Reactivity in the Web with Svelte

About Me

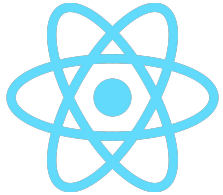
- Brendan Todahl
- Senior Consultant @ CGI
- Graduated from THE Ohio State University
- Running, Golf, OSU Football
- No blog 😭

Lots of code incoming

https://github.com/BrendanTodahl/SvelteReactivity

Virtual DOM & DOM Diffing

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

export default function App() {
  const [count, setCount] = useState(0)
  const [name, setName] = useState('World')

  function incrementCount() {
    setCount(count + 1)
  }

  function nameChange(event) {
    setName(event.target.value)
  }

  return (
    <div className="App">
      <h1>Hello {name}!</h1>
      <input type="text" value={name} onInput={nameChange} />
      <button onClick={incrementCount}>Clicks: {count}</button>
    </div>
  )
}
```



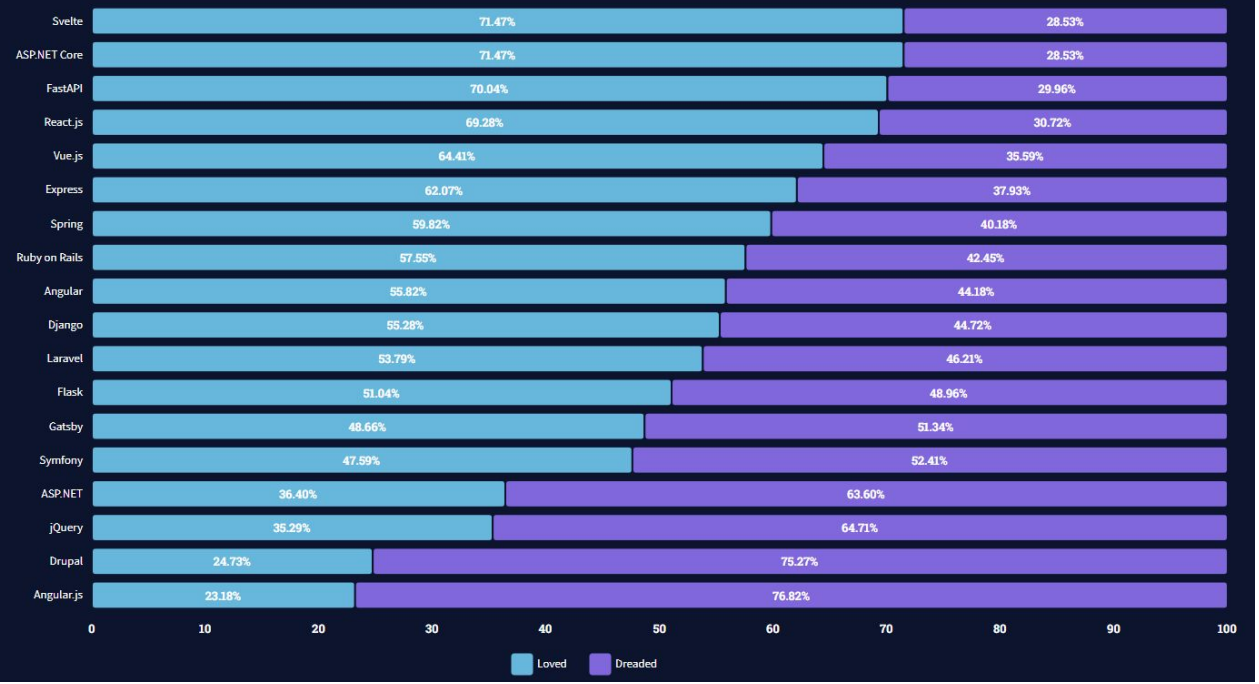
Can We Do Better?

Web frameworks

Newcomer Svelte takes the top spot as the most loved framework. React is the most wanted, desired by one in four developers.

Loved vs. Dreaded | Want

66,202 responses



SVELTE

adjective **/svelt/** attractively thin, graceful and stylish

Svelte Is a Compiler

App Size

“A React component is typically around 40% larger than its Svelte equivalent.”

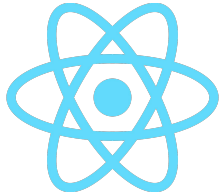
- Rich Harris

Did I mention a lot of code?

Virtual DOM & DOM Diffing

Virtual DOM & DOM Diffing

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

export default function App() {
  const [count, setCount] = useState(0)
  const [name, setName] = useState('World')

  function incrementCount() {
    setCount(count + 1)
  }

  function nameChange(event) {
    setName(event.target.value)
  }

  return (
    <div className="App">
      <h1>Hello {name}!</h1>
      <input type="text" value={name} onInput={nameChange} />
      <button onClick={incrementCount}>Clicks: {count}</button>
    </div>
  )
}
```



