

# Svelte Dasar



Eko Kurniawan Khannedy

# Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 14+ years experiences
- [www.programmerzamannow.com](http://www.programmerzamannow.com)
- [youtube.com/c/ProgrammerZamanNow](https://youtube.com/c/ProgrammerZamanNow)



# Eko Kurniawan Khannedy

- Telegram : [@khannedy](#)
- Linkedin : <https://www.linkedin.com/company/programmer-zaman-now/>
- Facebook : [fb.com/ProgrammerZamanNow](https://fb.com/ProgrammerZamanNow)
- Instagram : [instagram.com/programmerzamannow](https://instagram.com/programmerzamannow)
- Youtube : [youtube.com/c/ProgrammerZamanNow](https://youtube.com/c/ProgrammerZamanNow)
- Telegram Channel : [t.me/ProgrammerZamanNow](https://t.me/ProgrammerZamanNow)
- Tiktok : <https://tiktok.com/@programmerzamannow>
- Email : echo.khannedy@gmail.com

# Sebelum Belajar

- Kelas HTMLS, CSS dan JavaScript
- Kelas NodeJS
- Kelas Vite

# Pengenalan

# Kenapa Butuh Framework?

- Mungkin menjadi pertanyaan, kenapa kita butuh framework untuk membuat web Frontend?
- Salah satu alasannya adalah agar ada standarisasi saat membuat project terutama ketika bekerja dalam tim
- Tanpa adanya Framework, maka setiap orang akan membuat kode dengan gaya masing-masing

# Sejarah Svelte

- Svelte adalah salah satu FrontEnd Framework yang populer saat ini
- Pertama kali dibuat oleh Rich Harris
- Svelte pertama kali diperkenalkan sekitar tahun 2016, dan sekarang menjadi salah satu Frontend Framework yang populer
- <https://svelte.dev/>
- <https://github.com/sveltejs/svelte>

# Component

- Saat belajar Svelte, kita harus terbiasa dengan istilah Component
- Component adalah kumpulan kode yang bisa digunakan secara independen
- Component bisa berisikan satu atau lebih HTML Element, kode JavaScript dan CSS
- Tidak ada aturan harus seberapa kecil atau besar sebuah Component
- Anggap saja seperti membuat Function, kita bisa membuat Function yang besar, atau kecil, karena tujuan Function adalah agar bisa digunakan secara berulang-ulang, begitu juga tujuan dari Component



# Ekosistem

- Svelte mengembangkan ekosistem nya sendiri, berbeda dengan React dimana banyak sekali pihak diluar React yang mengembangkan ekosistem nya
- Svelte sendiri mengembangkan ekosistem nya sendiri dengan membuat SvelteKit
- Keuntungannya adalah integrasi antar ekosistem nya jadi lebih baik, karena dikembangkan oleh tim yang sama
- Kekurangannya ekosistemnya jadi tidak sebesar seperti React

**Membuat Project**

# Cara Membuat Project

- Untuk membuat Project, terdapat 2 cara, pertama menggunakan SvelteKit yang kedua menggunakan Vite
- Pada materi ini, kita akan menggunakan Vite, hal ini karena kita belum belajar SvelteKit, dan SvelteKit akan kita bahas di materi terpisah
- Namun tidak perlu khawatir, karena SvelteKit sebenarnya menggunakan Vite juga ketika membuat project nya

# Membuat Project

```
npm create vite@latest belajar-svelte-dasar -- --template svelte
```

# Menjalankan App

```
npm run dev
```

**Hello Svelte**

# Hello World

- Seperti biasa, hal yang biasa kita buat ketika belajar adalah membuat aplikasi Hello World
- Sekarang kita akan buat halaman Hello World menggunakan Svelte

# Membuat Component

- Biasanya di Svelte, kita akan membuat Component dalam satu file
- Misal kita akan membuat Component bernama HelloSvelte
- Maka kita bisa membuat file HelloSvelte.svelte
- Di dalam Component, biasanya terbagi menjadi 3 bagian, Script (berisi kode JavaScript/TypeScript), Style (berisi kode CSS), dan HTML
- Hal ini mempermudah proses pembuatan Component, karena kita tidak perlu membuat banyak file, kita cukup buat dalam satu file Svelte



# Kode : src/lib/HelloSvelte.svelte

 HelloSvelte.svelte ×

```
1  <script>
2      console.log(`Hello Svelte!`)
3  </script>
4
5  <style>
6      h1 {
7          color: red;
8      }
9  </style>
10
11 <h1>Hello Svelte!</h1>
```

# Menampilkan Component

- Untuk menampilkan Component, kita bisa menggunakan `mount()` function dari Svelte
- <https://svelte.dev/docs/svelte/svelte#mount>
- Selanjutnya kita bisa tentukan, komponen mana yang akan ditampilkan, dan dimana akan ditampilkan

## Kode : src/hello.js

JS hello.js ×

```
1 import {mount} from 'svelte'
2 import './app.css'
3 import HelloSvelte from "../lib/HelloSvelte.svelte";
4
5 const app = mount(HelloSvelte, {
6   target: document.getElementById('app'),
7 })
8
9 export default app
```

# Kode : hello.html

<> hello.html ×

```
1  <!doctype html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <link rel="icon" type="image/svg+xml" href="/vite.svg" />
6      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7      <title>Vite + Svelte</title>
8    </head>
9    <body>
10     <div id="app"></div>
11     <script type="module" src="/src/hello.js"></script>
12   </body>
13 </html>
```

# Kode : Vite Config

JS vite.config.js ×

```
1  import {defineConfig} from 'vite'
2  import {svelte} from '@sveltejs/vite-plugin-svelte'
3
4  // https://vite.dev/config/
5  export default defineConfig({
6    plugins: [svelte()],
7    build: {
8      rollupOptions: {
9        input: {
10          index: 'index.html',
11          hello: 'hello.html'
12        }
13      }
14    }
15  })
```

**Template**

# Template

- Kode yang terdapat pada file Svelte sebenarnya sama dengan kode HTML yang biasanya kita buat
- Hanya saja, terdapat beberapa fitur lainnya yang digunakan untuk mempermudah saat kita membuat komponen
- Kita akan bahas sambil berjalan secara bertahap semua fitur-fitur tambahannya

# Text Expression

- Svelte memiliki fitur bernama Text Expression, dimana kita bisa membuat mengakses data expression (yang mengembalikan value) di kode Script pada HTML yang kita buat
- Kita bisa menggunakan tanda kurung kurawal buka, lalu diisi dengan expression nya, dan ditutup dengan kurung kurawal tutup
- Misal, kita akan coba ubah komponen HelloSvelte sebelumnya



# Kode : HelloSvelte

```

HelloSvelte.svelte ×
1  <script>
2      const name = 'Svelte';
3  </script>
4
5  <style>
6      h1 {
7          color: red;
8      }
9  </style>
10
11 <h1>Hello {name.toUpperCase()}!</h1>
12
```

# Dynamic Attribute

- Selain sebagai text, Text Expression juga bisa digunakan sebagai value di HTML Attribute
- Bahkan, jika nama variable yang digunakan di Text Expression sama dengan nama attribute di HTML, kita tidak perlu menyebutkan lagi nama attribute nya

