
SVELTE

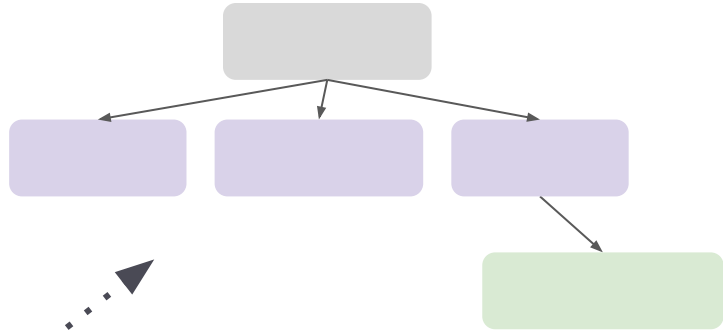
Agenda

1. Svelte/SvelteKit overview
2. Mini SvelteKit workshop – create a comments app!

weblab.is/svelte

More resources in `resources/outline.md`

What is Svelte?

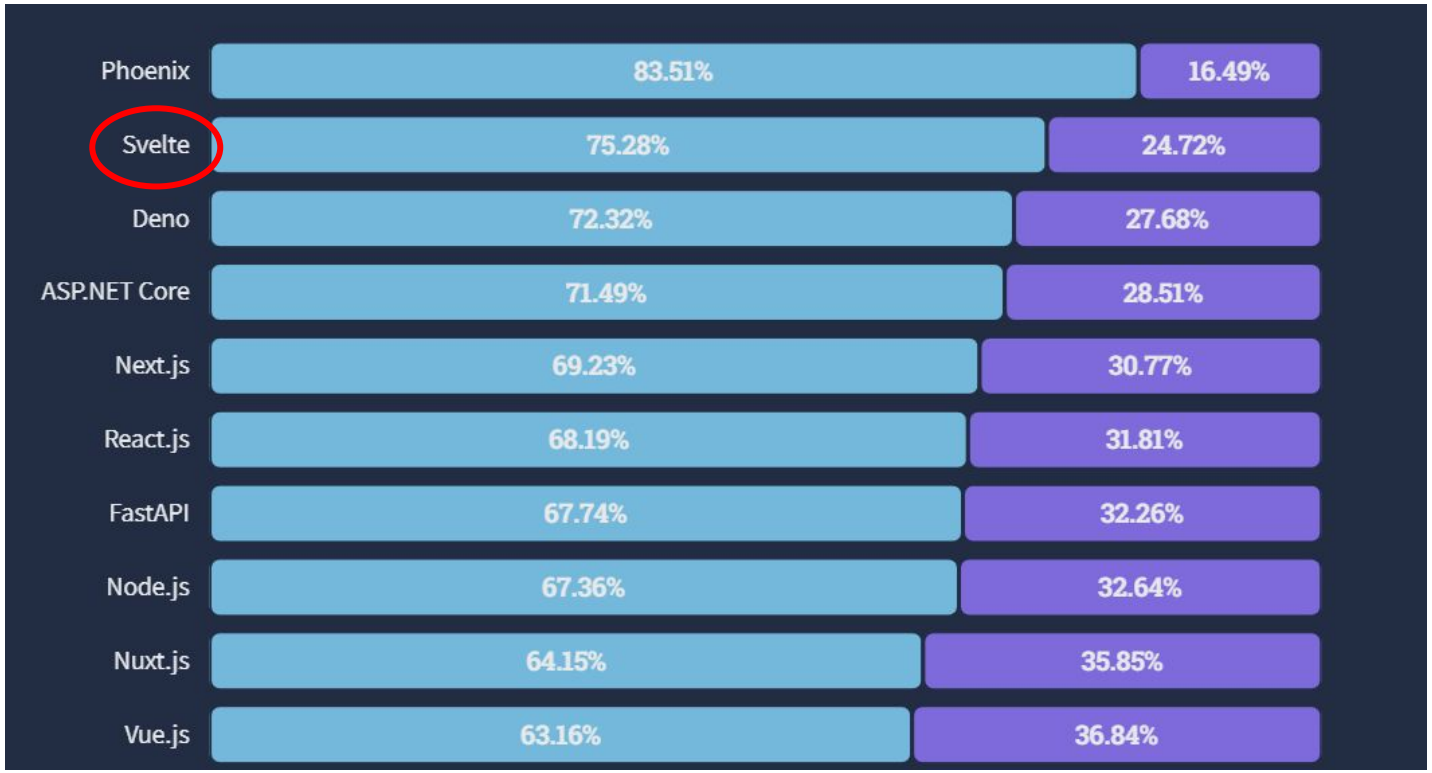


Svelte is a lightweight **component-based web framework** developed by **Rich Harris** in 2016.