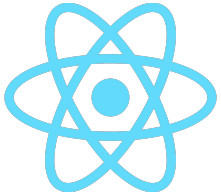
Virtual DOM & DOM Diffing

```vue
<template>
  <div>
    <h1>Hello {{ name }}!</h1>
    <input type="text" v-model="name" />
    <button @click="incrementCount">Clicks: {{ numClicks }}</button>
  </div>
</template>

<script>
export default {
  name: 'HelloWorld',
  methods: {
    incrementCount () {
      this.numClicks++
    }
  },
  data () {
    return {
      name: 'World',
      numClicks: 0
    }
  }
}
</script>
```

Virtual DOM & DOM Diffing

```
<script>
  let name = 'World'
  let count = 0

  function nameChange(event) {
    name = event.target.value
  }

  function incrementCount() {
    count += 1
  }
</script>

<h1>Hello {name}!</h1>
<input type="text" value="{name}" on:input={nameChange} />
<button on:click="{incrementCount}">Clicks: {count}</button>
```

```svelte
<script>
  let name = 'World'
  let count = 0
</script>

<h1>Hello {name}!</h1>
<input type="text" bind:value="{name}" />
<button on:click="{() => count += 1}">Clicks: {count}</button>
```



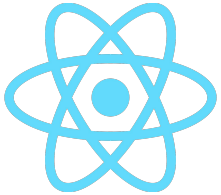
```jsx
import React, { useState } from 'react'

export default function App() {
  const [count, setCount] = useState(0)
  const [name, setName] = useState('World')

  function incrementCount() {
    setCount(count + 1)
  }

  function nameChange(event) {
    setName(event.target.value)
  }

  return (
    <div className="App">
      <h1>Hello {name}!</h1>
      <input type="text" value={name} onInput={nameChange} />
      <button onClick={incrementCount}>Clicks: {count}</button>
    </div>
  )
}
```

```svelte
<script>
  let name = 'World'
  let count = 0
</script>

<h1>Hello {name}!</h1>
<input type="text" bind:value="{name}" />
<button on:click="{() => count += 1}">Clicks: {count}</button>
```

```vue
<template>
  <div>
    <h1>Hello {{ name }}!</h1>
    <input type="text" v-model="name" />
    <button @click="incrementCount">Clicks: {{ numClicks }}</button>
  </div>
</template>

<script>
export default {
  name: 'HelloWorld',
  methods: {
    incrementCount () {
      this.numClicks++
    }
  },
  data () {
    return {
      name: 'World',
      numClicks: 0
    }
  }
}
</script>
```

What’s Wrong with the Virtual DOM?

Another Todo List App? Really?