# Kode : HelloSvelte



HelloSvelte.svelte ×

```
1 <script>
2   const name = 'Svelte';
3   const src = 'https://www.programmerzamannow.com/img/pzn.png'
4 </script>
5
6 <style>
7   h1 {
8     color: red;
9   }
10 </style>
11
12 <h1>Hello {name.toUpperCase()}!</h1>
13
14 <img src={src} alt="PZN Logo">
15
```

# Nested Component

- Dalam prakteknya, kita jarang sekali membuat satu halaman dalam satu komponen
- Biasanya kita akan buat komponen-komponen yang lebih kecil dan dalam satu halaman akan menggunakan komponen-komponen lainnya
- Untuk menggunakan komponen lain, kita bisa import komponen tersebut dan menggunakan layaknya tag HTML

# Kode : Logo

Logo.svelte ×

```
1 <script>
2   const src = 'https://www.programmerzamannow.com/img/pzn.png'
3 </script>
4
5 <img src={src} alt="PZN Logo">
6
```

# Kode : HelloSvelte

HelloSvelte.svelte ×

```
1 <script>
2   import Logo from "./Logo.svelte";
3
4   const name = 'Svelte';
5 </script>
6
7 <style>
8   h1 {
9     color: red;
10  }
11 </style>
12
13 <h1>Hello {name.toUpperCase()}!</h1>
14
15 <Logo/>
```

# HTML Tags

- Saat kita menggunakan Text Expression di Svelte, itu aman dari masalah XSS (Cross Site Scripting), oleh karena itu ketika kita akan menampilkan text yang berisi kode HTML, maka Svelte akan melakukan escape text terlebih dahulu
- Namun, jika pada kasus kita memang ingin menampilkan dalam bentuk format HTML apa adanya, maka kita perlu menggunakan tag `@html`

# Kode : HTML Tag

HelloSvelte.svelte ×

```
1 <script>
2   import Logo from "./Logo.svelte";
3
4   const name = 'Svelte';
5
6   const content = '<p>Hello world!</p>';
7 </script>
8
9 <style>
10   h1 {
11     color: red;
12   }
13 </style>
14
15 <h1>Hello {name.toUpperCase()}!</h1>
16
17 {@html content}
18
19 <Logo/>
```



# Reactive State

# Rune

- Rune adalah simbol di Svelte untuk mengontrol Svelte Compiler
- Rune memiliki prefix/awalan \$, dan terlihat seperti function, contohnya `$state("hello")`
- `$state("hello")` artinya kita menggunakan Rune state dan parameter "hello"
- Ada banyak Rune yang tersedia di Svelte, dan kita akan bahas secara bertahap, dimulai dengan `$state`
- [https://svelte.dev/docs/svelte/\\$state](https://svelte.dev/docs/svelte/$state)

# Reactive State

- Rune `$state` digunakan untuk membuat reactive state
- Apa bedanya state biasa dengan reactive state? Pada state biasa yang kita buat dalam variable, ketika terjadi perubahan pada variabel tersebut, maka UI tidak akan bereaksi terhadap perubahannya, hal ini menjadikan UI tidak akan berubah mengikuti perubahan state
- Sedangkan reactive state, memungkinkan UI bereaksi terhadap perubahan yang terjadi pada reactive state
- Saat kita menggunakan `$state`, kita bisa tentukan nilai awal state, dan untuk mengubahnya, kita bisa mengubahnya langsung hasil dari `$state` tersebut

# Tugas

- Sekarang kita akan coba membuat halaman yang menampilkan data Counter, lalu ketika tombol di klik nilai Counter akan naik
- Buat halaman counter.html
- Tambahkan halaman ke vite.config.js
- Buat src/counter.js
- Buat component src/lib/Counter.svelte

# Kode : Counter (tanpa Reactive State)

Counter.svelte ×

```
1 <script>
2   let count = 0;
3
4   function increment() {
5     count++;
6     document.getElementById("counter").innerHTML = "Counter : " + count + "";
7   }
8 </script>
9
10 <h1 id="counter">Counter : 0</h1>
11
12 <button onclick={increment}>increment</button>
13
```

# Kode : Counter (Reactive State)



Counter.svelte ×

```
1 <script>
2   let count = $state(0);
3
4   function increment() {
5     count++;
6   }
7 </script>
8
9 <h1>Counter : {count}</h1>
10
11 <button onclick={increment}>increment</button>
```

# Deep State

- Jika kita menggunakan tipe data seperti Array atau Object di `$state`, maka data tersebut akan dibungkus dalam JavaScript Proxy
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Proxy](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy)
- Dengan begitu, perubahan yang terjadi pada data tersebut, Svelte bisa mendeteksinya, dan secara otomatis UI akan bereaksi terhadap perubahan data nya
- `$state` tidak mendukung tipe data Collection seperti Set atau Map, namun Svelte menyediakan library terpisah untuk tipe data ini

# Tugas

- Sekarang kita akan coba buat halaman yang memiliki input `firstName` dan `lastName`, selanjutnya ketika di klik tombol Say Hello, nanti akan menampilkan tulisan Hello `firstName` `lastName`
- Buatlah halaman `say-hello.html`
- Registrasikan ke `vite.config.js`
- Buat `src/say-hello.js`
- Buat komponen `src/lib/SayHello.svelte`



# Kode : SayHello

SayHello.svelte x

```
1 <script>
2   let person = $state({
3     firstName: "",
4     lastName: ""
5   })
6
7   function sayHello() {
8     person.firstName = document.getElementById("firstName").value;
9     person.lastName = document.getElementById("lastName").value;
10  }
11 </script>
12
13 <div>
14   <input type="text" id="firstName" placeholder="First Name"> <br/>
15   <input type="text" id="lastName" placeholder="Last Name"> <br/>
16
17   <button onclick={sayHello}>Say Hello</button>
18
19   ⚡ <h1>Hello {person.firstName} {person.lastName}</h1>
20 </div>
```

# State Raw

- Jika kita tidak mau Object atau Array menjadi Reactive, misal kita hanya ingin variable nya saja yang reactive, kita bisa menggunakan `$state.raw`
- Dengan begitu, perubahan pada isi Object atau Array, tidak akan memicu perubahan pada UI

# Kode : SayHello

SayHello.svelte x

```
1 <script>
2   let person = $state.raw({
3     firstName: "",
4     lastName: ""
5   })
6
7   function sayHello() {
8     person = {
9       firstName: document.getElementById("firstName").value,
10      lastName: document.getElementById("lastName").value
11    }
12  }
13 </script>
14
15 <div>
16   <input type="text" id="firstName" placeholder="First Name"> <br/>
17   <input type="text" id="lastName" placeholder="Last Name"> <br/>
18
19   <button onclick={sayHello}>Say Hello</button>
20
21   ⚡ <h1>Hello {person.firstName} {person.lastName}</h1>
```

**Derived**

# Derived State

- Kadang-kadang, kita membuat state dari state lainnya
- Hal ini bisa kita lakukan menggunakan Rune `$derived(expression)`
- Dengan menggunakan ini, kita bisa membuat state yang bereaksi terhadap perubahan state lain
- [https://svelte.dev/docs/svelte/\\$derived](https://svelte.dev/docs/svelte/$derived)

# Tugas

- Kita akan ubah data state counter sebelumnya menjadi Array, dimana data array tersebut kita akan tambahkan angka baru setiap kita mengklik tombol increment