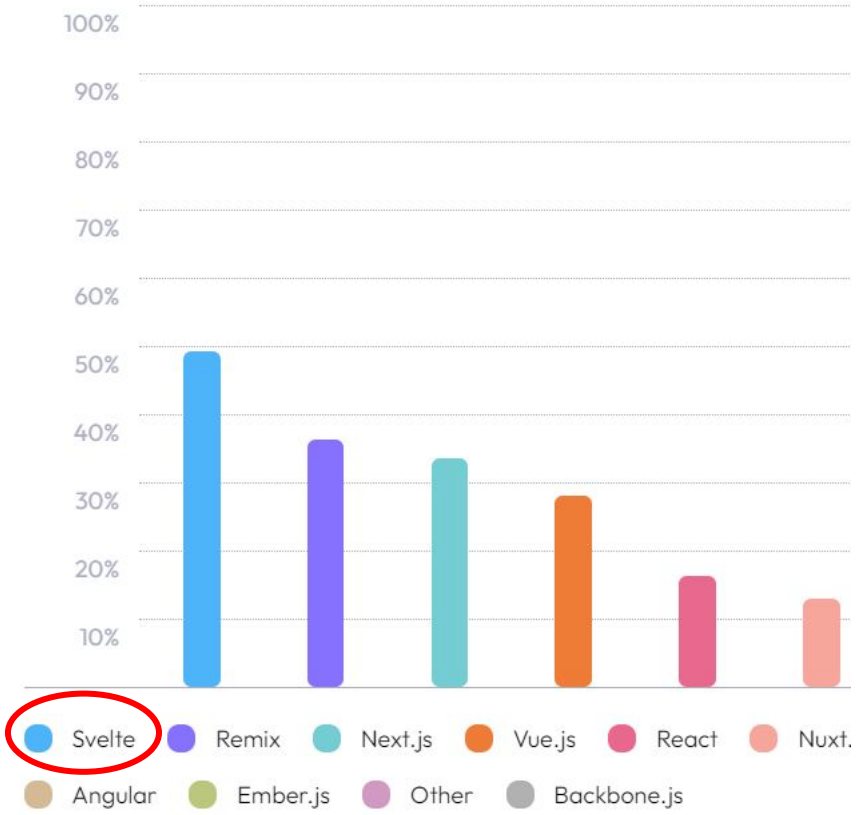
Former front end developer creating interactive articles with **The Guardian** and the **New York Times** graphics team. He now works at **Vercel** as of 2021.

Why do people like it?
(Svelte vs. React)

2nd **Most Loved Web Framework** in 2022 (*StackOverflow*)



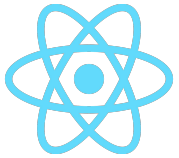
Which of the following frameworks would you like to learn in the future?



State of Frontend Survey, 2022

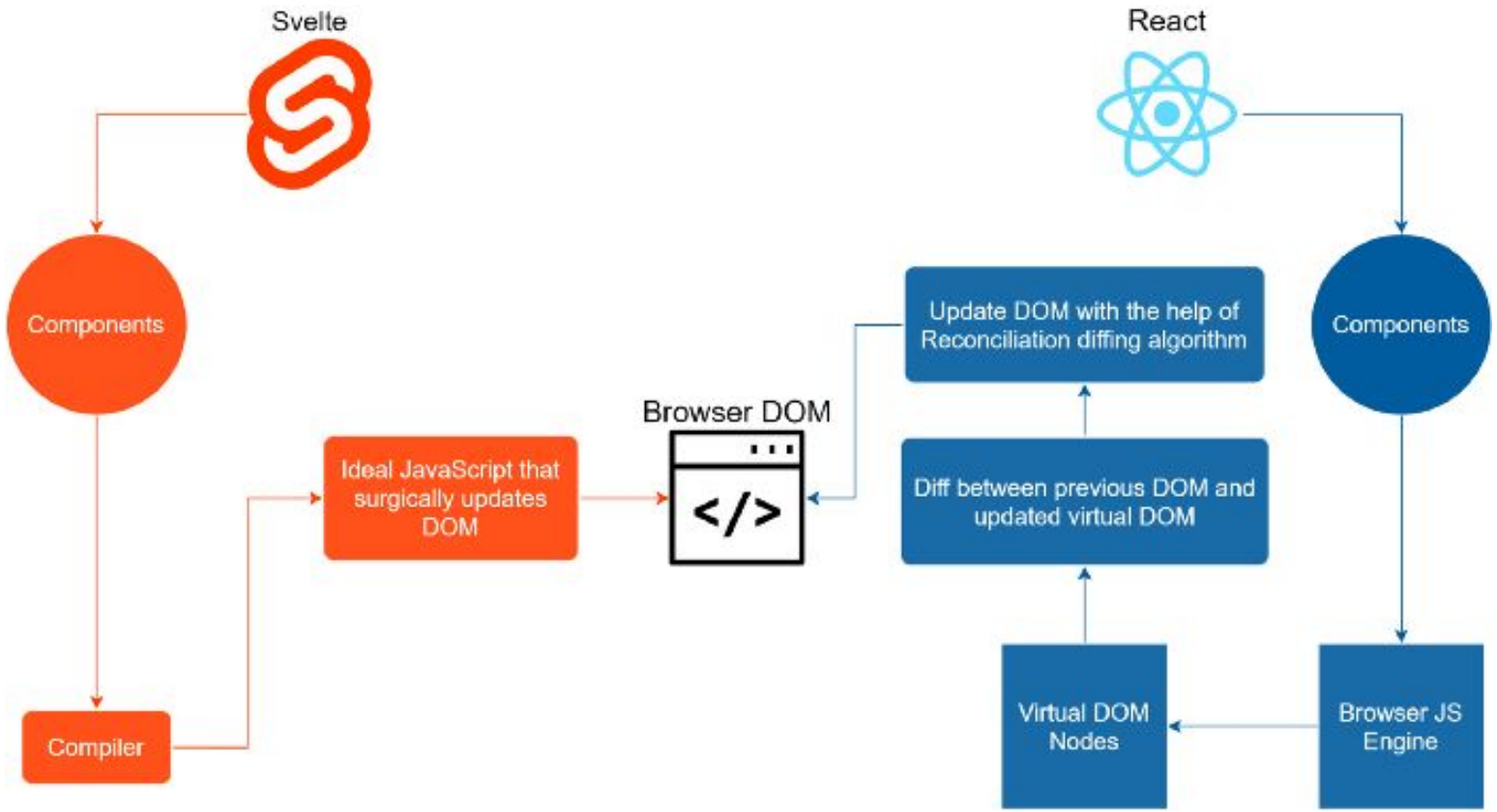
Why do people like it?
(Svelte vs. React)