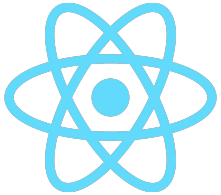


```
import React, { useState } from 'react'

export default function TodoList() {
  const [todos, setTodos] = useState([
    { done: false, text: 'Eat' },
    { done: false, text: 'Sleep' },
    { done: false, text: 'Learn Svelte' },
    { done: false, text: 'Repeat' }
  ])

  function toggleDone(t) {
    setTodos(todos.map(todo => {
      if (todo === t) return { done: !t.done, text: t.text }
      return todo
    }))
  }

  const [hideDone, setHideDone] = useState(false)

  function toggleHideDone() {
    setHideDone(!hideDone)
  }

  const filtered = hideDone
    ? todos.filter(todo => !todo.done)
    : todos

  return (
    <div>
      <label>
        <input type="checkbox" class="me-1" checked={hideDone} onChange={toggleHideDone} />
        Hide Done - Showing {filtered.length} of {todos.length}
      </label>

      <ul>
        {filtered.map(todo => (
          <li onClick={() => toggleDone(todo)}>
            {todo.done ? 'X' : ''} {todo.text}
          </li>
        ))}
      </ul>
    </div>
  )
}
```

☐ Hide Done

- X Eat
- Sleep
- Learn Svelte
- Repeat

Another Todo List App? Really?



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

export default function TodoList() {
  const [todos, setTodos] = useState([
    { done: false, text: 'Eat' },
    { done: false, text: 'Sleep' },
    { done: false, text: 'Learn Svelte' },
    { done: false, text: 'Repeat' }
  ])

  function toggleDone(t) {
    setTodos(todos.map(todo => {
      if (todo === t) return { done: !t.done, text: t.text }
      return todo
    }))
  }

  const [hideDone, setHideDone] = useState(false)

  function toggleHideDone() {
    setHideDone(!hideDone)
  }

  const filtered = hideDone
    ? todos.filter(todo => !todo.done)
    : todos

  return (
    <div>
      <label>
        <input type="checkbox" class="me-1" checked={hideDone} onChange={toggleHideDone} />
        Hide Done - Showing {filtered.length} of {todos.length}
      </label>

      <ul>
        {filtered.map(todo => (
          <li onClick={() => toggleDone(todo)}>
            {todo.done ? 'X' : ''} {todo.text}
          </li>
        ))}
      </ul>
    </div>
  )
}
```

```
const filtered = hideDone
    ? todos.filter(todo => !todo.done)
    : todos
```

```
const filtered = useMemo(
    () => hideDone
      ? todos.filter(todo => !todo.done)
      : todos,
    [todos, hideDone]
)
```

Reactive Programming

```
let a = 1;
let b = a + 1;
a = 10;
let equal = b === 11; // FALSE

b = a + 1;
equal = b === 11; // TRUE
```

Labeled Statements

```
let a = 1;
$: b = a + 1;
a = 10;
let equal = b === 11; // TRUE
```

Todo List App First Attempt

```
<script>
  let todos = [
    { done: false, text: 'Eat' },
    { done: false, text: 'Sleep' },
    { done: false, text: 'Learn Svelte' },
    { done: false, text: 'Repeat' }
  ]

  function toggleDone(t) {
    todos = todos.map(todo => {
      if (todo === t) return { done: !t.done, text: t.text }
      return todo
    })
  }

  let hideDone = false

  const filtered = hideDone
    ? todos.filter(todo => !todo.done)
    : todos
</script>

<label>
  <input type="checkbox" bind:checked={hideDone}>
  Hide Done
</label>

<ul>
  {#each filtered as todo}
    <li on:click={() => toggleDone(todo)}>
      {todo.done ? 'X' : ''} {todo.text}
    </li>
  {/each}
</ul>
```

Todo List App Complete

```svelte
<script>
  let todos = [
    { done: false, text: 'Eat' },
    { done: false, text: 'Sleep' },
    { done: false, text: 'Learn Svelte' },
    { done: false, text: 'Repeat' }
  ]

  function toggleDone(t) {
    todos = todos.map(todo => {
      if (todo === t) return { done: !t.done, text: t.text }
      return todo;
    })
  }

  let hideDone = false

  $: showing = filtered.length

  $: filtered = hideDone
    ? todos.filter(todo => !todo.done)
    : todos
</script>