# Kode : Counter



Counter.svelte ×

```
1 <script>
2   let count = $state([]);
3   let total = $derived(count.reduce((a, b) => a + b, 0));
4
5   function increment() {
6     count.push(1);
7   }
8 </script>
9
10 <h1>{count.join(" + ")} = {total}</h1>
11
12 <button onclick={increment}>increment</button>
13
```

# Derived By

- Jika pada kasus kita tidak bisa membuat expression untuk `$derive`, kita bisa membuat dalam bentuk function
- Namun kita perlu menggunakan `$derived.by(function)`
- Misal kita akan coba ubah yang sebelumnya kita buat



# Kode : Counter

Counter.svelte x

```
1 <script>
2   let count = $state([]);
3   // let total = $derived(count.reduce((a, b) => a + b, 0));
4   let total = $derived.by(() => {
5     let total = 0;
6     for (let i = 0; i < count.length; i++) {
7       total += count[i];
8     }
9     return total;
10  })
11
12  function increment() {
13    count.push(1);
14  }
15 </script>
16
17 <h1>{count.join(" + ")} = {total}</h1>
18
19 <button onclick={increment}>increment</button>
```

**Inspect**

# Inspect State

- Kadang saat membuat aplikasi, kita sering melakukan log perubahan yang terjadi di State
- Biasanya, kita akan lakukan menggunakan `console.log(state)`
- Namun, karena state itu sebenarnya adalah Proxy Object, maka akan terjadi warning ketika kita lakukan itu
- Disarankan untuk menggunakan Rune `$inspect(state)` jika kita ingin pendeteksian perubahan yang terjadi pada state
- Misal, ketika terjadi perubahan, kita ingin melakukan log data nya, kita bisa menggunakan `$inspect(state).with((type, data) => ...)`
- [https://svelte.dev/docs/svelte/\\$inspect](https://svelte.dev/docs/svelte/$inspect)

# Kode : Counter

```
11
12  ✓ function increment() {
13      count.push(1);
14      // console.log(count); // warning
15  }
16
17  $inspect(count).with((type, values) => console.log(type, values));
18  </script>
19
20  <h1>{count.join(" + ")} = {total}</h1>
21
22  <button onclick={increment}>increment</button>
23  |
```

# Peringatan

- `$inspect` hanya jalan di mode development, ketika kode Svelte sudah kita build menggunakan Vite, maka `$inspect` tidak akan berjalan
- Hal ini menjadi aman kita menambahkan `$inspect`, karena hanya jalan di development, tidak akan jalan di production

**Universal State**

# Universal State

- Jika kita pernah belajar React atau Vue, State biasanya hanya bisa digunakan di Component, sebagai local state
- Svelte berbeda, kita bisa menggunakan state diluar component, dan bisa digunakan oleh lebih dari satu component
- Hal ini menjadikan state di Svelte bisa digunakan sebagai global state
- Keuntungannya adalah, kita tidak perlu lagi menggunakan Global State Management seperti di React atau di Vue
- Namun perlu diingat, Rune hanya bisa digunakan di file Svelte, jadi kita bisa buat dalam file svelte.js

# Tugas

- Kita akan mencoba membuat komponen GlobalCounter, dimana data counter nya akan kita simpan di luar komponen
- Buat halaman global-counter.html
- Tambahkan halaman ke vite.config.js
- Buat src/global-counter.js
- Buat src/state/counter.svelte.js
- Buat component src/lib/GlobalCounter.svelte
- Buat component src/GlobalCounterApp.svelte



## Kode : counter.svelte.js

JS counter.svelte.js ×


```
1 export const counter = $state({  
2   count: 0  
3 })
```

# Kode : GlobalCounter

GlobalCounter.svelte ×

```
1 <script>
2   import {counter} from "../state/counter.svelte.js";
3
4   function increment() {
5     counter.count++;
6   }
7 </script>
8
9 <button onclick={increment}>Increment : {counter.count}</button>
```

# Kode : GlobalCounterApp

 GlobalCounterApp.svelte ×

```
1 <script>
2   import GlobalCounter from "../lib/GlobalCounter.svelte";
3 </script>
4
5 <GlobalCounter/>
6 <GlobalCounter/>
7 <GlobalCounter/>
8 <GlobalCounter/>
```

**Effect**

# Effect

- Saat state berubah, UI yang menggunakan state tersebut akan secara otomatis bereaksi terhadap perubahan state tersebut, hal ini kita sebut dengan effect
- Pada beberapa kasus, mungkin kita ingin membuat effect sendiri, dimana ketika terjadi perubahan state, kita ingin menjalankan kode tertentu (effect)
- Hal ini bisa kita lakukan menggunakan Rune `$effect`
- [https://svelte.dev/docs/svelte/\\$effect](https://svelte.dev/docs/svelte/$effect)

# Tugas

- Misal, kita akan membuat counter yang berjalan otomatis menggunakan `setInterval()`, namun kita akan memberi tombol untuk menaikkan data interval, yang harapannya nanti waktu internal nya akan berubah mengikuti data state interval
- Buat halaman `auto-counter.html`
- Tambahkan halaman ke `vite.config.js`
- Buat `src/auto-counter.js`
- Buat component `src/lib/AutoCounter.svelte`

# Kode : AutoCounter

AutoCounter.svelte ×

```
1 <script>
2   let counter = $state(0)
3   let interval = $state(1000)
4
5   function increment() {
6     interval += 1000;
7   }
8
9   $effect(() => {
10     setInterval(() => {
11       counter++
12     }, interval)
13   })
14 </script>
15
16 <h1>Counter : {counter}</h1>
17
18 <button onclick={increment}>Increment Interval : {interval} ms</button>
```

# Kenapa Bermasalah?

- Saat kita mengklik tombol Increment, mana data interval akan berubah, dan secara otomatis `$effect` akan dijalankan lagi
- Dikarenakan `setInterval()` sebelumnya belum berhenti, secara otomatis akan ada 2 `setInterval()` yang berjalan, ini bahaya karena bisa menyebabkan memory leak
- Kita bisa mengembalikan closure function pada `$effect` untuk memberi tahu jika Svelte menjalankan `$effect` lagi, maka closure function tersebut akan di eksekusi
- Kita bisa isi closure function tersebut untuk mematikan `setInterval()` sebelumnya



# Kode : AutoCounter

```
8
9  ✓ $effect(() => {
10  ✓   const id = setInterval(() => {
11      counter++
12    }, interval)
13
14    return () => clearInterval(id)
15  })
16  </script>
17
18  <h1>Counter : {counter}</h1>
19
20  <button onclick={increment}>Increment Interval : {interval} ms</button>
```

# Kapan Menggunakan Effect

- Effect merupakan cara yang biasanya dilakukan terakhir jika memang tidak ada cara lain, misal seperti memanipulasi DOM secara manual
- Jadi jangan terlalu banyak menggunakan effect, terutama misal ketika ingin memanipulasi data secara synchronized, kita bisa memanfaatkan contohnya `$derived`, dibanding menggunakan `$effect`

# Props


# Props

- Saat kita memanggil component, kadang kita ingin mengirimkan data ke component yang kita panggil, mirip seperti HTML attribute
- Di Svelte, kita bisa menambahkan hal ini dengan menggunakan Rune `$props`
- Semua attribute yang kita kirimkan ketika memanggil component, akan dibungkus dalam object, dan kita bisa mengambilkan menggunakan `$props()`
- [https://svelte.dev/docs/svelte/\\$props](https://svelte.dev/docs/svelte/$props)
- Karena `$props()` mengembalikan Object, kita bisa mengambil datanya menggunakan JavaScript Destructuring

