writable store

… Stores

- Can define and export all stores in **`src/lib/stores.js`**
  - **`import {derived, readable, writable} from 'svelte/store';`**

create stores by calling these functions

- Can import stores in any components
- Refer to store value by preceding store name with $
  - automatically subscribes on first use and unsubscribes when component instance is destroyed
- See “Writable Store” REPL

To save store values in **`sessionStorage`**, see “**`writableSession`** Store” slides in Bonus section.

Animation

- Supported by three packages
  - **`svelte/animate`**
  - **`svelte/motion`**
  - **`svelte/transition`**
- All are CSS-based rather than JS-based
  - good performance because main thread is not blocked
- Can define custom transitions
  - see “Custom Transition (spin)” REPL

Easing Functions in `svelte/easing`

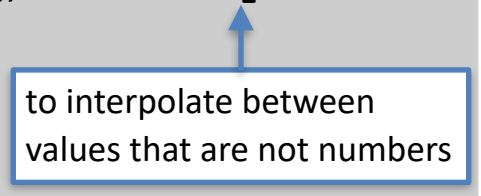
- These control rate of change through an animation
  - constant rate: **`linear`**
  - simple curves: **`sine`**, **`quad`**, **`cubic`**, **`quart`**, **`quint`**, **`expo`**, and **`circ`**
  - curves that move backward & forward: **`back`**, **`elastic`**, and **`bounce`**
  - actual names end in **`In`**, **`Out`**, or **`InOut`**
- Browse the Ease Visualizer to explore these
  - https://svelte.dev/examples#easing
  - for example, examine differences between **`backIn`**, **`backOut`**, and **`backInOut`**

`svelte/animate` Package

- Currently only defines **`flip`** function
  - stands for first, last, invert, play; doesn't flip anything
  - animates changes to x/y position from old to new
  - supports options **`delay`**, **`duration`**, and **`easing`**
  - see "Flip Animation" REPL

`svelte/motion` Package

- Defines **`spring`** and **`tweened`** functions
  - both return a writable store that is used to animate changes to a value
  - supports options **`delay`**, **`duration`**, **`easing`**, and **`interpolate`**
  - see “Pie Chart (svelte/motion)” REPL



`svelte/transition` Package

- Defines many directives
  - **`blur`**, **`draw`**, **`fade`**, **`fly`**, **`scale`**, and **`slide`**
  - see “Transition Animations” REPL
    - focus on one animation at a time
  - see “Draw Animation” REPLs
    - works with SVG **`path`** elements
- Defines **`crossfade`** function
  - see “Crossfade Demo” REPL

Special Elements ...

- Render a dynamically selected component
  - **`<svelte:component this={expr} props />`**
  - recall use in manual and hash routing
- Handle **`window`** events
  - **`<svelte:window on:eventName={function} />`**
  - examples of window event names are **`hashchange`** and **`resize`**
  - recall use of **`hashchange`** in hash routing
  - can use **`resize`** (in addition to CSS media queries) to implement responsive components

... Special Elements ...

- Get **`window`** properties
  - **`<svelte:window bind:`*`propertyName`*`={`*`variable`*`} />`**
  - available properties are
    `innerHeight`, **`innerWidth`**,
    `outerHeight`, **`outerWidth`**,
    `scrollX`, **`scrollY`**,
    and **`online`**
  - **`innerWidth`** is useful for developing responsive components
  - can only use once per component,
    but can bind to any number of properties

can use in reactive statements

… Special Elements

- Handle **`body`** events
  - **`<svelte:body on:eventName={function} />`**
  - not commonly used
- Insert elements in **`head`**
  - **`<svelte:head>elements</svelte:head>`**
  - examples include **`link`**, **`script`**, and **`title`**
- Specify Svelte compiler options
  - **`<svelte:options option={value} />`**
  - not commonly used

only **`title`** is commonly used

Libraries to Consider

- All are in npm - https://www.npmjs.com
- **dialog-polyfill**
  - for using HTML **`dialog`** element in browsers that do not yet support it (such as Firefox and Safari)
- **svelte-fa**
  - for rendering FontAwesome icons
- **svelte-material-ui**
  - collection of Material UI components implemented in Svelte

Testing SvelteKit Apps

- Three tools are covered
  - **Jest** for unit test
  - **Cypress** for end-to-end tests
  - **Storybook** for component demos and manual testing
- All are demonstrated in GitHub repo https://github.com/mvolkmann/sveltekit-testing

Testing with Jest ... https://jestjs.io

- Install Jest
  - **`npm install -D`** ***`name`*** where ***`name`*** is
    - if using JavaScript, only **`jest`**
    - if using TypeScript, **`ts-jest`** and **`@types/jest`**
- Add npm scripts in **`package.json`**

```
"test": "jest src",
"test watch": "npm run test -- --watch",
```

… Testing with Jest …

- Configure by creating **`jest.config.cjs`** containing

```
module.exports = {
  bail: false,
  moduleFileExtensions: ['js', 'ts'],
  transform: {
    '^.+\\.(ts|tsx)$': 'ts-jest'
  },
  verbose: true
};
```

file extension is **`.cjs`**, not **`.js`**

… Testing with Jest

- Create test files with **`.spec.js`** or **`.spec.ts`** extensions
- Example
  - **`src/lib/util.ts`**

```
export function add(n1: number, n2: number): number {
  return n1 + n2;
}
```

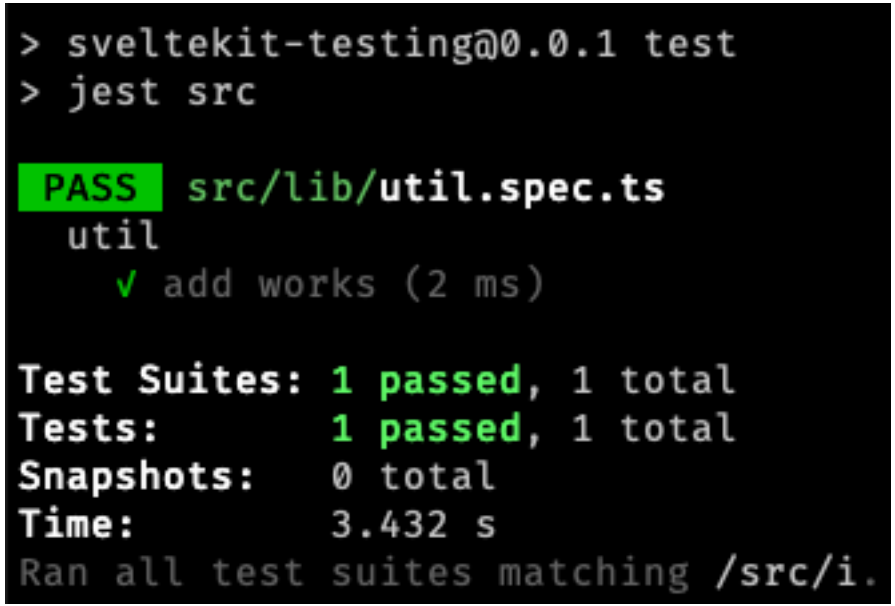
  - **`src/lib/util.spec.ts`**

```
import {add} from './util';
describe('util', () => {
  test('add works', () => {
    expect(add(0, 0)).toBe(0);
    expect(add(1, 2)).toBe(3);
  });
});
```