<label>
  <input type="checkbox" bind:checked={hideDone}>
  Hide Done
</label>

<span>Showing {showing} of {todos.length}</span>

<ul>
  {#each filtered as todo}
    <li on:click={() => toggleDone(todo)}>
      {todo.done ? 'X' : ''} {todo.text}
    </li>
  {/each}
</ul>
```

Conditionals & List Generating

Conditionals & List Generating

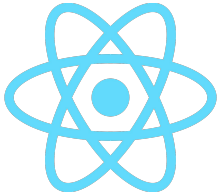
```
<script>
  let selectedColor = ''

  function setColor(color) {
    selectedColor = color
  }
</script>

<button on:click="{() => setColor('Red')}">Red</button>
<button on:click="{() => setColor('Blue')}">Blue</button>
<button on:click="{() => setColor('Green')}">Green</button>
<span>Color is: <strong>{selectedColor}</strong></span>
```

Red Blue Green Color is: **Red**

```svelte
<script>
  let selectedColor = ''

  function setColor(color) {
    selectedColor = color
  }
</script>

<button on:click="{() => setColor('Red')}">Red</button>
<button on:click="{() => setColor('Blue')}">Blue</button>
<button on:click="{() => setColor('Green')}">Green</button>
<span>Color is: <strong>{selectedColor}</strong></span>
```

```vue
<template>
  <div>
    <button @click="setColor('Red')">Red</button>
    <button @click="setColor('Blue')">Blue</button>
    <button @click="setColor('Green')">Green</button>
    <span>Color is: <strong>{{ selectedColor }}</strong></span>
  </div>
</template>

<script>
export default {
  name: 'ColorList1',
  methods: {
    setColor (color) {
      this.selectedColor = color
    }
  },
  data () {
    return {
      selectedColor: ''
    }
  }
}
</script>
```

```svelte
<script>
  let selectedColor = ''

  function setColor(color) {
    selectedColor = color
  }
</script>

<button on:click="{() => setColor('Red')}">Red</button>
<button on:click="{() => setColor('Blue')}">Blue</button>
<button on:click="{() => setColor('Green')}">Green</button>
<span>Color is: <strong>{selectedColor}</strong></span>
```



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

export default function App() {
  const [color, setColor] = useState('')

  function handleClick(color) {
    setColor(color)
  }

  return (
    <div className="App">
      <button onClick={() => handleClick('Red')}>Red</button>
      <button onClick={() => handleClick('Blue')}>Blue</button>
      <button onClick={() => handleClick('Green')}>Green</button>
      <span>Color is: <strong>{color}</strong></span>
    </div>
  )
}
```

Conditionals & List Generating

```
<script>
  let colors = ['Red', 'Blue', 'Green']
  let selectedColor = ''

  function setColor(color) {
    selectedColor = color
  }
</script>

{#each colors as color}
  <button on:click="{() => setColor(color)}">{color}</button>
{/each}
<span>Color is: <strong>{selectedColor}</strong></span>

<br/>
{#if !selectedColor}
  <span>Pick a color!</span>
{:else if selectedColor === 'Red'}
  <span>That's my favorite color!</span>
{:else}
  <span>That's almost as cool as Red!</span>
{/if}
```