# Tugas

- Kita akan coba membuat komponen yang menampilkan informasi Row User, dimana ada id, name dan address
- Buat halaman user.html
- Tambahkan halaman ke vite.config.js
- Buat src/user.js
- Buat component src/lib/UserRow.svelte
- Buat component src/UserApp.svelte

## Kode : UserRow

 UserRow.svelte ×

```
1  <script>
2    const {id, name, address} = $props()
3  </script>
4
5  <tr>
6    <td>{id}</td>
7    <td>{name}</td>
8    <td>{address}</td>
9  </tr>
```

# Kode : UserApp

UserApp.svelte x


```
1  <script>
2    import UserRow from "../lib/UserRow.svelte";
3  </script>
4
5  <table>
6    <thead>
7      <tr>
8        <th>Id</th>
9        <th>Name</th>
10       <th>Address</th>
11     </tr>
12   </thead>
13   <tbody>
14     <UserRow id="1" name="John" address="123 Main St"/>
15     ⚡ </tbody>
16 </table>
```

# Spread Props

- Kadang pada kasus misal kita sudah punya object dan ingin mengirimnya menggunakan props, agak menyulitkan karena harus mengetikkan semua attribute nya
- Untungnya kita bisa menggunakan Spread operator di Props



# Kode : UserApp

```
3
4   const user = {
5     id: 1,
6     name: "John",
7     address: "123 Main St"
8   }
9 </script>
10
11 <table>
12   <thead>
13     <tr>
14       <th>Id</th>
15       <th>Name</th>
16       <th>Address</th>
17     </tr>
18   </thead>
19   <tbody>
20     <UserRow {...user}/>
21      </tbody>
```

**If Block**

# If

- Saat kita membuat halaman web, pastinya nanti kita akan sering membuat halaman yang dinamis tergantung data yang harus ditampilkan.
- Artinya kita perlu menambahkan logic ketika menampilkan template di component
- Svelte menyediakan markup `#if` pada text expression di template yang bisa kita gunakan untuk percabangan
- <https://svelte.dev/docs/svelte/if>

# Tugas

- Kita akan coba membuat komponen yang menampilkan Todo, dimana akan ada props `done:boolean`, jika `done` bernilai `true`, maka Todo akan kita coret
- Buat halaman `todo.html`
- Tambahkan halaman ke `vite.config.js`
- Buat `src/todo.js`
- Buat component `src/lib/Todo.svelte`
- Buat component `src/ToDoApp.svelte`

# Kode : Todo

Todo.svelte ×

```
1 <script>
2   const {id, name, done} = $props()
3 </script>
4
5 {#if done}
6   <strike>{id} - {name}</strike>
7 {:else}
8   {id} - {name}
9 {/if}
```

# Kode : TodoApp



TodoApp.svelte ×

```
1 <script>
2
3 import Todo from "../lib/Todo.svelte";
4 </script>
5
6 <ul>
7   <li><Todo id="1" name="Belajar HTML" done={true}/></li>
8   <li><Todo id="2" name="Belajar CSS" done={false}/></li>
9   <li><Todo id="3" name="Belajar JavaScript" done={false}/></li>
10  <li><Todo id="4" name="Belajar NodeJS" done={false}/></li>
11 </ul>
```

**Each Block**

# Each Block

- Saat membuat halaman web, kita kadang sering membuat elemen dari kumpulan data. Dibandingkan menggunakan copy paste semua elemen sejumlah datanya, kita bisa menggunakan Each Block di Template
- <https://svelte.dev/docs/svelte/each>



# Tugas

- Kita akan ubah halaman TodoApp dimana data todo kita akan simpan dalam Array of Object

# Kode : TodoApp

TodoApp.svelte ×

```
1 <script>
2   import Todo from "../lib/TODO.svelte";
3
4   const data = [
5     {
6       id: 1,
7       name: "Belajar HTML",
8       done: true
9     },
10    {
11      id: 2,
12      name: "Belajar CSS",
13      done: false
14    },
15    {
16      id: 3,
17      name: "Belajar JavaScript",
18      done: false
19    },
```

```
20    {
21      id: 4,
22      name: "Belajar NodeJS",
23      done: false
24    }
25  ]
26 </script>
27
28 <ul>
29   {#each data as todo}
30     <li>
31       <Todo {...todo}/>
32     </li>
33   {/each}
34 </ul>
```

# Masalah di Each Block

- Saat memanipulasi data Array yang digunakan di Each Block, misal menghapus data, maka Svelte akan coba menghapus elemen terakhir. Termasuk ketika menambah data, Svelte akan menambah elemen di bagian akhir
- Jika sebelumnya kita menggunakan React atau Vue, dimana komponen akan di render ulang semuanya, maka di Svelte cara kerjanya tidak seperti itu
- Ini untuk mempercepat proses karena tidak dibutuhkan render ulang seluruh komponen pada Each Block
- Namun, bisa jadi ada masalah

# Kode : Todo

Todo.svelte ×

```
1  <script>
2    const {id, name, done} = $props()
3
4    const emojis = {
5      "1": "🐼 ",
6      "2": "🔥 ",
7      "3": "😬 ",
8      "4": "🤖 ",
9    }
10
11    const emoji = emojis[id];
12    console.log(emoji);
13  </script>
14
15  {emoji}
16
17  {#if done}
18    <strike>{id} - {name}</strike>
```

# Kode : TodoApp

TodoApp.svelte x

```
1 <script>
2   import Todo from "../lib/Todo.svelte";
3
4   function remove() {
5     data.shift()
6   }
7
8   const data = $state([
9     {
10      id: 1,
11      name: "Belajar HTML",
12      done: true
13    }
14  ])
```

```
29   });
30 </script>
31
32 <button onclick={remove}>Remove</button>
33
34 <ul>
35   {#each data as todo}
36     <li>
37       <Todo {...todo}/>
38     </li>
39   {/each}
40 </ul>
41
```

# Keyed Each Block

- Jika kita lihat hasilnya di Browser Log, ternyata ketika kita hapus data di Array, maka komponen tidak akan dipanggil ulang
- Hal ini otomatis akan mempercepat proses, namun masalahnya adalah, karena Svelte akan menghapus atau menambah di bagian akhir, maka ketika kita hapus yang awal, maka element tidak berubah, hanya berubah pada bagian yang Reactive State nya saja
- Untuk memberitahu element mana yang harusnya dihapus, diubah atau ditambahkan, maka kita bisa menambah informasi key pada Each Block
- <https://svelte.dev/docs/svelte/each#Keyed-each-blocks>

## Kode : TodoApp

```
31
32 <button onclick={remove}>Remove</button>
33
34 <ul>
35   {#each data as todo (todo.id)}
36     <li>
37       <Todo {...todo}/>
38     </li>
39   {/each}
40 </ul>
```

# Await Block



# Await Block

- Saat kita membuat halaman web, kita sering sekali menggunakan Promise
- Svelte memiliki block untuk mengakses function Promise, sehingga kita bisa menentukan apa yang mau ditampilkan sebelum Promise selesai dan setelah Promise selesai
- Kita bisa menggunakan Await Block
- <https://svelte.dev/docs/svelte/await>