- **Compiled** framework
- **Small bundle** size, optimized to be **lightweight** and fast (great for mobile)
- Very easy to learn and get started
- **High popularity**, growing community and support
- Maintained by core developers and the **community**



- **Virtual DOM**
- **Large bundle** size, but intended to provide lots of **features** and capabilities
- Somewhat easy to learn (hooks are weird)
- **Extremely popular**, huge community and support
- Maintained by **Meta**

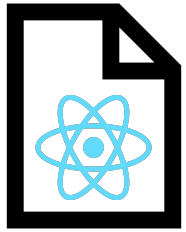
More Comparisons – **Virtual vs. Real DOM**



More Comparisons – **Components**

.svelte

Looks more familiar, like a regular **HTML file**

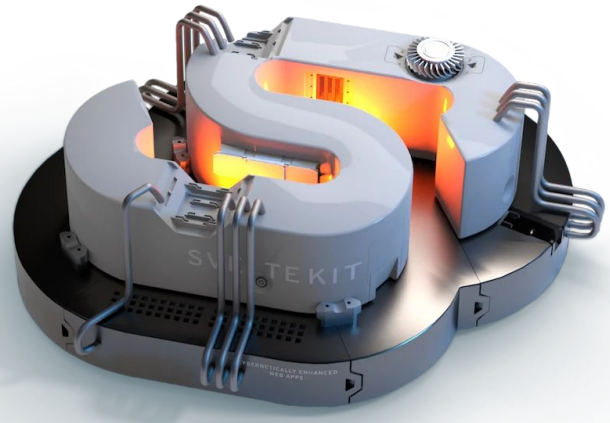


.js/ts/jsx/tsx

```svelte
<script>
  // Script goes here
</script>

<main>
  <!-- Markup (zero or more items) goes here -->
  This is an example Svelte component
</main>

<style>
  /* Styles go here */
</style>
```

```jsx
import React from 'react';

const Component = () => {
  // Script content goes here

  // Styling usually has to be written in a separate file

  // HTML/JSX returned here
  return <div>This is a React component</div>;
};

export default Component;
```

SvelteKit



SvelteKit is a “metaframework” built for Svelte, allowing you to create **full-stack** projects easily

(Similar to Next.js for React)

Adds many **features** required in a **modern web application**:

- Routing
- Error handling
- Data fetching
- Production build optimization
- Server-side rendering
- Environment variables

Feel free to **follow along** or **read the outline** in the `resources/` directory!

SVELTE(KIT) WALKTHROUGH

```
<cd out of catbook-react>
git clone <clone-uri>
cd svelte-workshop
git checkout step1
```

weblab.is/svelte

Home

Click me!

5

STEP 1

Creating a SvelteKit App

```
git reset --hard
git checkout step1
```

Step 1: SvelteKit Project Creation

```
git checkout step1
```

Tasks:

1. Create Svelte App
2. Install dependencies
3. Update `.prettierrc` and format code
4. Start the website

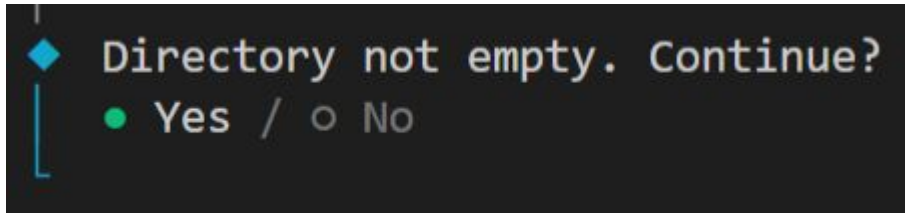
Step 1: SvelteKit Project Creation

```
git checkout step1
```

1. Create Svelte App – SvelteKit provides a **creation tool** through npm

```
npm create svelte@latest .
```

(Enter to use current directory)



Step 1: SvelteKit Project Creation

```
git checkout step1
```

```
◆ Which Svelte app template?
  ○ SvelteKit demo app
  ● Skeleton project (Barebones scaffolding for your new SvelteKit app)
  ○ Library project
```

```
◆ Add type checking with TypeScript?
  ○ Yes, using JavaScript with JSDoc comments
  ○ Yes, using TypeScript syntax
  ● No
```

```
◆ Select additional options (use arrow keys/space bar)
  □ Add ESLint for code linting
  ■ Add Prettier for code formatting
  □ Add Playwright for browser testing
  □ Add Vitest for unit testing
  □ Try the Svelte 5 preview (unstable!)
```

Step 1: SvelteKit Project Creation

```
git checkout step1
```

2. Install dependencies

```
npm i
```

3. Update `.prettierrc` (optional) and format code

```
npm run format
```

4. Start the website, open **`localhost:5173`**

```
npm run dev
```

Step 1: SvelteKit Project Creation

```
git checkout step1
```

(localhost:5173)

STEP 2

Setup and Simple Counter

```
git reset --hard
git checkout step2 --force
```

Step 2: The Basics

```
git checkout step2
```

Tasks:

1. Install VSCode Tools
2. Understand the basics of **Svelte components**
3. Create a simple counter app

routes/+page.svelte

Step 2: The Basics

```
git checkout step2
```

1. Install VSCode Tools

Official recommended **editing environment**

+

Svelte Component Basics

```
git checkout step2
```

- Svelte components consist of **3 sections**: scripts, markup, and styles
- **Logic** (JS) goes in the **`<script>`** tag
  - Use the **`let`** keyword to automatically create **state**
- **Styles** (CSS) go in the **`<style>`** tag
  - Styles are **scoped** to the work only in the component
- **Markup** (HTML) goes anywhere outside of these tags
  - Use brackets { } to include non-HTML (**like JSX!**)

```
<script>
  // Script goes here
</script>

<main>
  <!-- Markup (zero or more items) goes here -->
  This is an example Svelte component
</main>

<style>
  /* Styles go here */
</style>
```

Step 2: The Basics

```
git checkout step2
```

3. Create a simple counter app

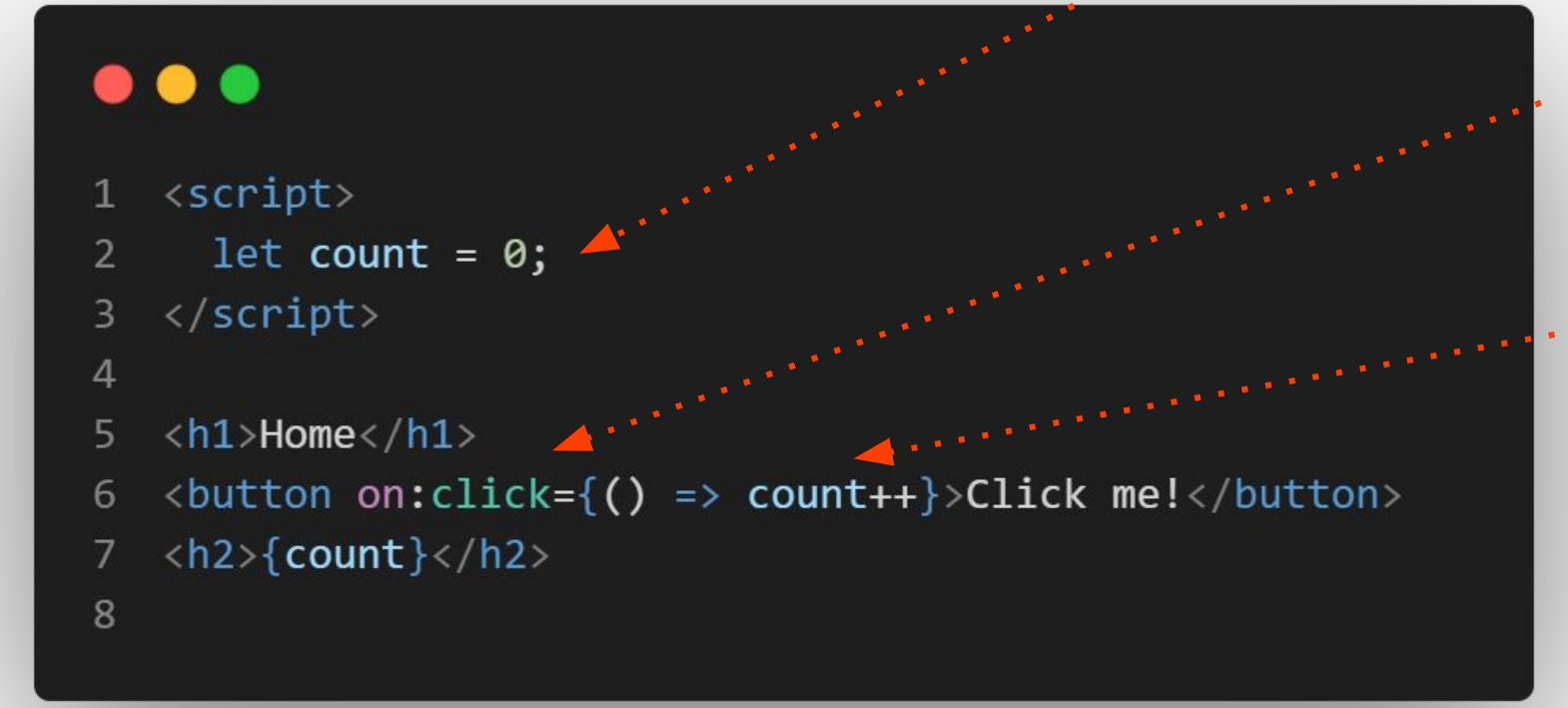
routes/+page.svelte

Step 2: The Basics

```
git checkout step2
```

3. Create a simple counter app

*Initializing Mutable **State***



*On click **binding***

Callback function
*that **updates** state*

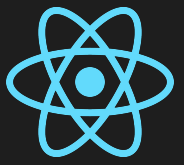
Step 2: The Basics

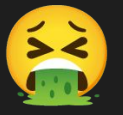
```
git checkout step2
```

Svelte vs. React 😱

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

<h1>Home</h1>
<button on:click={() => count++}>Click me!</button>
<h2>{count}</h2>

```

```jsx
import React, { useState } from 'react';

const Component = () => {
  const [count, setCount] = useState(0);

  return (
    <>
    <h1>Home</h1>
    <button onClick={() => setCount(count + 1)}>
      Click me!
    </button>
    <h2>{count}</h2>
    </>
  )
};

export default Component;
```





STEP 3

File-Based Routing

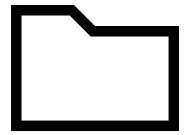
```
git reset --hard
git checkout step3
```

Step 3: File based routing