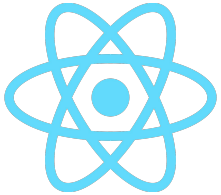
Red Blue Green Color is: **Red**

That's my favorite color!

```svelte
<script>
  let colors = ['Red', 'Blue', 'Green']
  let selectedColor = ''

  function setColor(color) {
    selectedColor = color
  }
</script>

{#each colors as color}
  <button on:click="{() => setColor(color)}">{color}</button>
{/each}
<span>Color is: <strong>{selectedColor}</strong></span>

<br/>
{#if !selectedColor}
  <span>Pick a color!</span>
{:else if selectedColor === 'Red'}
  <span>That's my favorite color!</span>
{:else}
  <span>That's almost as cool as Red!</span>
{/if}
```

```vue
<template>
  <div>
    <button v-for="color in colors" :key="color" @click="setColor(color)">{{ color }}</button>
    <span>Color is: <strong>{{ selectedColor }}</strong></span>

    <span class="d-block mt-2" v-if="!selectedColor">Pick a color!</span>
    <span class="d-block mt-2" v-else-if="selectedColor === 'Red'">That's my favorite color!</span>
    <span class="d-block mt-2" v-else>That's almost as cool as Red!</span>
  </div>
</template>

<script>
export default {
  name: 'ColorList2',
  methods: {
    setColor (color) {
      this.selectedColor = color
    }
  },
  data () {
    return {
      colors: ['Red', 'Blue', 'Green'],
      selectedColor: ''
    }
  }
}
</script>
```

```svelte
<script>
  let colors = ['Red', 'Blue', 'Green']
  let selectedColor = ''

  function setColor(color) {
    selectedColor = color
  }
</script>

{#each colors as color}
  <button on:click="{() => setColor(color)}">{color}</button>
{/each}
<span>Color is: <strong>{selectedColor}</strong></span>

<br/>
{#if !selectedColor}
  <span>Pick a color!</span>
{:else if selectedColor === 'Red'}
  <span>That's my favorite color!</span>
{:else}
  <span>That's almost as cool as Red!</span>
{/if}
```



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

export default function App() {
  const [color, setColor] = useState('')

  function handleClick(color) {
    setColor(color)
  }

  function ColorList(props) {
    const colors = props.colors || []
    const listItems = colors.map((color) =>
      <button onClick={() => handleClick(color)}>{color}</button>
    )

    return (
      <span>{listItems}</span>
    )
  }

  const colors = ['Red', 'Blue', 'Green']

  return (
    <div className="App">
      <ColorList colors={colors} />
      <span>Color is: <strong>{color}</strong></span>

      <br/>

      {!color && <span>Pick a color!</span>}
      {color === 'Red' && <span>That's my favorite color!</span>}
      {color && color !== 'Red' && <span>That's almost as cool as Red!</span>}
    </div>
  )
}
```

Components

Components

```svelte
<!-- index.svelte -->
<script>
  import ChildComponent from './childComponent.svelte'
</script>

<p>This paragraph is purple!</p>
<ChildComponent number={100}/>

<style>
  p {
    color: purple;
    font-size: 2em;
  }
</style>
```

```svelte
<!-- childComponent.svelte -->
<script>
  export let number = 0
</script>

<p>This paragraph is not purple.</p>
<span>The value of number is: {number}</span>
```

This paragraph is purple!

This paragraph is not purple.

The value of number is: 100

```svelte
<!-- index.svelte -->
<script>
  import ChildComponent from './childComponent.svelte'
</script>

<p>This paragraph is purple!</p>
<ChildComponent number={100}/>

<style>
  p {
    color: purple;
    font-size: 2em;
  }
</style>
```

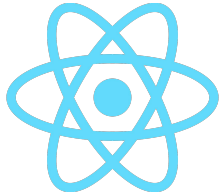
```svelte
<!-- childComponent.svelte -->
<script>
  export let number = 0
</script>

<p>This paragraph is not purple.</p>
<span>The value of number is: {number}</span>
```

```vue
<!-- Component.vue -->
<template>
  <div>
    <p>This paragraph is purple!</p>
    <child-component number="100" />
  </div>
</template>

<script>
import ChildComponent from "./components/ChildComponent.vue"

export default {
  name: "App",
  components: { ChildComponent }
}
</script>

<style scoped>
p {
  color: purple;
  font-size: 2em;
}
</style>
```

```vue
<!-- ChildComponent.vue -->
<template>
  <div>
    <p>This paragraph is not purple</p>
    <span>The value of number is: {{ number }}</span>
  </div>
</template>

<script>
export default {
  name: "ChildComponent",
  props: {
    number: {
      default: 0
    }
  }
}
</script>
```

```svelte
<!-- index.svelte -->
<script>
  import ChildComponent from './childComponent.svelte'
</script>