# Tugas

- Kita akan coba membuat komponen yang menampilkan Article, dimana datanya akan kita ambil dari Ajax
- Buat halaman article.html
- Tambahkan halaman ke vite.config.js
- Buat src/article.js
- Buat component src/lib/Article.svelte
- Buat file public/article.json

## Kode : article.json

{ article.json ×

```
1 {  
2   "id": "1",  
3   "title": "Judul Artikel",  
4   "content": "This is a sample article content to demonstrate how an  
5 }
```



# Kode : Article

Article.svelte ×

```
1  <script>
2    async function getArticle() {
3      let response = await fetch("/article.json");
4      return await response.json();
5    }
6  </script>
7
8  {#await getArticle()}
9    <p>Loading ...</p>
10  {:then article}
11    <h1>{article.title}</h1>
12    <p>{article.content}</p>
13  {:catch error}
14    <p>Error: {error.message}</p>
15  {/await}
```

**Event**

# Event

- Menambahkan Event ke Element di Template sangat mudah, kita bisa menggunakan `on<event>={function}`, dimana function bisa function yang kita buat di script, atau closure function

# Tugas

- Sekarang kita akan coba tambahkan form untuk input Todo, dan tambahkan Button untuk menambah Todo

## Kode : Todo

 Todo.svelte ×

```
1 <script>
2   const {id, name} = $props()
3 </script>
4
5 {id} - {name}
```



# Kode : TodoApp

TodoApp.svelte x

TE,

```
1 <script>
2   import Todo from "../lib/Todo.svelte";
3
4   const data = $state([]);
5
6   let id = 0;
7
8   function add(e) {
9     e.preventDefault();
10    const input = document.getElementById("todo");
11    data.push({id: id++, name: input.value});
12    input.value = "";
13  }
14 </script>
```

15

16

17

18

19

20

21

22

23

24

25

26

27

28

```
<form>
```

```
  <input type="text" id="todo">
```

```
  <button onclick={add}>Add</button>
```

```
</form>
```

```
<ul>
```

```
  {#each data as todo (todo.id)}
```

```
    <li>
```

```
      <Todo {...todo}/>
```

```
    </li>
```

```
  {/each}
```

```
</ul>
```

# Tugas

- Tambahkan tombol Delete untuk menghapus Todo

# Kode : TodoApp

TodoApp.svelte x

```
1 <script>
2   import Todo from "../lib/Todo.svelte";
3
4   let data = $state([]);
5
6   let id = 0;
7
8   function add(e) {
9     e.preventDefault();
10    const input = document.getElementById("todo");
11    data.push({id: id++, name: input.value});
12    input.value = "";
13  }
14
15  function remove(id) {
16    data = data.filter((item) => item.id !== id);
17  }
18 </script>
```

```
20 <form>
21   <input type="text" id="todo">
22   <button onclick={add}>Add</button>
23 </form>
24
25 <ul>
26   {#each data as todo (todo.id)}
27     <li>
28       <Todo {...todo}/>
29       <button onclick={() => remove(todo.id)}>
30         Remove
31       </button>
32     </li>
33   {/each}
34 </ul>
```

# Event Props

- Saat kita membuat komponen, tidak hanya data yang bisa kita gunakan sebagai props
- Kita juga bisa mengirim event sebagai props
- Misal, kita akan coba buat komponen untuk EditTodo untuk melakukan edit data Todo

# Kode : EditTodo

```
EditTodo.svelte x
1  <script>
2    const {id, name, onedit} = $props()
3
4    function save() {
5      const input = document.getElementById(id);
6      onedit(id, input.value)
7    }
8  </script>
9
10 {id} <input type="text" id={id} value={name}>
11 <button onclick={save}>Save</button>
```

# Kode : TodoApp

```
19
20  function edit(id) {
21    for (let i = 0; i < data.length; i++) {
22      if (data[i].id === id) {
23        data[i] = {edit: true, ...data[i]};
24      }
25    }
26  }
27
28  function onEdit(id, name) {
29    for (let i = 0; i < data.length; i++) {
30      if (data[i].id === id) {
31        data[i] = {id, name, edit: false};
32      }
33    }
34  }
35  </script>
```

```
42  <ul>
43    {#each data as todo (todo.id)}
44      <li>
45        {#if todo.edit}
46          <EditTodo id={todo.id} name={todo.name}
47            onedit={onEdit}/>
48        {:else}
49          <Todo {...todo}/>
50          <button onclick={() => edit(todo.id)}>
51            Edit</button>
52          <button onclick={() => remove(todo.id)}>
53            Remove</button>
54        {/if}
55      </li>
56    {/each}
57  </ul>
```

# Binding

# Binding

- Biasanya data mengalir dari parent ke child element. Namun kadang ada kondisi kita ingin menggunakan data dari child element ke parent element
- Contoh yang biasa kita lakukan adalah ketika menggunakan input element
- Kita biasanya mengambil data dari input element secara manual di parent element, seperti yang kita lakukan sebelumnya
- Svelte memiliki fitur untuk melakukan binding, dimana attribute pada child element bisa kita sinkronisasikan dengan data pada parent element
- <https://svelte.dev/docs/svelte/bind>



# Tugas

- Kita akan ubah aplikasi Todo yang sebelumnya kita buat, menjadi menggunakan binding

# Kode : EditTodo

```
EditTodo.svelte ×  
1  <script>  
2    let {id, name, onedit} = $props()  
3  
4    function save() {  
5      | onedit(id, name)  
6    }  
7  </script>  
8  
9  {id} <input type="text" id={id} bind:value={name}>  
10 <button onclick={save}>Save</button>  
11
```

# Kode : TodoApp

TodoApp.svelte ×

```
1  <script>
2    import Todo from "../lib/Todo.svelte";
3    import EditTodo from "../lib/EditTodo.svelte";
4
5    let data = $state([]);
6    let name = $state("");
7
8    let id = 0;
9
10   function add(e) {
11     e.preventDefault();
12     data.push({id: id++, name: name});
13     name = "";
14   }
15
```

```
36
37   <form>
38     <input type="text" id="todo" bind:value={name}>
39     <button onclick={add}>Add</button>
40   </form>
41
```

**Style**

# Style

- Saat membuat membuat Element, kadang kita perlu menambahkan CSS Style pada Element tersebut
- Kita bisa menggunakan attribute style pada Element
- <https://svelte.dev/docs/svelte/style>

## Kode : Counter

```
15     },  
16  
17     $inspect(count).with((type, values) => console.log(type, values));  
18 </script>  
19  
20 <h1 style:color="red">{count.join(" + ")} = {total}</h1>  
21  
22 <button onclick={increment}>increment</button>  
23
```

# Scoped Style

- Selain menggunakan attribute style, kita juga bisa menggunakan tag style pada template untuk membuat Style pada Komponen yang kita buat
- Namun perlu diketahui, style yang kita buat dalam Komponen hanya bisa digunakan pada komponen tersebut
- Artinya kita bisa membuat nama CSS Class yang sama pada Komponen yang berbeda tanpa harus takut terjadi bentrok
- <https://svelte.dev/docs/svelte/scoped-styles>