```
git checkout step3
```

Tasks:

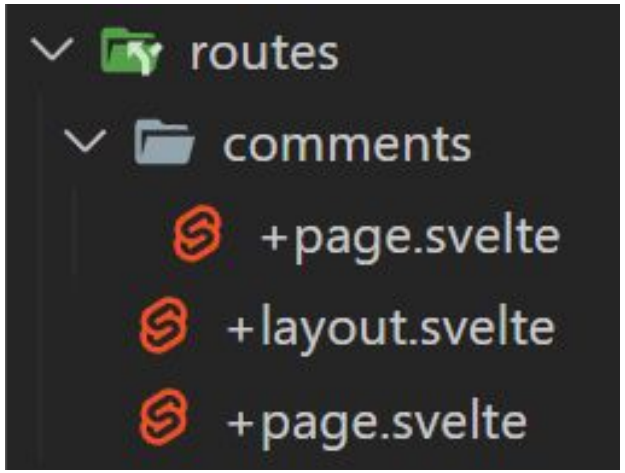
1. Understand **file based routing** with pages (`+page.svelte`)
2. Add a new page



routes/

File Based Routing – **Pages**

```
git checkout step3
```



example.com/comments

example.com/

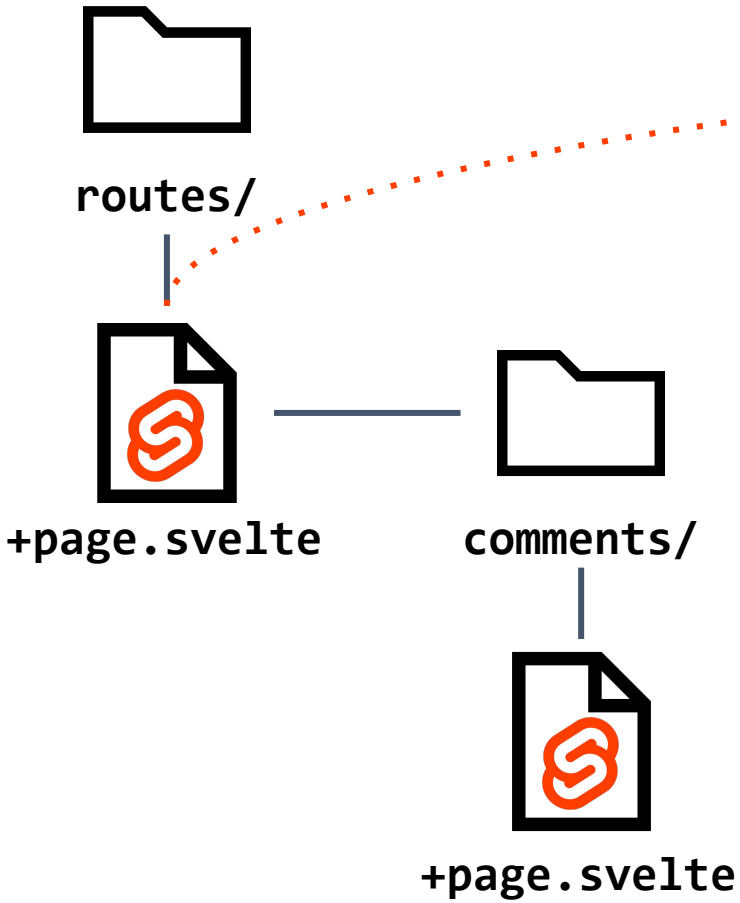
No need for separate routing libraries!

+page.svelte

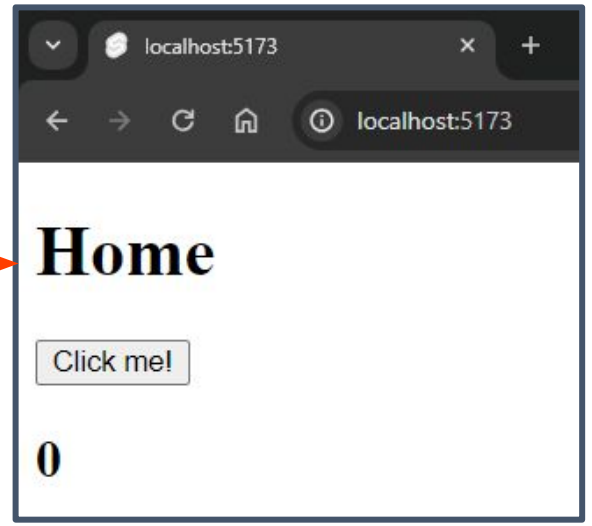
Step 3: File based routing

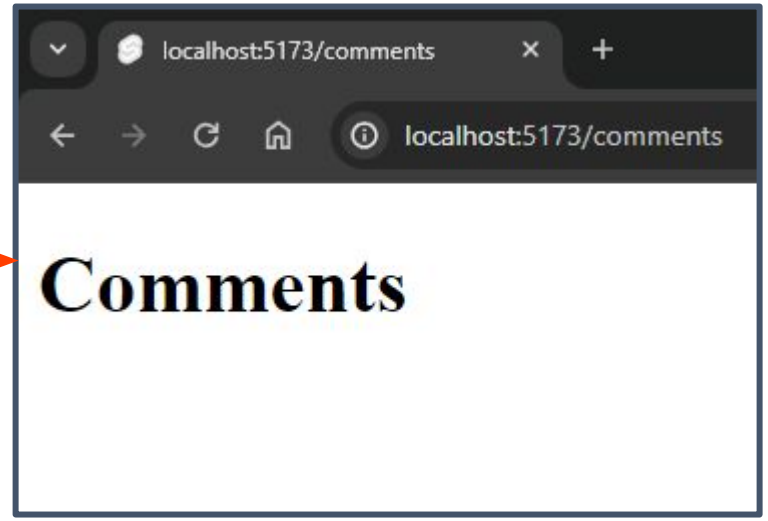
```
git checkout step3
```

2. Add a new page called **Comments**