- To run tests
  - enter **`npm run test`** or **`npm run test:watch`**

Jest Output

```
cd sveltekit-testing
npm test
```



Testing Components With Jest

- Jest can test components too
  - need to install additional libraries like **`svelte-jester`** and **`@testing-library/svelte`**
  - but consider relying on Cypress for component tests
- Much more to learn about Jest,
  but this is enough to get you started

Testing with Cypress ... https://www.cypress.io

- Install by entering **`npm install -D cypress`**
- Add npm scripts in **`package.json`**

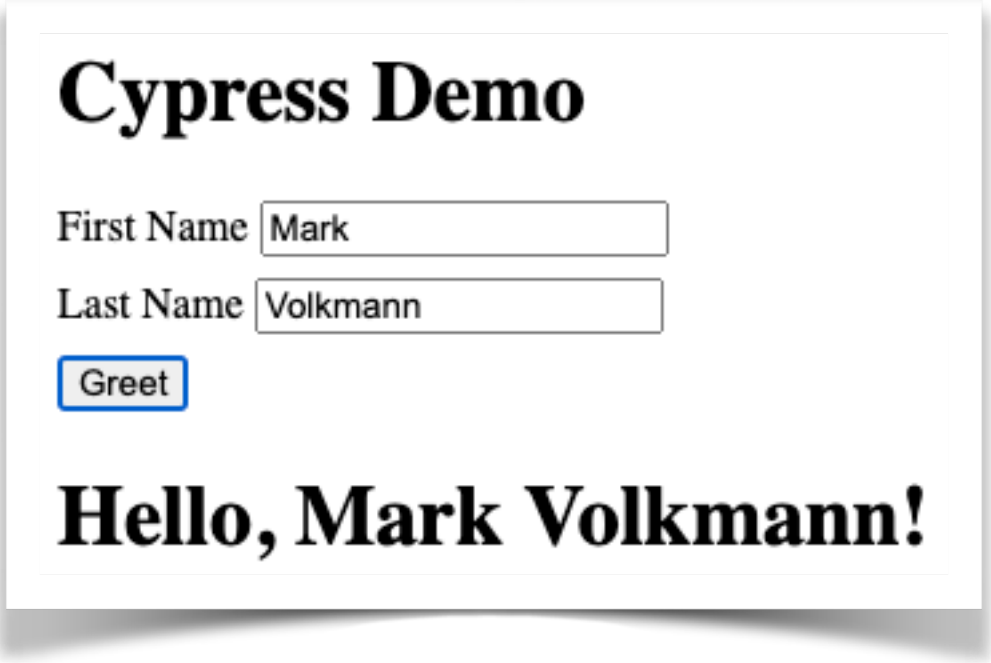
```
"cy:open": "cypress open",
"cy:run": "cypress run",
```

- Create initial Cypress files by entering
  - **`npm run cy:open`**
  - will get error “The plugins file is missing or invalid.”
  - change extension of **`cypress/plugins/index.js`** to **`.cjs`**

… Testing with Cypress

- Create tests in **`cypress/integration`** directory with **`.spec.js`** extension
  - see example ahead
  - can delete provided example tests
- Run tests
  - start local server by entering **`npm run dev`**
  - start Cypress by entering **`npm run cy:open`**
  - click a specific test file or "Run *n* integration tests"
- Much more to learn about Cypress, but this is enough to get you started

Cypress: Example Component ...



… Cypress: Example Component

src/routes/index.svelte

```
<script>
  let firstName = '';
  let lastName = '';
  let greeting = '';
  function greet() {
    greeting = `Hello, ${firstName} ${lastName}!`;
  }
</script>
<h1>Cypress Demo</h1>

<form on:submit|preventDefault={greet}>
  <div>
    <label for="first-name">First Name</label>
    <input name="first-name" bind:value={firstName} />
  </div>
  <div>
    <label for="last-name">Last Name</label>
    <input name="last-name" bind:value={lastName} />
  </div>
  <button>Greet</button>
</form>

<h1>{greeting}</h1>

<style>
  form > div {
    margin-bottom: 0.5rem;
  }
</style>
```

Cypress: Example Test

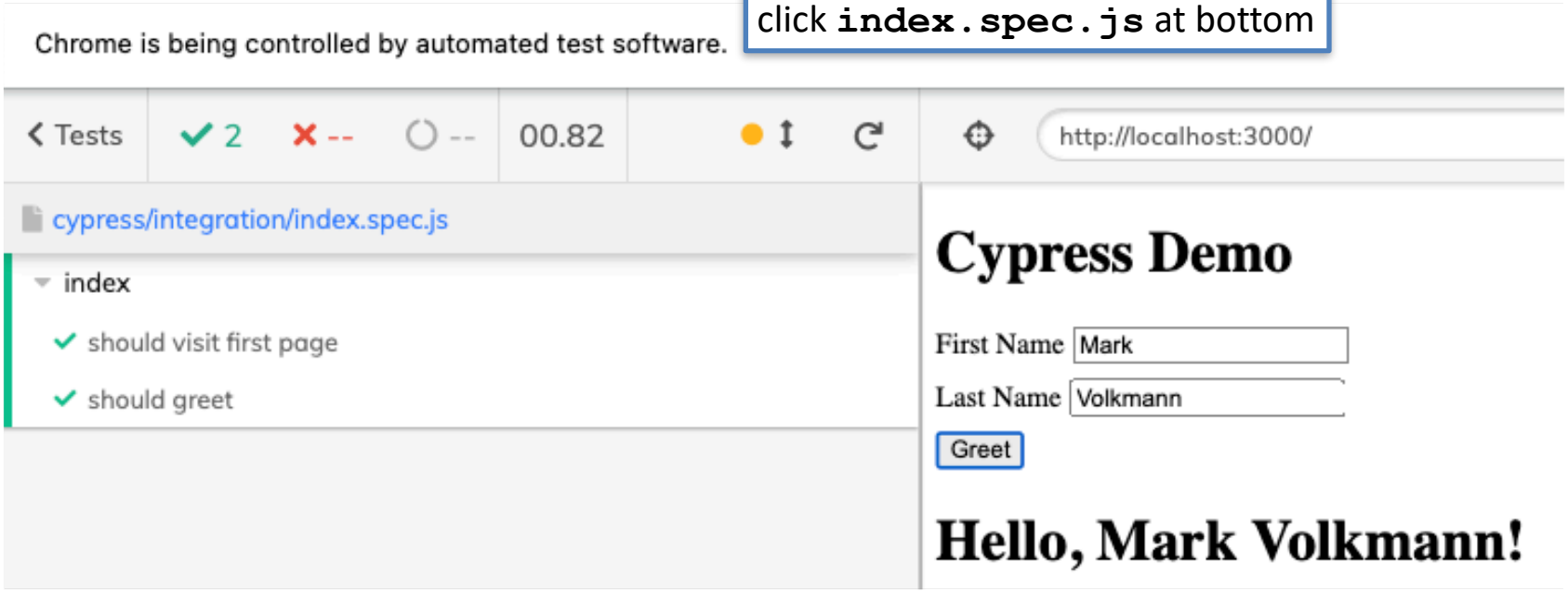
cypress/integration/index.spec.js

```
const baseurl = 'http://localhost:3000/';

describe('index', () => {
  it('should visit first page', () => {
    cy.visit(baseurl);
    cy.contains('Cypress Demo');
  });

  function type(label, text) {
    cy.contains(label).parent().children('input').type(text);
  }

  it('should greet', () => {
    cy.visit(baseurl);
    type('First Name', 'Mark');
    type('Last Name', 'Volkmann');
    cy.contains('Greet').click();
    cy.contains('Hello, Mark Volkmann!');
  });
});
```