# Kode : EditTodo

EditTodo.svelte ×

```
1  <script>
2    let {id, name, onedit} = $props()
3
4    function save() {
5      onedit(id, name)
6    }
7  </script>
8
9  {id} <input type="text" id={id} bind:value={name}>
10 <button class="btn" onclick={save}>Save</button>
11
12 <style>
13   .btn {
14     background-color: #007BFF; /* Primary blue color */
15     color: white; /* White text */
16     border: none; /* Remove default border */
17     border-radius: 5px; /* Rounded corners */
18     padding: 10px 20px; /* Padding inside the button */
```



# Global Style

- Jika ada kebutuhan kita ingin membuat Style yang global di Svelte, kita bisa tambahkan :global pada style nya
- <https://svelte.dev/docs/svelte/global-styles>
- Misal, kita akan pindahkan class .btn ke TodoApp, sehingga semua element bisa menggunakan .btn

# Kode : TodoApp

```
49     <Todo {...todo}/>
50     <button onclick={() => edit(todo.id)} class="btn">
51       Edit
52     </button>
53     <button onclick={() => remove(todo.id)} class="btn">
54       Remove
55     </button>
56   {/if}
57 </li>
58 {/each}
59 </ul>
60
61 <style>
62   :global {
63     .btn {
64       background-color: #007BFF; /* Primary blue color */
65       color: white; /* White text */
66       border: none; /* Remove default border */
67       border-radius: 5px; /* Rounded corners */
68       padding: 10px 20px; /* Padding inside the button */
69       font-size: 16px; /* Slightly larger text */
70       cursor: pointer; /* Pointer cursor on hover */
71       transition: background-color 0.3s ease, transform 0.2s ease; /
```

# Class Attribute

- Kita tahu bahwa dalam HTML, kita bisa gunakan class attribute di element untuk menggunakan class CSS yang ingin kita pilih
- Di Svelte, kita juga bisa lakukan hal itu, namun ada penambahan feature di Svelte ketika menggunakan class attribute
- <https://svelte.dev/docs/svelte/class>

# Kode : Counter

```
17     $inspect(count).with((type, values) => console.log(type, values));
18   </script>
19
20   <h1 class={total % 2 === 0 ? 'red' : 'blue'}>{count.join(" + ")} = {total}</h1>
21
22   <button onclick={increment}>increment</button>
23
24   <style>
25     .red {
26       color: red;
27     }
28     .blue {
29       color: blue;
30     }
31   </style>
```

**Action**

# Action

- Action adalah function yang dipanggil ketika elemen dipasang (mounted).
- Kita bisa menggunakan use:nama\_action pada element yang akan ditambahkan Action
- Action biasanya menggunakan \$effect, sehingga ketika element dilepas (unmounted), closure effect akan dipanggil untuk reset
- <https://svelte.dev/docs/svelte/use>
- <https://svelte.dev/docs/svelte/svelte-action>

# Tugas

- Kita akan coba membuat komponen yang menampilkan text yang akan kita tambahkan informasi time pada text nya
- Buat halaman time.html
- Tambahkan halaman ke vite.config.js
- Buat src/time.js
- Buat component src/lib/Time.svelte

# Kode : Time

Time.svelte ×

```
1 <script>
2
3   const time = (node) => {
4     const text = node.textContent
5
6     $effect(() => {
7       const id = setInterval(() => {
8         node.textContent = text + ' ' + new Date().toLocaleTimeString()
9       }, 1000)
10
11       return () => clearInterval(id)
12     })
13   }
14
15 </script>
16
17 <h1 use:time>Title</h1>
18
19 <p use:time>Hello World</p>
```



# Parameter

- Kadang mungkin ada keadaan di tiap element kita ingin menggunakan Action yang sama, namun menggunakan data yang berbeda
- Kita bisa menambahkan parameter pada Action tersebut, sehingga tiap element yang menggunakan Action yang sama, bisa memiliki informasi parameter yang berbeda-beda

# Kode : Time

```
Time.svelte ×  
1 <script>  
2  
3   const time = (node, interval) => {  
4     const text = node.textContent  
5  
6     $effect(() => {  
7       const id = setInterval(() => {  
8         node.textContent = text + ' ' + new Date().toISOString()  
9       }, interval)  
10  
11      return () => clearInterval(id)  
12    })  
13  }  
14  
15 </script>  
16  
17 <h1 use:time={500}>Title</h1>  
18  
19 <p use:time={1000}>Hello World</p>  
20
```

**Transition**

# Transition

- Transition merupakan feature dimana kita menambahkan transisi element ketika masuk dan keluar dari DOM
- Kita bisa menggunakan transition: pada element yang akan ditambahkan
- <https://svelte.dev/docs/svelte/transition>
- Svelte menyediakan beberapa proses transition yang bisa kita gunakan
- <https://svelte.dev/docs/svelte/svelte-transition>

# Tugas

- Kita akan coba update halaman TodoApp dengan menambahkan transition pada tiap Todo

# Kode : TodoApp

TodoApp.svelte x

```
1 <script>
2   import Todo from "../lib/Todo.svelte";
3   import EditTodo from "../lib/EditTodo.svelte";
4   import {fade} from "svelte/transition"
5
```

```
42
43 <ul>
44   {#each data as todo (todo.id)}
45     <li transition:fade>
46       {#if todo.edit}
47         <EditTodo id={todo.id} name={todo.name}
48           onedit={onEdit}/>
49       {:else}
50         <Todo {...todo}/>
51         <button onclick={() => edit(todo.id)} class="edit">
52           Edit
53         </button>
54         <button onclick={() => remove(todo.id)} class="remove">
55           Remove
```

# Transition Parameter

- Beberapa jenis transition memiliki parameter yang bisa kita ubah
- Kita bisa menggunakan `transition:jenis={parameter}` untuk mengubah parameter dari transition yang kita gunakan
- <https://svelte.dev/docs/svelte/svelte-transition>

# Kode : TodoApp

```
43 <ul>
44   {#each data as todo (todo.id)}
45   <li transition:fly={{y: 200, duration: 2000}}>
46     {#if todo.edit}
47       <EditTodo id={todo.id} name={todo.name}
48         onedit={onEdit}/>
49     {:else}
50       <Todo {...todo}/>
51       <button onclick={() => edit(todo.id)} class="btn">
52         Edit
53       </button>
54       <button onclick={() => remove(todo.id)} class="btn">
55         Remove
56       </button>
```



# In dan Out

- Kadang kita ingin menggunakan jenis transition yang berbeda ketika masuk dan keluar
- Kita bisa menggunakan in: dan out: jika ingin menggunakan jenis transition yang berbeda
- <https://svelte.dev/docs/svelte/in-and-out>

# Kode : TodoApp

```
42
43 <ul>
44   {#each data as todo (todo.id)}
45     <li in:fly={{y: -200, duration: 2000}} out:fly={{y: 200, duration: 2000}}>
46       {#if todo.edit}
47         <EditTodo id={todo.id} name={todo.name}
48           onedit={onEdit}/>
49       {:else}
50         <Todo {...todo}/>
51         <button onclick={() => edit(todo.id)} class="btn">
52           Edit
53         </button>
54         <button onclick={() => remove(todo.id)} class="btn">
55           Remove
56         </button>
57       {/if}
58     </li>
```

# Transition Event

- Transition memiliki event yang bisa kita tambahkan pada element yang menggunakan transition tersebut
- <https://svelte.dev/docs/svelte/transition#Transition-events>