<p>This paragraph is purple!</p>
<ChildComponent number={100}/>

<style>
  p {
    color: purple;
    font-size: 2em;
  }
</style>
```

```svelte
<!-- childComponent.svelte -->
<script>
  export let number = 0
</script>

<p>This paragraph is not purple.</p>
<span>The value of number is: {number}</span>
```



```jsx
// components.js
import "./styles.css"
import ChildComponent from "./childComponent"

export default function App() {
  return (
    <div>
      <p class="purple">This paragraph is purple!</p>
      <ChildComponent number="100" />
    </div>
  )
}
```

```jsx
// childComponent.js
export default function ChildComponent(props) {
  return (
    <div>
      <p>This paragraph is not purple.</p>
      <span>The value of number is: {props.number || 0}</span>
    </div>
  )
}
```

```jsx
// global.css
p.purple {
  color: purple;
  font-size: 2em;
}
```

Data Stores

Data Stores

```
import { writable, readable, derived } from 'svelte/store'
```

```
export const count = writable(0)
```

```
export const time = readable(new Date(), function start(set) {
  // start code here
})
```

```
const start = new Date()
export const elapsed = derived(
  time,
  $time => Math.round(($time - start) / 1000)
)
```

Data Stores

```svelte
<!-- index.svelte -->
<script>
  import { color } from './stores.js'
  import ColorButtons from './colorButtons.svelte'

  let selectedColor

  color.subscribe(value => {
    selectedColor = value
  })
</script>

<ColorButtons></ColorButtons>

<h1>
  {selectedColor}
</h1>
```

```svelte
<!-- colorButtons.svelte -->
<script>
  import { color } from './stores.js'

  function setColor(selectedColor) {
    color.set(selectedColor);
  }
</script>

<button on:click="{() => setColor('red')}">red</button>
<button on:click="{() => setColor('blue')}">blue</button>
<button on:click="{() => setColor('green')}">green</button>
```

```svelte
// stores.js
import { writable } from 'svelte/store'
export const color = writable('Pick a color!')
```

red blue green

Pick a color!

Custom Stores

As long as it implements `subscribe()`, then it's a store.

```
// stores.js
import { writable } from 'svelte/store'
export const color = writable('Pick a color!')
```

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

function createColor() {
  const { subscribe, set } = writable('Pick a color!')

  return {
    subscribe,
    setColor: (color) => set(color)
  }
}

export const color = createColor()
```

Custom Stores

```svelte
<!-- index.svelte -->
<script>
  import { color } from './stores.js'
  import ColorButtons from './ColorButtons.svelte'

  let selectedColor

  color.subscribe(value => {
    selectedColor = value
  })
</script>

<ColorButtons></ColorButtons>

<h1>
  {selectedColor}
</h1>
```

```svelte
<!-- ColorButtons.svelte -->
<script>
  import { color } from './stores.js'
</script>

<button on:click="{() => color.setColor('red')}">red</button>
<button on:click="{() => color.setColor('blue')}">blue</button>
<button on:click="{() => color.setColor('green')}">green</button>
```

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

function createColor() {
  const { subscribe, set } = writable('Pick a color!')