finds parent element of **label**, then **input** child

Cypress Web UI

```
cd sveltekit-testing
npm run dev
```
open another terminal
```
npm run cy:open
```
click **`index.spec.js`** at bottom



Testing with Storybook ... https://storybook.js.org

- Install by entering **`npx sb init`**
- This does several things
  - adds npm scripts in **`package.json`**

    ```
    "storybook": "start-storybook -p 6006",
    "build-storybook": "build-storybook",
    ```

  - creates **`.storybook`** directory
    - contains **`main.js`** that configures Storybook
  - creates **`src/stories`** directory
    - contains example components and stories that can be deleted

… Testing with Storybook

- In `.storybook/main.js` change `require` to `import`
- Change extension of `.storybook/main.js` to `.cjs`
- Add stories
  - create files in `src/stories` with `.stories.js` extension
  - typically file name matches component source file name
  - these files can define multiple stories for the same component, each demonstrating different features
  - see example ahead
- Run by entering `npm run storybook`

Pie Component ...

```
<script>
  import {tweened} from 'svelte/motion';

  export let bgColor = 'tan';
  export let fgColor = 'blue';
  export let size = 50;
  export let value; // 0 to 100

  const store = tweened(value, { duration: 500 });
  let dashArray = '';
  $: half = size / 2;
  $: viewBox = `0 0 ${size} ${size}`;
  $: circumference = 2 * Math.PI * half;
  $: {
    const v = Math.max(0, Math.min(100, value));
    store.set(v);
    const dash = ((v / 100) * circumference) / 2;
    dashArray = `${dash} ${circumference - dash}`;
  }
</script>
```

Understanding this component is not important. It is just a good component to demonstrate in Storybook.

describes pie wedges

ensures value is in range

… Pie Component

```
<svg height={size} width={size} {viewBox}>
  <circle class="bg" fill={bgColor} r={half} cx={half} cy={half} />
  <circle
    class="fg"
    r={half / 2}
    cx={half}
    cy={half}
    fill="transparent"
    stroke={fgColor}
    stroke-width={half}
    stroke-dasharray={dashArray}
  />
</svg>

<style>
  svg {
    transform: rotate(-90deg);
  }
</style>
```

Pie Stories

```
import Pie from '../lib/Pie.svelte';

export default {
  title: 'Pie',
  component: Pie,
  argTypes: {
    bgColor: { control: 'color' },
    fgColor: { control: 'color' },
    size: { control: 'number' },
    value: { control: 'range' }
  }
};
```

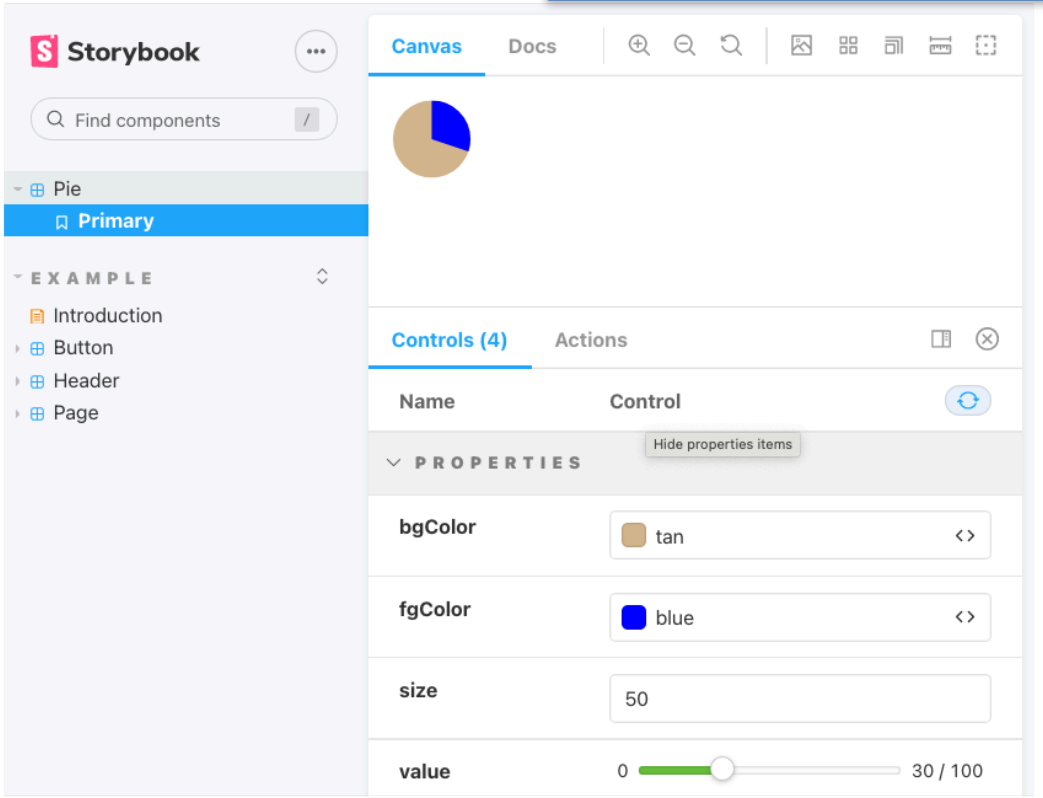
creates controls for changing prop values

```
const Template = (props) => ({Component: Pie, props});

export const Primary = Template.bind({});
Primary.args = {
  bgColor: 'tan',
  fgColor: 'blue',
  size: 50,
  value: 30
};
```