# Kode : TodoApp

```
42
43  ✓ <ul>
44  ✓    {#each data as todo (todo.id)}
45  ✓      <li in:fly={{y: -200, duration: 2000}} out:fly={{y: 200, duration: 2000}}
46      onintrostart={() => console.log("intro start")}
47      onintroend={() => console.log("intro end")}
48      onoutrostart={() => console.log("outro start")}
49      onoutroend={() => console.log("outro end")}
50      >
51      ✓    {#if todo.edit}
52      ✓      <EditTodo id={todo.id} name={todo.name}
53      |      onedit={onEdit}/>
54      {:#else}
55      <Todo {...todo}/>
```

# Reactive Class

# Reactive Class

- Sebelumnya kita selalu menggunakan \$state dan \$derived pada variable
- \$state dan \$derived juga bisa digunakan pada property di class, sehingga bisa menciptakan Reactive Class

# Tugas

- Kita akan coba membuat komponen yang menampilkan box (kotak) yang akan kita ubah ukuran tinggi dan lebar nya dari input spinner
- Buat halaman box.html
- Tambahkan halaman ke vite.config.js
- Buat src/box.js
- Buat component src/lib/Box.svelte

# Kode : Box

Box.svelte ×

```
1 <script>
2   class Box {
3     width = $state(100)
4     height = $state(100)
5     area = $derived(this.width * this.height)
6   }
7
8   const box = new Box();
9 </script>
10
11 <div>
12   Width : <input type="range" min="0" max="500" bind:value={box.width}>
13   <br>
14   Height : <input type="range" min="0" max="500" bind:value={box.height}>
15 </div>
16
17 <div style:width="{box.width}px" style:height="{box.height}px" style:background="red">
18   {box.area}
19 </div>
```



**Reactive Built-in**

# Reactive Built-in

- Kadang mungkin ada kondisi kita ingin membuat reactive state yang tipe data nya bukan tipe data sederhana, object atau array. Misal Map, Set, Date, dan lain-lain
- Svelte menyediakan wrapper class untuk tipe data tersebut
- <https://svelte.dev/docs/svelte/svelte-reactivity>

# Tugas

- Kita akan coba membuat halaman untuk input data user dan nilai, dimana kita akan gunakan tipe data reactive SvelteMap
- Buat halaman student.html
- Tambahkan halaman ke vite.config.js
- Buat src/student.js
- Buat component src/lib/Student.svelte

# Kode : Student

Student.svelte x

```
1 <script>
2   import {SvelteMap} from "svelte/reactivity";
3
4   const students = new SvelteMap();
5
6   let student = $state("")
7   let value = $state(0);
8
9   function addStudent() {
10     students.set(student, value);
11
12     student = "";
13     value = 0;
14   }
15 </script>
16
17 <div>
18   Student : <input type="text" bind:value={student}/>
19   <br>
20   Value : <input type="number" bind:value={value}/>
21   <br>
22   <button onclick={addStudent}>Add Student</button>
23 </div>
```

```
24
25 <div>
26   <table>
27     <thead>
28       <tr>
29         <th>Student</th>
30         <th>Value</th>
31       </tr>
32     </thead>
33     <tbody>
34       {#each students as entry (entry[0])}
35         <tr>
36           <td>{entry[0]}</td>
37           <td>{entry[1]}</td>
38         </tr>
39       {/each}
40     </tbody>
41   </table>
42 </div>
```

**Store**

# Store

- Sebelum Svelte 5, untuk membuat State diluar komponen, kita perlu menggunakan Store. Namun sebenarnya di Svelte 5, kita bisa menggunakan Rune untuk membuat Global/Universal State
- Di bab ini kita akan bahas cara lama membuat Universal State menggunakan Store, karena mungkin saya kita menggunakan library yang masih menggunakan Store
- <https://svelte.dev/docs/svelte/svelte-store>

# Jenis-Jenis Store

- Store terdapat dua jenis, yaitu Readable dan Writeable.
- Readable merupakan store yang bisa dibaca datanya, dan Writeable merupakan store yang bisa dibaca dan diubah datanya
- <https://svelte.dev/docs/svelte/svelte-store>
- Untuk membuat store Readable, kita bisa menggunakan function readable(), sedangkan untuk membuat store Writeable, kita bisa menggunakan function writable()
- Saat mengakses store, kita perlu menggunakan tanda \$ di awal variable store

# Tugas

- Kita akan coba membuat komponen seperti GlobalCounter, namun menggunakan Store
- Buat halaman store-counter.html
- Tambahkan halaman ke vite.config.js
- Buat src/store-counter.js
- Buat src/state/counter.js
- Buat component src/lib/StoreCounter.svelte
- Buat component src/StoreCounterApp.svelte




## Kode : counter

JS counter.js ×


```
1 import {writable} from "svelte/store";  
2  
3 export const counter = writable(0);  
4 counter.subscribe(count => console.log(count));  
5
```

# Kode : StoreCounter

 StoreCounter.svelte ×

```
1 <script>
2   import {counter} from "../state/counter.js";
3
4   function increment() {
5     $counter.count++;
6   }
7 </script>
8
9 <button onclick={increment}>Increment : {$counter.count}</button>
```

# Kode : StoreCounterApp

 StoreCounterApp.svelte ×

```
1  <script>
2    import StoreCounter from "../lib/StoreCounter.svelte";
3  </script>
4
5  <StoreCounter/>
6  <StoreCounter/>
7  <StoreCounter/>
```

# Snippet dan Render

# Snippet

- Snippet merupakan cara untuk membuat komponen yang bisa digunakan, namun masih tetap di dalam komponen yang sama, tanpa harus membuat file lain
- Ini cocok untuk pada kasus yang sederhana ketika kita ingin memecah komponen, namun tidak ingin memecah komponen dalam file yang terpisah
- Untuk membuat snippet, kita bisa gunakan `{#snippet nama_snippet(param)}` dalam file Svelte
- <https://svelte.dev/docs/svelte/snippet>

# Render

- Render merupakan perintah yang digunakan untuk menampilkan snippet
- Untuk menggunakan render, kita bisa menggunakan perintah `{@render nama_snippet(param)}`
- <https://svelte.dev/docs/svelte/@render>

# Tugas

- Kita akan coba mengubah daftar list Todo pada TodoApp menjadi snippet

# Kode : TodoApp

```
37
38   {#snippet todoRow(todo)}
39     <li in:fly={{y: -200, duration: 2000}} out:fly={{y: 200, duration: 2000}}
40       | onintrostart={() => console.log("intro start")}
41       | onintroend={() => console.log("intro end")}
42       | onoutrostart={() => console.log("outro start")}
43       | onoutroend={() => console.log("outro end")}
44     >
45     {#if todo.edit}
46       <EditTodo id={todo.id} name={todo.name}
47       | onedit={onEdit}/>
48     {:else}
49       <Todo {...todo}/>
50       <button onclick={() => edit(todo.id)} class="btn">
51         Edit
52       </button>
53       <button onclick={() => remove(todo.id)} class="btn">
54         Remove
55       </button>
56     {:/if}
57   </li>
58 {/snippet}
59
```

```
64
65   <ul>
66     {#each data as todo (todo.id)}
67     | {@render todoRow(todo)}
68     {/each}
69   </ul>
70
```