```
1  <h1>Comments</h1>
2
```





STEP 4

Layouts and add Navbar

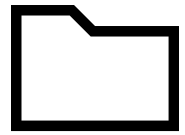
```
git reset --hard
git checkout step4 --force
```

Step 4: Layouts and Navbar

```
git checkout step4
```

Tasks:

1. Understand **file based routing** with **layouts** (`+layout.svelte`)
2. Add a **navbar** component
3. Add a global CSS file



routes/

File Based Routing – **Layouts**

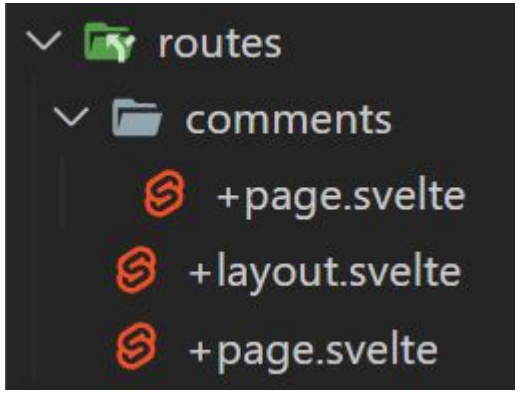
```
git checkout step4
```

`+layout.svelte`

Any content you want to **keep** across different sub-routes

```
1  <Navbar />
2  <slot />
```

Special `<slot>` tag, which displays the content of the **other routes**



`<Navbar>` component will be shown on both `/` and `/comments`

Tip: Global CSS can be imported in a `+layout.svelte` file (layouts explained in next step)

Step 4: Layouts and Navbar

```
git checkout step4
```

2. Create and add a navbar component

Navbar.svelte

+layout.svelte

Tip: Svelte recommends putting components in `lib/components`

Import the file from anywhere using:

```
import ComponentX from '$lib/components/ComponentX.svelte';
```

Step 4: Layouts and Navbar

```
git checkout step4
```

2. **Create** and **add** a navbar component

Navbar.svelte

```svelte
<ul>
  <li><a href="/">Home</a></li>
  <li><a href="/comments">Comments</a></li>
</ul>

<style>
  ul {
    display: flex;
    justify-content: space-evenly;
    background-color: aliceblue;
    padding: 20px;
  }
</style>
```

routes/+layout.svelte

```svelte
<script>
  import Navbar from '$lib/components/Navbar.svelte';
</script>

<Navbar />
<slot />
```

Step 4: Layouts and Navbar

```
git checkout step4
```

3. Add a global CSS file (import in `+layout.svelte`)

app.css

+layout.svelte

```css
body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

li {
  list-style: none;
}
```

STEP 5

Props and Loop Rendering

```
git reset --hard
git checkout step5 --force
```

Step 5: Props and Loop Rendering

```
git checkout step5
```

Tasks:

1. Understand **props**
2. Create a `Comment` component
3. Understand **loop rendering**
4. Render comments with **loop rendering**

`Comment.svelte`

`comments/`
`+page.svelte`

Props in Svelte

```
git checkout step5
```

- Pass in props using **`propName={propValue}`** in Component tags
- Take in props by using the syntax: **`export let propName;`**
- Reference the props in your markup by simply including them in JavaScript mode **`{}`**

Parent.svelte

```
1  ...
2  <main>
3    <ComponentX name={"señor cat"} />
4  </main>
```

Child.svelte

```
1  <script>
2    export let propA;
3  </script>
4
5  <main>
6    {propA}
7  </main>
```

One way street! Parent → Child

Step 5: Props and Loop Rendering

```
git checkout step5
```

2. Create a **generic component** for individual comments using props

Comment.svelte

Hint: Comments can have `id`, `user`, `title`, and `body` properties

Hint: Remember that components go in `lib/components`

Step 5: Props and Loop Rendering

```
git checkout step5
```

2. Create a **generic component** for individual comments using props

Comment.svelte

```svelte
<script>
  export let id;
  export let user;
  export let title;
  export let body;
</script>
```

```svelte
<article>
  <h3>
    {user}
  </h3>
  <h2>{title}</h2>
  <p>{body}</p>
</article>
```

```svelte
<style>
  article {
    background-color: lightblue;
    padding: 20px;
    border-radius: 20px;
    margin-bottom: 20px;
  }
</style>
```

Loop Rendering in Svelte

```
git checkout step5
```

Example.svelte

```svelte
<script>
  let items = ['a', 'b', 'c', 'd'];
</script>

<ul>
  {#each items as item (item)}
    <li>{item}</li>
  {/each}
</ul>
```

*Make sure to include a **key***

If you have an **array** of items you want to render, use an **#each** block

Step 5: Props and Loop Rendering

```
git checkout step5
```

4. Render a list of dummy comments using loop rendering in the Comments page

comments/
+page.svelte