defines a story;
can define more than one with different names and default props

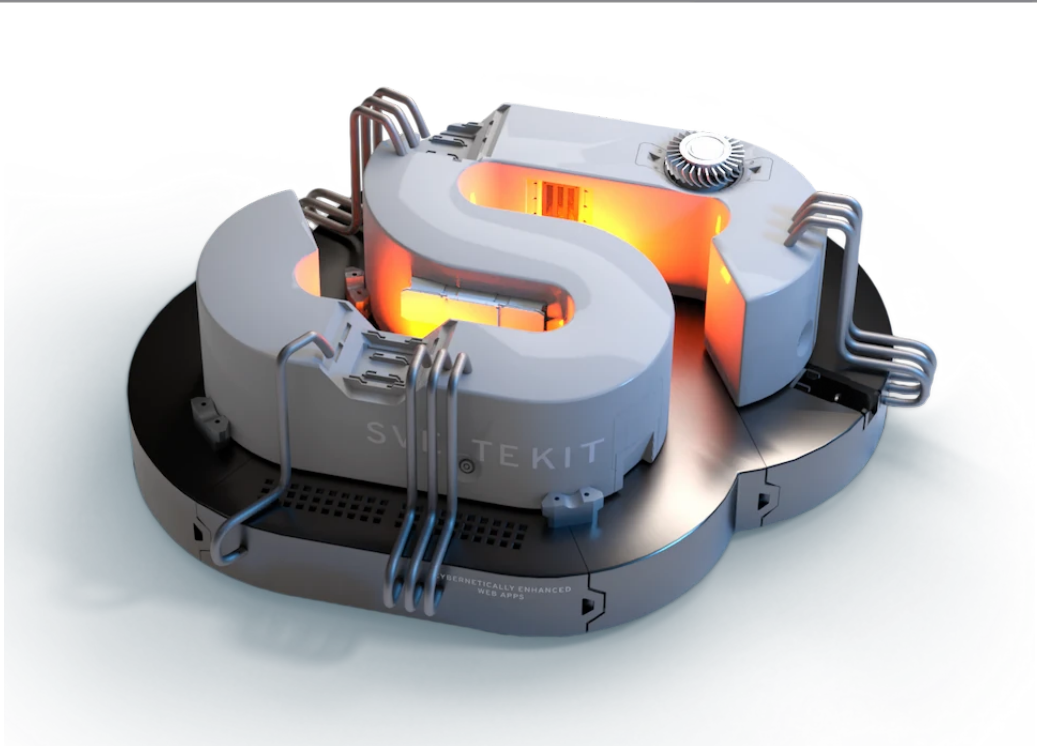
Storybook Web UI

```
cd sveltekit-testing
npm storybook
```



Lesson #2 Q & A

#3: Deeper into SvelteKit



npm Scripts Provided by SvelteKit

- **`check`** - checks for errors using **`svelte-check`**
- **`check:watch`**
  - same as **`check`**, but rechecks automatically when changes are detected
- **`lint`** - checks for errors using ESLint
- **`format`** - formats code using Prettier
- **`dev`** - runs app in development mode
- **`build`** - builds production version of app using specified adapter
- **`preview`** - runs app in production mode
  - must build production version before running this

we'll discuss adapters later

Using TypeScript

- Request when creating a new SvelteKit project
  - can configure manually, but that's much more work
- Indicate usage in **`script`** tags
  - **`<script lang="ts">`**
- Add types to declarations including
  - props - **`export let name: type;`**
  - variables: **`const name: type;`** and **`let name: type;`**
  - function parameters
  - function return types

SvelteKit TypeScript types are defined in **`node_modules/@sveltejs/kit/types;`** import them from **`@sveltejs/kit`**

Server-Side Rendering (SSR)

- By default, SvelteKit renders first page visited on server and others in browser
- After downloading first server-rendered page, renders it again in browser to make it interactive
  - listening for events, updating state, and surgically updating DOM
- SSR can be disabled
  - globally - edit **`svelte.config.js`** and set **`kit.ssr`** to **`false`**
  - for a specific component

```
<script context="module">
  export const ssr = false;
</script>
```

File-based 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
  - access in module context **`load`** functions with **`page.params`**
- Pages can be rendered in three ways
  - build time (using static adapter), runtime on server, or runtime in browser (disabling SSR)

Mapping Files to URLs

all these file paths are under `src/routes/`

File	URL	Parameters
`index.svelte`	`/`	none
`person.svelte`	`/person`	none
`person/index.svelte`	`/person`	none
`person/[personId].svelte`	`/person/`*`personId`*	`personId`
`person/[personId]/index.svelte`	`/person/`*`personId`*	`personId`
`person/[personId]/dog/[dogId].svelte`	`/person/`*`personId`*`/dog/`*`dogId`*	`personId` and `dogId`
`person/[personId]/dog/[dogId]/index.svelte`	`/person/`*`personId`*`/dog/`*`dogId`*	`personId` and `dogId`
`person/[personId]/dog/[dogId]/photos.svelte`	`/person/`*`personId`*`/dog/`*`dogId`*`/photos`	`personId` and `dogId`

Demo Project

- Many features of SvelteKit are demonstrated in the GitHub repo at https://github.com/mvolkmann/sveltekit-routes

`load` Functions

- Page and layout source files can define a **`load`** function
  - can load data, perhaps by calling REST services
  - primary use is to specify props to pass to component instances
  - page is not rendered until this function completes
  - must be in **`module`** context because
    it runs before component is rendered
  - only used in page and layout components, not other components
  - called in both server (SSR) and browser (client-rendered)

`load` Function Parameter

only one

- Object containing these properties

`fetch` and `page` are the most frequently used

  - **`fetch`**
    - function for using Fetch API provided by browser or a server library
  - **`page`**
    - object with **`host`**, **`path`**, **`params`**, and **`query`** properties

strings

objects

  - **`session`**
    - for passing data from server such as current user id
  - **`stuff`**
    - data passed from layout components
    - useful when multiple pages have common layout and need same data

`load` Function Return Value

- Object containing these properties

`props`, `status`, and `error` are the most frequently used

  - **`error`** - error description if any, perhaps from REST call failures
  - **`maxage`** - seconds to cache page; doesn't apply to layouts
  - **`props`** - object specifying props to pass to component instances
  - **`redirect`**
    - to redirect to different page,
      perhaps based on data returned from a REST call
  - **`status`** - HTTP status code
  - **`stuff`** only specified in layout components
    - data provided as input to subsequent page and layout components