# Mengirim Snippet ke Component

- Di Svelte versi 4, saat kita ingin mengirim komponen ke komponen, biasanya menggunakan Slot, namun di Svelte 5, kita bisa menggunakan Snippet, tidak perlu menggunakan Slot lagi
- Snippet sebenarnya seperti function biasa, jadi kita bisa kirim sebagai props ketika memanggil Component
- Salah satu kelebihan dari Snippet, kita bisa mengirim sebagai props dalam inner element Component nya, sehingga tidak perlu kita sebutkan lagi sebagai attribute

# Tugas

- Kita akan coba membuat halaman Blog, dimana dalam blog ada bagian header, content dan footer. Bagian content misal kita ingin buat dalam bentuk dinamis, sehingga kita bisa mengirim snippet kedalam bagian content nya
- Buat halaman `blog.html`
- Tambahkan halaman ke `vite.config.js`
- Buat `src/blog.js`
- Buat component `src/lib/Layout.svelte`
- Buat component `src/lib/Blog.svelte`

# Kode : Layout

```
Layout.svelte ×  
1  <script>  
2    const {title, content, footer} = $props();  
3  </script>  
4  
5  <h1>{title}</h1>  
6  
7  <div>  
8    {@render content()}  
9  </div>  
10  
11 <footer>{footer}</footer>
```

# Kode : Blog

Blog.svelte x

```
1 <script>
2   import Layout from "./Layout.svelte";
3 </script>
4
5 <Layout title="Contoh Blog" footer="Copyright 2024">
6   {#snippet content()}
7     <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aut bl
8       dorum eius exercitationem harum ipsa itaque nam nihil numquam
9       voluptatibus!</p>
10    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Archit
11      expedita incidunt laborum officiis, quibusdam quod saepe unde,
12      impedit praesentium. Accusantium, eaque!</p>
13   {/snippet}
14 </Layout>
```

**Context**

# Masalah Dengan Global State

- Sebelumnya kita tahu bahwa Reactive State di Svelte bisa digunakan diluar Component, artinya bisa digunakan sebagai Global State
- Hal ini sangat cocok ketika kita hanya membuat web SPA (Single Page Application) yang dijalankan di Client Side, karena Global State tersebut hanya berjalan di tiap Browser pengguna
- Namun jika aplikasi Svelte kita berjalan di server dan memiliki Server Component, maka penggunaan Global State sangat berbahaya, karena perubahan di server bisa mengubah semua Server Component sehingga Client bisa mendapat data yang sama

# Context


- Alternatif lain jika kita ingin membuat State yang Global adalah dengan memanfaatkan Context
- Context adalah tempat kita menyimpan data, saat kita simpan data di Context, maka hanya Component tersebut dan juga Child Component nya yang bisa mengakses data di State tersebut
- Context itu tidak Reactive, sehingga jika kita ingin menjadikan Reactive, kita harus memasukkan data Reactive kedalam Context
- <https://svelte.dev/docs/svelte/context>

# Tugas

- Kita akan coba membuat komponen seperti GlobalCounter, namun menggunakan Context
- Buat halaman context-counter.html
- Tambahkan halaman ke vite.config.js
- Buat src/context-counter.js
- Buat component src/lib/ContextCounter.svelte
- Buat component src/ContextCounterApp.svelte



# Kode : ContextCounter

 ContextCounter.svelte ×

```
1 <script>
2   import {getContext} from "svelte";
3
4   const counter = getContext("counter")
5
6   function increment() {
7     counter.count++;
8   }
9 </script>
10
11 <button onclick={increment}>Increment : {counter.count}</button>
```

# Kode : ContextCounterApp



ContextCounterApp.svelte ×

```
1  <script>
2    import ContextCounter from "../lib/ContextCounter.svelte";
3    import {setContext} from "svelte";
4
5    const counter = $state({count: 0});
6
7    setContext("counter", counter)
8  </script>
9
10 <ContextCounter/>
11 <ContextCounter/>
12 <ContextCounter/>
```

# Lifecycle Hook

# Lifecycle Hook

- Saat komponen ditampilkan, kita bisa menambahkan hook ketika komponen dibuat dan dihapus
- Untuk menambahkan hook ketika komponen dibuat, kita bisa menggunakan `onMount(callback)`
- Untuk menambahkan hook ketika komponen dihapus, kita bisa menggunakan `onDestroy(callback)`
- Khusus callback dalam `onMount()` dapat mengembalikan closure function yang akan dieksekusi ketika dihapus
- <https://svelte.dev/docs/svelte/lifecycle-hooks>

# Tugas

- Kita akan coba ubah komponen Blog, agar mengambil data Blog dari AJAX menggunakan `onMount()`

## Kode : blog.json

{ } blog.json ×

```
1 {  
2   "title": "Contoh Blog",  
3   "content": "<p>Lorem ipsum dolor sit amet, consectetur  
4   "footer" : "Copyright 2025"  
5 }
```



# Kode : Blog

Blog.svelte x

```
1 <script>
2   import Layout from "../Layout.svelte";
3   import {onMount} from "svelte";
4
5   let blog = $state({
6     title: "",
7     footer: "",
8     content: ""
9   })
10
11   onMount(async () => {
12     const response = await fetch("/blog.json")
13     blog = await response.json()
14   })
15 </script>
```

```
16
17 <Layout title={blog.title} footer={blog.footer}>
18   {#snippet content()}
19     {@html blog.content}
20   {/snippet}
21 </Layout>
22
```

**Special Element**



# Special Element

- Saat kita membuat web, kadang-kadang kita ingin mengubah element seperti window, document, body atau head
- Sayangnya, hal ini tidak bisa dilakukan secara langsung karena Svelte sendiri biasanya akan di render ke dalam body halaman yang sudah ada
- Jika kita ingin menambahkan event misalnya ke element-element tersebut, maka kita harus gunakan kode JavaScript
- Untungnya, Svelte memiliki komponen spesial, yang digunakan untuk merepresentasikan element-element tersebut
- Dengan begitu, kita bisa menambah event layaknya seperti komponen biasanya

# Daftar Special Element

- Ada beberapa elemen spesial yang dimiliki oleh Svelte, yaitu
- `svelte:window`, sebagai representasi dari Window :  
<https://svelte.dev/docs/svelte/svelte-window>
- `svelte:document`, sebagai representasi dari Document :  
<https://svelte.dev/docs/svelte/svelte-document>
- `svelte:body`, sebagai representasi dari Body :  
<https://svelte.dev/docs/svelte/svelte-body>
- `svelte:head`, sebagai representasi dari Head :  
<https://svelte.dev/docs/svelte/svelte-head>

# Tugas

- Kita akan coba membuat halaman untuk mendeteksi lokasi mouse di Body, lokasi mouse akan kita tampilkan secara realtime
- Buat halaman mouse.html
- Tambahkan halaman ke vite.config.js
- Buat src/mouse.js
- Buat component src/lib/Mouse.svelte

# Kode : Mouse

Mouse.svelte ×

```
1 <script>
2   const mouse = $state({
3     x: 0,
4     y: 0
5   })
6
7   function handleMouseMove(event) {
8     mouse.x = event.clientX
9     mouse.y = event.clientY
10  }
11 </script>
12
13 <svelte:body onmousemove={handleMouseMove}/>
14
15 <h1>X : {mouse.x}</h1>
16 <h1>Y : {mouse.y}</h1>
```

# Referensi

# Referensi

- <https://svelte.dev/docs/svelte/overview>

**Materi Selanjutnya**

# Materi Selanjutnya

- Svelte Kit