  return {
    subscribe,
    setColor: (color) => set(color)
  }
}

export const color = createColor()
```

Performance

- https://krausest.github.io/js-framework-benchmark/current.html
- Shows benchmarks for numerous amount of current frameworks.

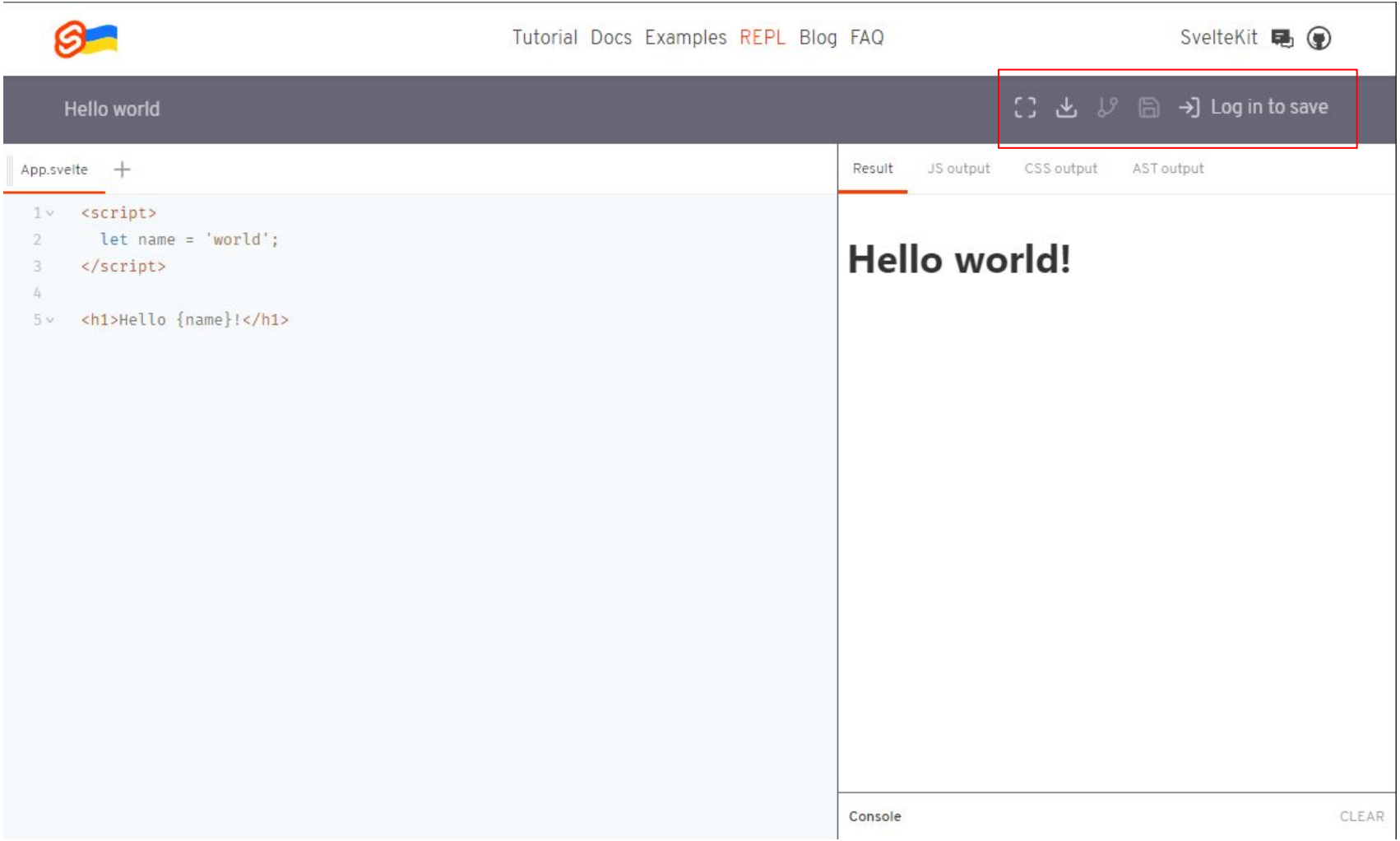
Keyed results

Keyed implementations create an association between the domain data and a dom element by assigning a 'key'. If data changes the dom element with that key will be updated. In consequence inserting or deleting an element in the data array causes a corresponding change to the dom.

Duration in milliseconds ± 95% confidence interval (Slowdown = Duration / Fastest)

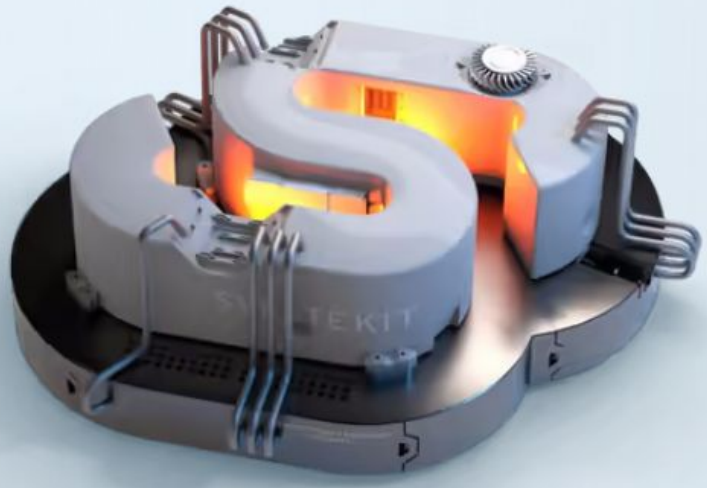
Name Duration for...	vanillajs	svelte-v3.46.2	vue-v3.2.26	react-hooks-v17.0.2	angular-v13.0.0
Implementation notes	772				
create rows creating 1,000 rows	75.6 ± 1.0 (1.00)	93.2 ± 1.3 (1.23)	105.2 ± 3.5 (1.39)	112.0 ± 2.7 (1.48)	115.6 ± 2.1 (1.53)
replace all rows updating all 1,000 rows (5 warmup runs).	75.9 ± 0.5 (1.00)	93.1 ± 0.4 (1.23)	90.8 ± 0.6 (1.20)	95.4 ± 0.4 (1.26)	101.7 ± 0.8 (1.34)
partial update updating every 10th row for 1,000 rows (3 warmup runs). 16x CPU slowdown.	160.0 ± 3.8 (1.00)	173.0 ± 3.2 (1.08)	184.1 ± 4.6 (1.15)	180.2 ± 3.0 (1.13)	166.5 ± 2.8 (1.04)