`load` Function Error Handling

- Don’t need to wrap REST calls in try/catch if displaying errors using error page is acceptable
  - we’ll see how to define an error page later
- But typically if **`res.ok`** is **`false`** the code should
  - return an object with **`error`** and **`status`** properties or
  - **`throw await res.text();`**

`load` Function Called When

- `load` functions are called when the page is rendered and again every time one of these change if they are used in the function
  - `page.path`
  - `page.query`
  - `session`
  - `stuff`
- Note that a change only to `page.params` does not trigger a call

Example `load` Function

from `src/routes/person/[personId]/dog/[dogId]/index.svelte`

```
<script context="module" lang="ts">
  import type {LoadInput, LoadOutput} from '@sveltejs/kit';

  export async function load(
    {fetch, page}: LoadInput): Promise<LoadOutput> {
    const {personId, dogId} = page.params;
    const url = `/api/person/${personId}/dog/${dogId}`;
    const res = await fetch(url);
    if (res.ok) {
      const dog = await res.json();
      return {maxage: 60, props: {dog}};
    }

    const error = await res.text();
    return {error, status: res.status};
  }
</script>
```

Caches page for specified **`personId`** and **`dogId`** for 60 seconds. Any request for cached values received in that time period will be served the cached page and this **`load`** function will not be run.

Prefetching ...

- Calls **`load`** function of page before request to navigate to page is made
  - can be triggered by hovering of an anchor tag (**`<a>`**) that has **`sveltekit:prefetch`** attribute
  - can be triggered programmatically by calling **`prefetch`** function defined in **`$app/navigation`**
    - ex. when **`mouseover`** or **`focus`** event occurs on **`button`** element
- Can make page render faster
  - because when navigation is actually requested, data needed by page has already been loaded

see **`a`** and **`button`** elements in **`src/routes/index.svelte`**

… Prefetching

- Regardless of whether prefetching is used, target page will not render until **`load`** function completes
- If **`load`** function might be slow, perhaps due to calling a slow REST service, display a loading indicator
  - see how the sveltekit-routes app implements this in **`src/routes/index.svelte`** by using the **`navigating`** store and a call to **`setCursor`**

see Bonus slide “Provided Modules - **`$app/stores`**”

Layouts

- Components that define common
  - content and formatting for a set of pages
  - optional **`load`** function
- Contain a **`slot`** element for rendering a page component
- Defined in files named **`__layout.svelte`**
- Can be in multiple nested directories for nested layouts
- Layout in **`src/routes`**, applies to every page
  - common to define page header, nav links, and page footer here
  - if not present, defaults to **`<slot />`**

Layout Example

```
<header>...</header>

<nav>...</nav>

<main>
  <slot />
</main>

<footer>...</footer>

<style>...</style>
```

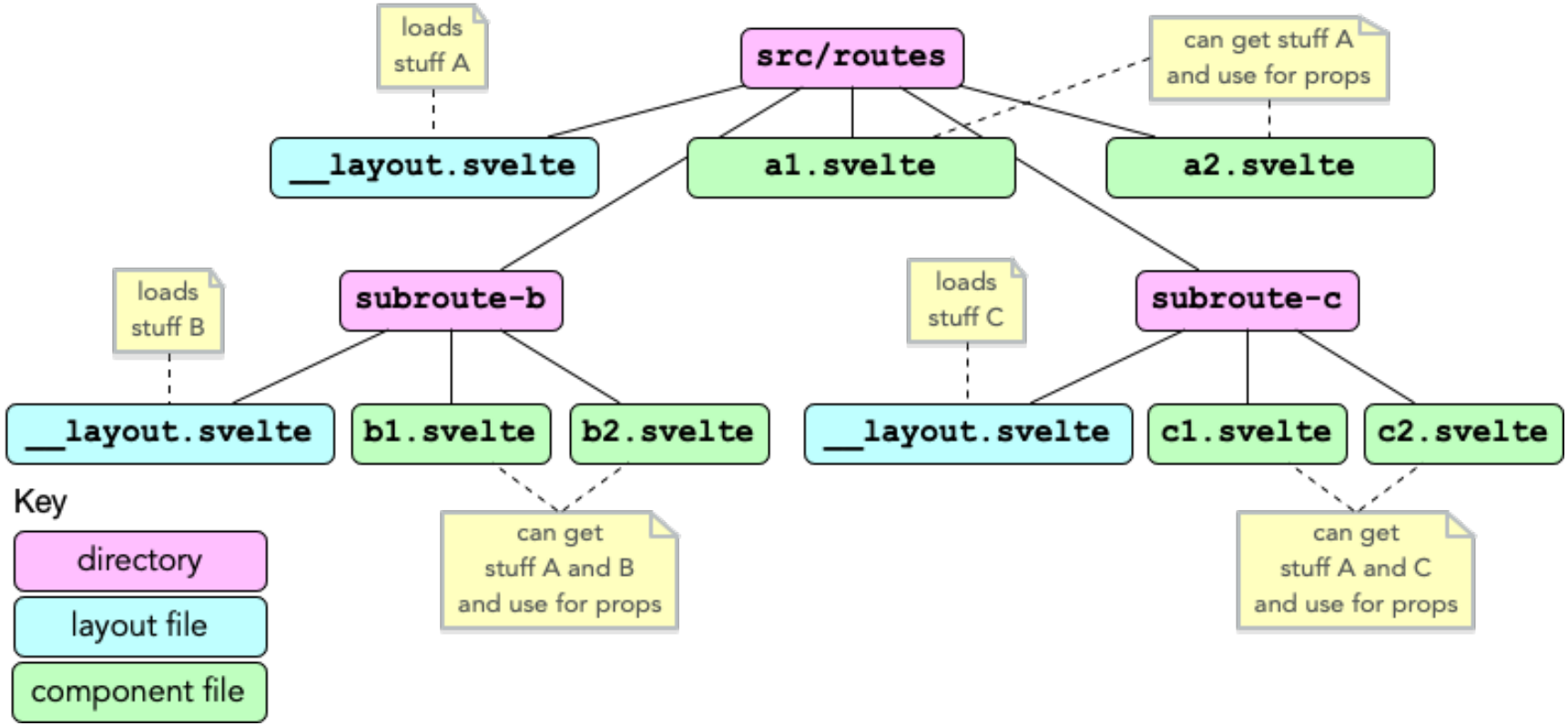
Sharing Data With Layouts

- If a set of pages share a common layout and the pages need some of same data from endpoints
  - define `load` function in `__layout.svelte`
    - get data from endpoints
    - return object with `stuff` property whose value is an object with properties that hold data needed by pages
  - define a `load` function in each page
    - get data needed from `stuff` property
    - return object with `props` property whose value is an object with properties that hold prop values

could be just `<slot />`

see demo in repo sveltekit-layout-context

Nested Layouts



Error Page ...