```svelte
<script>
  import Comment from '$lib/components/Comment.svelte';

  let comments = [
    {
      id: 0,
      user: 'Username',
      title: 'Comment',
      body: 'Lorem ipsum dolor sit amet...',
    },
  ];
</script>
```

Hint: Use this script as a starting point. Write the markup!

Step 5: Props and Loop Rendering

```
git checkout step5
```

4. Render a list of dummy comments using loop rendering in the Comments page

comments/
+page.svelte

```svelte
<script>
  import Comment from '$lib/components/Comment.svelte';

  let comments = [
    {
      id: 0,
      user: 'Username',
      title: 'Comment',
      body: 'Lorem ipsum dolor sit amet...',
    },
  ];
</script>
```

```svelte
<h1>Comments</h1>
<ul>
  {#each comments as comment (comment.id)}
    <li>
      <Comment {...comment} />
    </li>
  {/each}
</ul>
```

Loops and Conditional Rendering

```
git reset --hard

git checkout step3
```

If you want to show only some content depending on a **condition**, use an **`#if`** or (**`#if :else`**) block

```
1  <script>
2    let user = { loggedIn: false };
3  </script>
4
5  {#if user.loggedIn}
6    <div class="content" />
7  {:else}
8    <h1>Please log in to see the content!</h1>
9  {/if}
```

If you have an **array** of items you want to display, use an **`#each`** block

*Make sure to include a **key***

```
1  <script>
2    let items = ['a', 'b', 'c', 'd'];
3  </script>
4
5  <ul>
6    {#each items as item (item)}
7      <li>{item}</li>
8    {/each}
9  </ul>
```

STEP 6

Use a Global Store

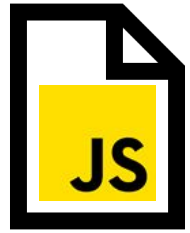
```
git reset --hard
git checkout step6 --force
```

Step 6: Use a global store

```
git checkout step6
```

Tasks:

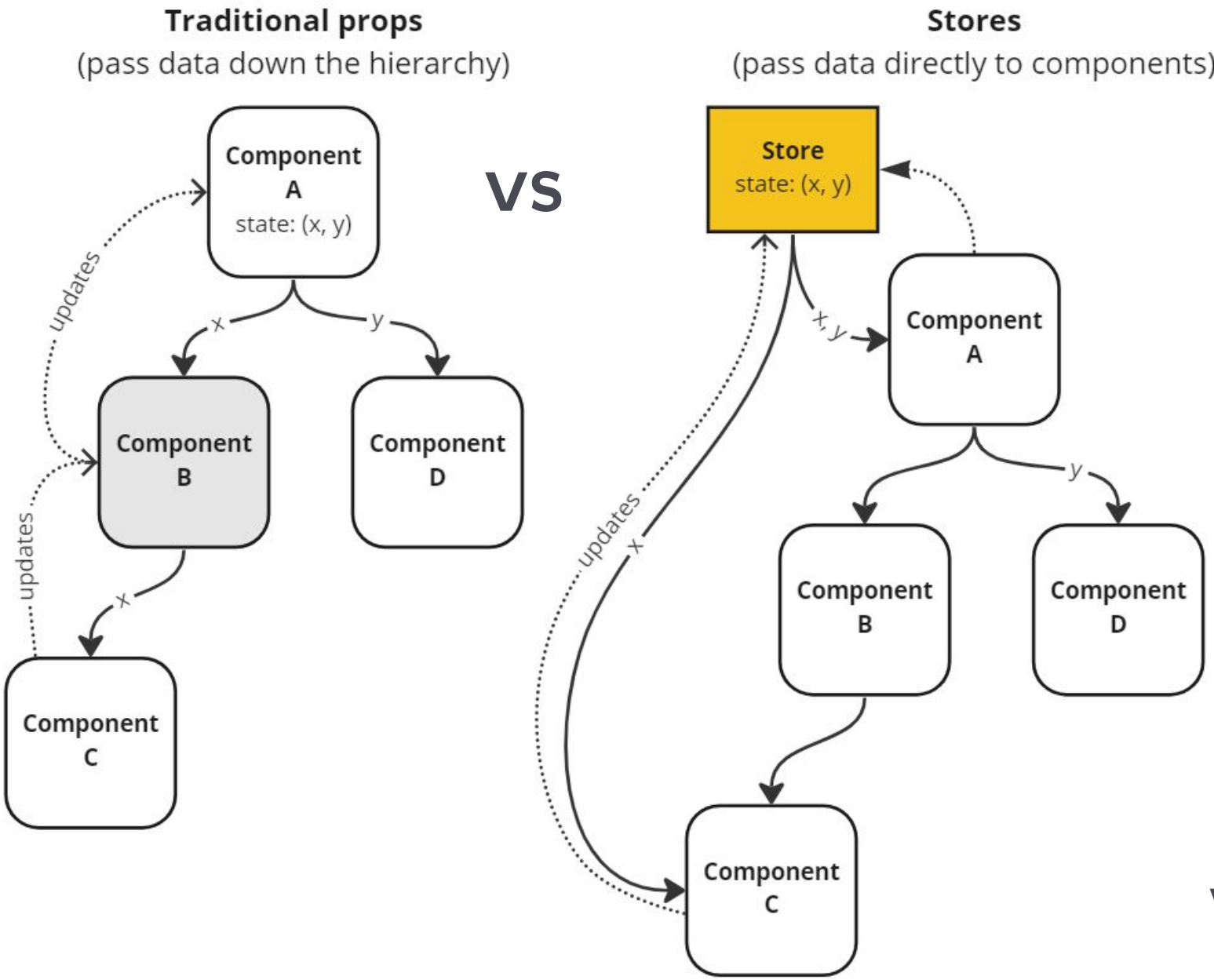
1. Understand **global writable stores**
2. Create a global store for the comments data
3. Replace the local comments array in `routes/comments/+page.svelte` with the comments data from the global store



`lib/stores/store.js`

`comments/+page.svelte`