select row highlighting a selected row. (no warmup runs). 16x CPU slowdown.	19.3 ± 1.4 (1.00)	30.0 ± 1.1 (1.55)	33.2 ± 1.1 (1.72)	66.7 ± 2.2 (3.45)	58.7 ± 3.2 (3.04)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4x CPU slowdown.	46.3 ± 0.4 (1.00)	48.9 ± 0.4 (1.06)	48.9 ± 0.5 (1.06)	320.1 ± 1.3 (6.92)	333.3 ± 1.9 (7.20)
remove row removing one row. (5 warmup runs).	19.3 ± 0.4 (1.00)	20.1 ± 0.5 (1.04)	21.5 ± 0.2 (1.12)	21.2 ± 0.4 (1.10)	20.0 ± 0.5 (1.04)
create many rows creating 10,000 rows	762.7 ± 34.9 (1.00)	998.0 ± 33.6 (1.31)	1,008.6 ± 14.3 (1.32)	1,342.8 ± 45.6 (1.76)	1,108.8 ± 16.7 (1.45)
append rows to table appending 1,000 to a table of 1,000 rows. 2x CPU slowdown.	169.4 ± 1.9 (1.00)	206.1 ± 2.6 (1.22)	199.2 ± 4.1 (1.18)	226.7 ± 4.5 (1.34)	232.2 ± 2.2 (1.37)
clear rows clearing a table with 1,000 rows. 8x CPU slowdown.	48.7 ± 0.5 (1.00)	71.2 ± 0.9 (1.46)	66.8 ± 1.2 (1.37)	69.7 ± 1.3 (1.43)	144.9 ± 3.9 (2.97)
geometric mean of all factors in the table	1.00	1.23	1.26	1.79	1.88
compare: Green means significantly faster, red significantly slower	compare	compare	compare	compare	compare

How to Adopt Svelte?


Tutorial Docs Examples REPL Blog FAQ
SvelteKit
Hello world
Log in to save
App.svelte
<script>
let name = 'world';
</script>
<h1>Hello {name}!</h1>
Result
JS output
CSS output
AST output
Hello world!
Console
CLEAR

SVELTEKIT

THE FASTEST WAY TO
BUILD SVELTE APPS



https://svelte.dev/

Thank You