- Define in **`src/routes/__error.svelte`**
  - a default error page is provided if this is missing
- Rendered when
  - no matching route is found for UL
  - a page **`load`** function returns object w/ **`error`** property or throws an error

... Error Page

- Define module context **`load`** function
  - passed object containing **`error`** and **`status`** properties
  - stack trace is present in **`error`** value in development, but removed in production to avoid exposing implementation details
  - return object with **`props`** property that specifies props to be passed to component defined in same file
- Define component to render
  - same as other components with **`script`** element, HTML, and **`style`** element

Error Page Example

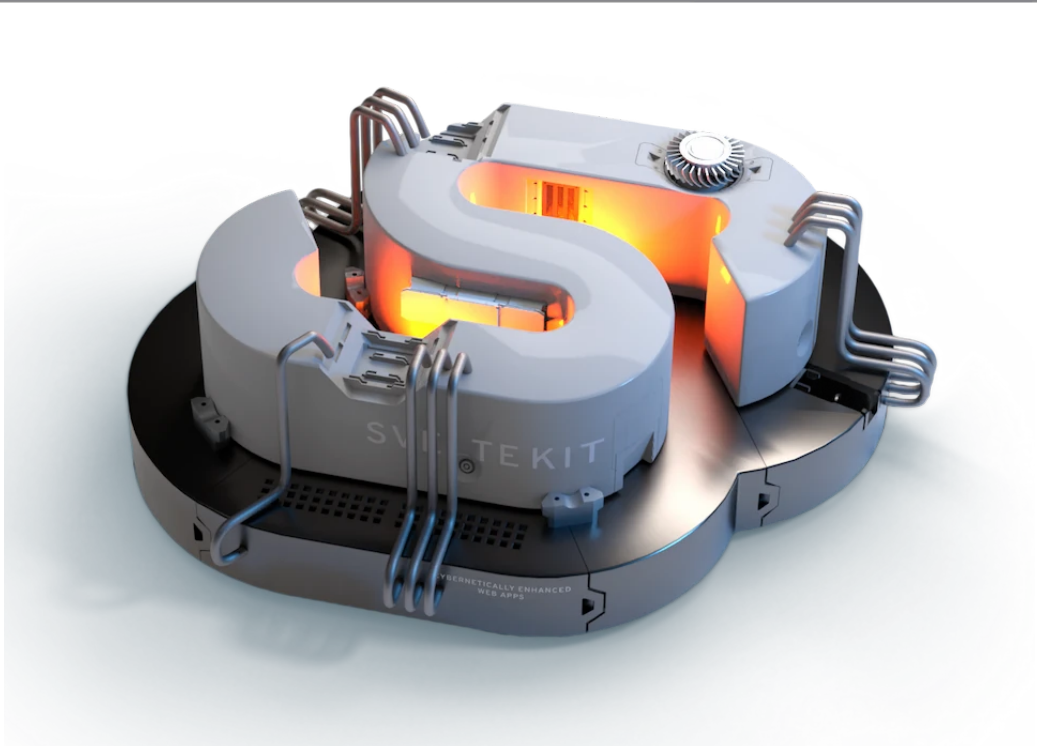
example in sveltekit-routes repo
`src/routes/__error.svelte`

```
<script context="module" lang="ts">
  import type {ErrorLoadInput, LoadOutput} from '@sveltejs/kit';

  export function load({error}: ErrorLoadInput): LoadOutput {
    return {
      props: {message: error.message}
    };
  }
</script>

<script lang="ts">
  export let message: string;
</script>
```

```
<h1>An error occurred ...</h1>
<p>{message}</p>

<style>
  p {
    color: red;
  }
</style>
```

Lesson #3 Q & A

#4: SvelteKit with API Endpoints



API Endpoints

- Endpoints are typically implemented as REST services
  - implies specific usage of HTTP verbs/methods
- SvelteKit apps can define the endpoints they use
  - can also use endpoints implemented with any tech stack hosted outside the app
    - Node.js, Deno, Python, Java, C#, Go, Rust, ...

API Endpoint Source Files ...

- Defined in **`src/routes`** directory like pages
- Two conventions
  - define in **`src/routes/api`** (prefer this)
  - define in **`src/routes`** in files with **`.json.js|ts`** extension
    - avoids conflict with page routes

… API Endpoint Source Files

- File and directory names inside square brackets indicate that a path parameter will be captured
  - just like in page routing
  - stored in **`Request`** object **`params`** property
- Define endpoints with exported functions named **`get`**, **`post`**, **`put`**, and **`del`**
  - **`delete`** is a JS keyword
  - these can send requests to other endpoints using global **`fetch`** function

Endpoints Not Targeting Specific Resource

- Define in file named **`index.js|ts`**
- **`get`** function retrieves all instances
  - returns array of objects
- **`post`** function creates a new instance
  - returns the instance or at least its id
  - status should be 201 Created
  - response header **`Location`** should be set to URL of new resource

Endpoints Targeting Specific Resource

- Define in file named **`[paramName].js|ts`**
  - typically **`paramName`** represents an instance id
- **`get`** function retrieves an existing instance
- **`put`** function updates an existing instance
- **`del`** function deletes an existing instance

Endpoint Function Input/Output

- Passed a **`Request`** object
- Return **`Promise`** that resolves to a **`Response`** object

```
import type {Request, Response} from '@sveltejs/kit';
```

- Properties of **`Request`** and **`Response`** objects are described on next two slides

Request Properties

- **`method: string`** - **`GET`**, **`POST`**, **`PUT`**, or **`DELETE`**
- **`host: string`** - of server
- **`path: string`** - of request URL
- **`params: Record<string, string>`** - path parameters
- **`query: URLSearchParams`** - query parameters
- **`headers: Headers`** - request headers
- **`rawBody: Uint8Array`** - binary request body
- **`body: ParameterizedBody<Body>`** - text request body
- **`locals: Locals`** - populated by "hooks" see bonus slides