```
git checkout step6
```



Writable Stores

Setting up a global store

```
git checkout step6
```

Implementation:

a. Create the store in its own *JavaScript* file and **export** it

```
1  import { writable } from 'svelte/store';
2
3  export const DataStore = writeable('initial data');
```

b. Import the store into your component and access it's value with **`$StoreName`** *(Svelte will automatically rerender when updated!)*

```
import { DataStore } from '$lib/stores/store';

$DataStore
```

Step 6: Use a global store

```
git checkout step6
```

2. Create a global store for the comments data

lib/stores/store.js

3. Replace the local comments array in `routes/comments/+page.svelte` with the comments data from the global store

routes/comments/+page.svelte

Hint: Don't forget to import!

Step 6: Use a global store

```
git checkout step6
```

2. Create a global store for the comments data

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

export const CommentsStore = writable([
  {
    id: 0,
    user: 'Username',
    title: 'Comment',
    body: 'Lorem ipsum dolor...',
  },
]);
```

3. Replace the local comments array in `routes/comments/+page.svelte` with the comments data from the global store

```
{#each $CommentsStore as comment (comment.id)}
```

STEP 7

Add New Comments

```
git reset --hard
git checkout step7 --force
```

Step 7: Add new comments

```
git checkout step7
```

comments/
+page.svelte

Tasks:

1. Understand **update stores** and **input binding** (reading from user input)
2. Create a “new comment” form that adds new comments

Updating Stores and Binding

```
git checkout step7
```

To update a store's value, simply import the store and call its **`update`** function!

```
<script>
  import { DataStore } from 'path/to/store';

  // Updates the store to hold the new value
  DataStore.update((prevValue) => {
    return newValue;
  });
</script>
```

To keep track of **user input**, use **`bind:value={someState}`**, which saves input to state automatically

```
<script>
  // Whatever is typed in the <input/> box
  // will be reflected here
  let inputValue = '';
</script>

<input type="text" bind:value={inputValue} />
<h1>{inputValue}</h1>
```

Step 7: Add new comments

```
git checkout step7
```

routes/comments/
+page.svelte

2. Create a “new comment” form that adds new comments

Hint: Think about how you’d do this in React and just change the syntax!

Hint: Use a **`<form>`** HTML element and the **`on:submit`** binding

Step 7: Add new comments

```
git checkout step7
```

routes/comments/
+page.svelte

2. Create a “new comment” form that adds new comments

```
<script/>

let newCommentTitle = '';

const addComment = () => {
  const newComment = {
    id: Math.floor(Math.random() * 10000),
    user: 'Temp',
    title: newCommentTitle,
    body: 'Lorem ipsum dolor...',
  };

  CommentsStore.update((prevStore) => [newComment, ...prevStore]);

  newCommentTitle = '';
};
```

```
<html/>

<form on:submit={addComment}>
  <input type="text" bind:value={newCommentTitle} />
  <button type="submit">Add Comment</button>
</form>
```

STEP 8

Delete Comments

```
git reset --hard
git checkout step8
```

Step 8: Delete comments

```
git checkout step8
```

`Comment.svelte`

Tasks:

1. Add a "**delete**" button on *each* comment. When the button is clicked, **remove** that comment from the array of comments (comment should disappear).

Hint 1: You only need to modify **`Comment.svelte`** (take advantage of the global store!)

Hint 2: Use the **`filter`** function on the comments array

Step 8: Delete comments

```
git checkout step8
```

1. Add a "**delete**" button on *each* comment. When the button is clicked, **remove** that comment from the array of comments (comment should disappear).

Comment.svelte


```
import { CommentsStore } from '$lib/stores/store';

const deleteComment = () => {
  CommentsStore.update((prevStore) => prevStore.filter((comment) => comment.id !== id));
};
```



```
<button on:click={deleteComment}>Delete {id}</button>
```

STEP 9

Add animations

```
git reset --hard
git checkout step9
```

Step 9: Add animations

```
git checkout step9
```

Tasks:

1. Animate a **comment** when it is created/deleted
2. Add **page transition animations** when switching from the home page to the comments page

`Comment.svelte`

`comments/+page.svelte`

Step 9: Add animations

```
git checkout step9
```

Tasks:

1. Animate a **comment** when it is created/deleted

```
transition:fly={{ x: -200 }}
```

2. Add **page transition animations** when switching from the home page to the comments page

Questions?

```
git reset --hard
git checkout complete
```

weblab.is/svelte

More resources in `resources/outline.md`

Announcements

- **Homework 3** due before lecture tomorrow: weblab.is/homework3
- **Office Hours** TONIGHT 6 PM - 9 PM
- **Hackathon** 7 PM - 1 AM tomorrow (Friday) night!
  - In 32-082
  - There will be food and drinks!
- **Come to lecture tomorrow!** Learn how to deploy your website, Render sponsor lecture (raffling two $20 Amazon gift cards), sendoff
- **Milestone 2 (Minimum Viable Product)** due next Wednesday (Jan 24) 6:00 PM!

Stay for Victory's lecture on recruiting in industry!