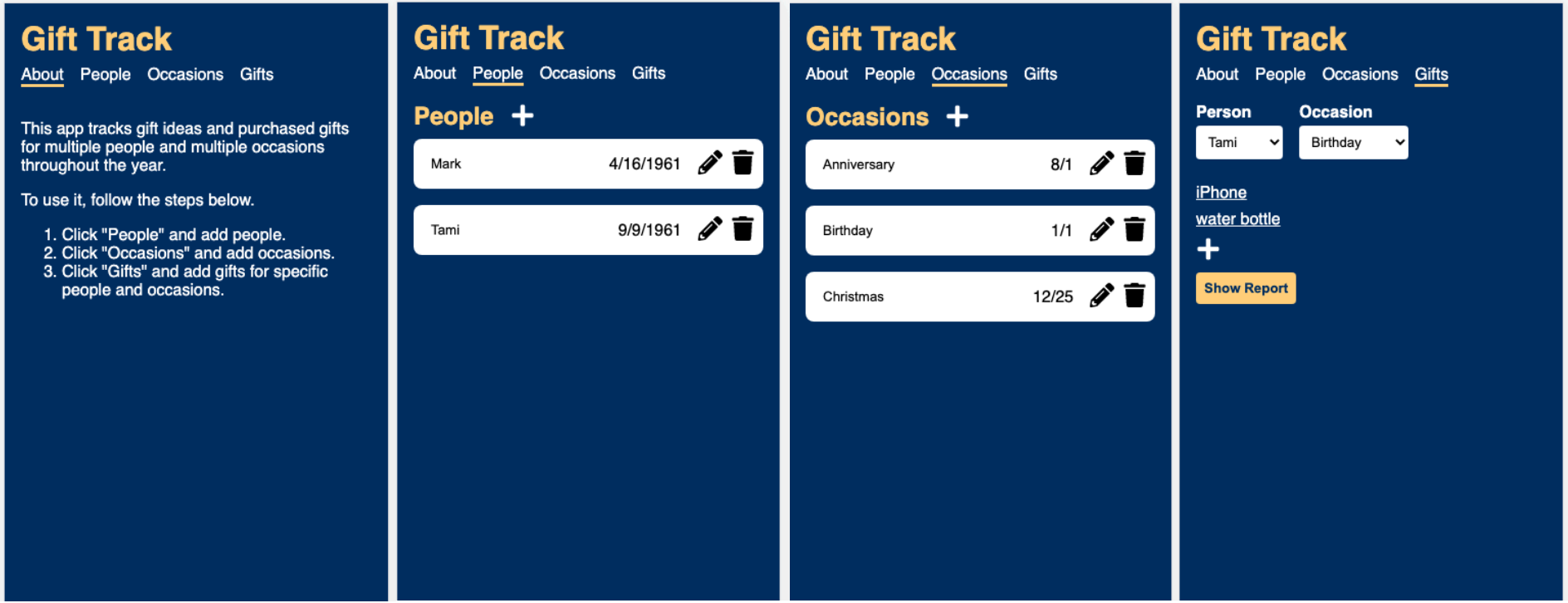
Response Properties

- **`status?: number`** - HTTP status code
  - defaults to 200
- **`headers?: ResponseHeaders`** - response headers
  - object where keys are header names and values are their values
- **`body?: JSONResponse | Uint8Array`** - response body
  - string if **`Content-Type`** is **`text/plain`**
  - object, array, or primitive if **`Content-Type`** is **`application/json`**
  - **`FormData`** object if **`Content-Type`** is **`application/x-www-form-urlencoded`** or **`multipart/form-data`**
  - **`Uint8Array`** otherwise; can hold binary data

Gift Track App ...

- Let's look at an app that uses many features of SvelteKit, including endpoints
- Tracks gift ideas for specific people and occasions
- Designed to work well on mobile devices
- 4 pages: About, People, Occasions, and Gifts
- Supports CRUD operations
  - on people, occasions, and gifts
  - demo each of these
- Persists to SQLite database

... Gift Track App



Endpoint Examples ...

- From Gift Track app https://github.com/mvolkmann/gift-track
  - **src/routes/api/occasion/index.ts**

An occasion has a **name** (ex. Birthday), **id**, and an optional date.

```
import type {Request} from '@sveltejs/kit';
import type {Occasion} from '$lib/types';
import {addOccasion, getAllOccasions} from '../data';

export async function get(): Promise<{body: Occasion[]}> {
  return {body: getAllOccasions()};
}

export async function post(request: Request): Promise<{body: Occasion}> {
  let occasion = (request.body as unknown) as Occasion;
  occasion = addOccasion(occasion);
  return {body: occasion};
}
```

new value includes assigned id

… Endpoint Examples …

- src/routes/api/occasion/[occasionId].ts

```
import type {Request} from '@sveltejs/kit';
import type {Occasion, OccasionResponse} from '$lib/types';
import {deleteOccasion, getOccasion, updateOccasion} from '../data';

export async function del(request: Request): Promise<OccasionResponse> {
  const id = Number(request.params.occasionId);
  const success = deleteOccasion(id);
  return {status: success ? 200 : 404};
}
```

specifies that **body** property type is **Occassion**

… Endpoint Examples

- src/routes/api/occasion/[occasionId].ts

```
export async function get(request: Request): Promise<OccasionResponse> {
  const id = Number(request.params.occasionId);
  const occasion = getOccasion(id);
  return occasion ? {body: occasion} : {status: 404};
}

export async function put(request: Request): Promise<OccasionResponse> {
  const id = Number(request.params.occasionId);
  const occasion = (request.body as unknown) as Occasion;
  occasion.id = id;
  const success = updateOccasion(occasion);
  return {status: success ? 200 : 404};
}
```

More Endpoint Details

- Default JSON responses
  - if no `Content-Type` header and body is an object, it will be converted to a JSON string
- If nothing is returned, `status` is set to `404`
- Files under `src/routes` with names beginning with an underscore are considered private
  - not pages or endpoints
  - can be imported by page and endpoint files
  - used to share code between these

Gift Track App

- Time permitting, review more code from this app
- Time permitting, download repo and run locally

Segment #4 Q & A

Wrap Up and Next Steps

- Now that you have seen nearly all the features of Svelte and SvelteKit ...
  - Consider how you would achieve the same things in other frameworks.
  - Would it require more code?
  - Would that code be harder to understand?
- There's no substitute for hands on experience
  - create some SvelteKit apps
  - send me questions; glad to help! - r.mark.volkmann@gmail.com

#5: Bonus Material

Including Dynamic HTML

- Can create strings of HTML in JS
  and render with `{@html expression}`
- Sanitize if untrusted
  - prevents cross-site scripting attacks
    by removing elements like `script`
  - one library to consider is sanitize-html in npm

`writableSession` Store ...

- Custom store that saves value in **`sessionStorage`**
- Restores value from there on browser refresh

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

function persist<T>(key: string, value: T) {
  if (value === null || value === undefined) {
    sessionStorage.removeItem(key);
  } else {
    sessionStorage.setItem(key, JSON.stringify(value));
  }
}
```

continued on next slide

... writableSession Store

```
function writableSession<T>(key: string, initialValue: T): Writable<T> {
  const sessionValue = JSON.parse(sessionStorage.getItem(key));
  const store = writable<T>(sessionValue || initialValue);
  store.subscribe(value => persist<T>(key, value));
  return store;
}

// Example usage
export const temperatureStore = writableSession<number>(
  'temperature',
  100 // initial value
);
```

Component Libraries

- A SvelteKit project can define a component library instead of an app
- Steps
  - add npm script **`"package": "svelte-kit package"`**
  - enter **`npm run package`**
    - compiles components in **`src/lib`** into **`package`** directory
    - contains all files needed to publish to npm including
      - **`package.json`** file
      - compiled component definitions
      - TypeScript type definitions in **`.d.ts`** files
  - enter **`npm publish`** to publish to npm

Route Conflicts

- Can occur when generating static pages that get content from endpoints (not common)
  - because output is written to file system
  - example: endpoints defined in **`src/routes/api/sport/index.js`** and **`src/routes/api/sport/hockey.js`** generate the files **`sport`** and **`sport/hockey`**
  - creates a conflict because **`sport`** cannot be both a file and directory
  - a solution is to modify the endpoint file extensions so the files are **`src/routes/api/sport/index.json.js`** and **`src/routes/api/sport/hockey.json.js`**
  - now generated files are named **`sport.json`** and **`sport/hockey.json`** and **`sport`** is only a directory

Using Environment Variables

- Vite uses **`dotenv`** to load environment variables from a **`.env`** file https://github.com/motdotla/dotenv
- Currently names should start with **`VITE_`**
- Create file **`src/lib/env.js|ts`** and add lines like

```
export const NAME = import.meta.env.VITE_NAME;
```

- Import in **`.svelte`** and **`.js|ts`** files with

```
import {NAME} from '$lib/env';
```

`tick` Function

- Rarely needed
  - example use case is restoring cursor position in an **`input`** after programmatically modifying its value
    - ex. masked input for entering a phone number
- Returns a **`Promise`**
  - resolves when pending DOM state changes have been applied
- **`import tick{} from 'svelte'`**
- Call from an **`async`** function with **`await tick();`**
- Then access the DOM and make updates

Transition Events

- Listen for these events to execute code at specific points in a Svelte transition
  - **`introstart`** - when an “in” transition begins
  - **`introend`** - when an “in” transition ends
  - **`outrostart`** - when an “out” transition begins
  - **`outroend`** - when an “out” transition ends
- One use case is moving focus into an **`input`** that transitioned into view
  - listen for **`introend`** event

Hooks

- Advanced feature of SvelteKit
- Defined by functions defined in `src/hooks.js|ts` or `src/hooks/index.js|ts`
- All run on server
- **`handle`** can modify response headers and bodies
- **`handleError`** can customize error messages and/or log them
- **`getSession`** returns session object that client code can access
- **`externalFetch`** can “modify or replace a fetch request for an external resource that happens inside a **`load`** function that runs on the server or during pre-rendering”

Provided Modules - `$app/env`

- Object with these properties
  - **`amp`** - boolean indicating if AMP mode is enabled
    - see https://amp.dev
  - **`browser`** - boolean indicating if code is running in browser
  - **`dev`** - boolean indicating if running in development mode
  - **`mode`** - the Vite mode; “development” or “production”
    - configured by **`config.kit.vite.mode`** property
  - **`prerendering`** - boolean indicating if prerendering is being used

most useful of these

Provided Modules - `$app/navigation`

- Object with these properties
  - **`goto(`*`href`*`, `*`options`*`)`** most useful of these
    - function that navigates to given URL
    - returns **`Promise`** that resolves if navigation succeeds or rejects if it fails
    - typically options are not supplied
  - **`invalidate(`*`href`*`)`**
    - function that causes the **`load`** function of the **`href`** to run again when navigating to that page
  - **`prefetch(`*`href`*`)`**
    - function that programmatically prefetches a given page
  - **`prefetchRoutes(`*`pathArray`*`?)`**
    - function that programmatically prefetches multiple pages

Provided Modules - `$app/paths`

- Object with these properties
  - **`assets`**
    - path from where assets like images are served
    - value comes from **`config.kit.paths.assets`** property
    - defaults to same as next property
  - **`base`**
    - path from where app is served

Provided Modules - `$app/stores`

- Object with these properties
  - **`getStores()`**
    - function that doesn't seem valuable
  - **`navigating`** most useful of these
    - readable store with value **`{from, to}`** while navigating and **`null`** otherwise
    - can use to set/unset wait cursor or show/hide loading spinner
  - **`page`**
    - readable store with value **`{host, path, params, query}`**
    - same as **`page`** property in object passed to **`load`** functions
  - **`session`**
    - writable store that holds data passed from server such as current user id

Provided Modules - `$lib`

- Path alias to **`src/lib`**
- Can add more path aliases
  - edit **`svelte.config.js`**
  - set **`kit.vite.resolve.alias`** to an object whose keys are aliases and whose values are calls to **`path.resolve`**
- If using TypeScript
  - also edit **`compilerOptions.path`** property in **`tsconfig.json`**

```
import path from 'path';
...
  kit: {
    ...
    vite: {
      resolve: {
        alias: {
          $routes: path.resolve('src/routes'),
          $src: path.resolve('src'),
          $view: path.resolve('src/view')
        }
      }
    }
  }
```

```
"paths": {
  "$routes/*": ["src/routes/*"],
  "$src/*": ["src/*"],
  "$view/*": ["src/view/*"]
}
```

Provided Modules - `$service-worker`

- Only available in service workers,
  so only need to learn about this if using them
- One use is to make navigation faster
  by precaching some JavaScript and CSS files
- Can import an object from this that has these properties
  - **`build`** - array of URLs for generated files that can be cached
  - **`files`** - array of URLs for static files that can be cached
  - **`timestamp`** - value of **`Date.now()`** at build time,
    useful for generating a unique cache name

Adapters ... as of September 2021, not very mature

- Adapt a SvelteKit app for deployment to a specific target
  - take files of the built app as input and
    output files needed for deployment
- Adapters currently provided by Svelte team
  - **`cloudflare-workers`**, **`netlify`**, **`node`**, **`vercel`**
  - **`static`** (use when entire site is static)
- Adapters currently provided by the community
  - **`begin`**, **`deno`**, **`firebase`**

… Adapters

- Steps to use
  - install with **`npm install -D /@sveltejs/adapter-`*`name`*`@next`**
    - the **`@next`** part won't be needed in the future
  - edit **`svelte.config.js`**
    - add **`import `*`name`*` from '@sveltejs/adapter-`*`name`**
    - set **`kit.adapter`** property in **`config`** object to a call to the imported function ***`name`***
    - documentation for the adapter will describe options that can be passed

Custom Adapters

- Can write your own adapters
  for other deployment targets
- See https://kit.svelte.dev/docs#writing-an-adapter