# Componentes

Son archivos con extensión .svelte

Dentro de este archivo todo se va a tratar como si de html se tratara, separando solo el contenido de los <scripts> y <style> que van a estar encerrados en sus respectivas etiquetas

## Insertar valores

### atributos

Dentro de los atributos se pueden insertar valores usando:

<script> let valor=”a”; let variableCoincidente=”valor”; //si coincide el nombre con el atributo </script>

<etiqueta atributo={valor} o atributo=”algo {valor}” o { variableCoincidente } ejemplo {src}/>

TS

<script lang=”ts”>

let valor: string =”a”;

let variableCoincidente: string =”valor”; //si coincide el nombre con el atributo

</script>

### Directamente en el html

O directamente en el html

<div> {valor} </div>

### Interpretación dinámica

El código entre llaves se interpreta como js

<div> {valor===1?’uno’:’dos’} </div>

# Imágenes:

Necesitan por obligación el atributo alt

# Importar

Se importan de forma igual que react pero dentro de los <scripts>

## Importar componentes

<scripts>

Import CustomComponente from ‘./CustomComponente.svelte’

</scripts>

# @html

Se le puede forzar a que un valor string se interprete como html

<div> {@html variable} </div>

# Eventos

Primero se declara alguna funcion en el script

Luego en un elmento html es on:nombreEvento={metodoParaEvento}

<button on:click={ metodoParaEvento }> </>

## Tipos

click, pointermove, input ,submit, change, keydown

### input

se llama ca vez que se escribe algo

<script>

Let name=”el nombre”

</script>

<input type=’text’ value={name} on:input={(event)=>{ name=event.target.value}} />

<h1> el nombre: {name} </h1>

## inline

<div on: pointermove={(e)=>{

variableExterna={x:e.clientX,y:e.clientY};

}} />

## Modificadores

Se puede modificar el comportamiento de un evento, por ejemplo que solo se llame una vez

<button on:click|once={ metodoParaEvento }> </>

once,preventDefault,

### varios

<button on:click|once| preventDefault ={ metodoParaEvento }> </>

## Dispatch

Para disparar una especie de eventos personalizados que pueden ser escuchado desde un hijo a un padre

Componente hijo que Lanza el evento

Import { createEventDispcher } from ‘svelte’;

const despachador= createEventDispcher();

const payload={ …};

despachador(‘nombre-llave’, payload,);

parecido a un hook de react

Componente padre que espera el evento

<script>

Function handlerDespachador(event){

let payload=event.detail

Alert(payload.algunAtributo);

}

</script>

<Hijo on: nombre-llave={ handlerDespachador } />

### profundidad

Si es entre varios niveles de profundidad, cada componente intermedio tiene que tener on:nombre-llave, sin asignarle un handler

intermedio

<Hijo on: nombre-llave/>

Padre

<Intermedio on: nombre-llave={ handlerDespachador } />

# Declaraciones reactivas

Automáticamente declara la variable como un let

Se ‘llama’ cada ves que uno de sus componentes cambien

Osea por ejemplo el html si se recarga solo si cambia alguna variable que lo afecta (solo si cambia por asignación, osea si un atributo de un objeto cambia no recarga, si un arreglo cambia no recarga), pero no es el caso del los script, solo se ejecutan una vez al cargar el componente, pero con esto, este código se va a llamar cada ves que una variable (no las declaradas dentro de el) de la que depende cambie

## variable

$: variable=’valor’

## codigo

Tambien se puede declarar de forma reactiva código

### Solo codigo

$:{

alert(‘paso algo ’+count)

alert(‘otra cosa’)

}

### condicionales

$: if (count>=0){

alert(‘paso algo’)

count=0

}

Fíjate que utilice varias líneas para esto, el limite lo dan las llaves en este caso

# Propiedades

Es como se pasan datos de un padrea a un hijo, (en react eran los parámetros del componente),

Para crear una ‘propertie’ básicamente es declarar una variable en el script como export

Hijo.svelte

<script>

export variablePropiedad=’valor por defecto’;

export variablePropiedad2; //sin por defecto, toma true por defecto

</script>

<div> { variablePropiedad2} </div>

Padre.svelte

<script>

Import Hijo from ‘./Hijo.svelte’

</script>

<Hijo variablePropiedad2=’valor’ variablePropiedad2 /> <!-- el 2 toma true -->

## Descomposición

Padre.svelte

<script>

Import Hijo from ‘./Hijo.svelte’

Const data={

‘variablePropiedad’:’valor1’

‘variablePropiedad2’:’valor2’

}

</script>

<Hijo {…data} />

## $$.props

Tiene todos los atributos que se la han pasado (como todos los parámetros de la funcion de componente en react)

Hijo.svelte

Let variable=$$.props.atributo

# If html

Para en la parte de la html usar el if

{#if count>2}

Html

{/if}

## Else

{#if count>2}

Html si true

{:else}

Html si false

{/if}

## Elif

{#if count>2}

Html si true

{:else if count>1}

Html si esto se cumple

{:else}

Html si nada se cumple

{/if}

# Each For

<script>

Const variableArreglo=[“a”,”b”,”c”];

</script>

{# each variableArreglo as variableIndividual}

<div>{ variableIndividual }</div>

{/each}

## Indice

{# each variableArreglo as variableIndividual , i }

<div>indice:{i} elemento: { variableIndividual }</div>

{/each}

## Key, actualización correcta

Si se va a actualizar el arreglo, para que se renderice bien es necesario que exista un key,

De momento se le pasa en el lugar donde ubicábamos el índice (valorIndividual.AtributoKey)

<script>

Const variableArreglo=[

{id:1,name:”a”},

{id:2,name:”b”},

{id:3,name:”c”},

];

Function handlerClick(){

variableArreglo.slice(1);

}

</script>

{# each variableArreglo as variableIndividual (variableIndividual.id)}

<div on:click={ handlerClick }>{ variableIndividual }</div>

{/each}

# #await Promesas

Si tienes una variable que es una promesa, generalmente es una variable que se inicializa al llamar a un método asyn

Se puede declarar en el html directamente un código para cuando se esta cargando, para cuando se obtenga con éxito y con error

<script>

let variablePromesa=callPromesa();

</script>

{#await variablePromesa }

Html cargando …

{:then valorObtenido}

<div> { valorObtenido } </div>

{:cath error}

<div> {error.message} </div>

{/await}

# Bind

Enlaza una variable con el valor de input, de modo que se actualicen cada vez que uno de los dos cambia

## input

<script>

Let name=”el nombre”

</script>

<input type=’text’ bind:value={name} />

<h1> el nombre: {name} </h1>

## Checkbox

<input type=’checkbox’ bind:checked={chequeado} />

## Radio button

<scrips>

Let valor\_seleccionado=”valor actual”;

</scrips>

<input type=’radio’ value=”valor de este radio a” bind:group={ valor\_seleccionado } />

<input type=’radio’ value=”valor de este radio b” bind:group={ valor\_seleccionado } />

## Checkbox

<scrips>

Let valores\_seleccionados=[];

</scrips>

<input type=’checkbox’ value=”valor de este a” bind:group={ valores\_seleccionados } />

<input type=’checkbox’ value=”valor de este b” bind:group={ valores\_seleccionados } />

## Select multiple

<scrips>

Let valores\_seleccionados=[];

</scrips>

<select multiple bind:value={ valores\_seleccionados } >

<option value=”a”>A</>

<option value=”b”>B</>

</select>

## This

Se pueden almacenar referencias a los elementos HTML

<scrips>

Let div;

</scrips>

<div bind:this={div} />

## Store

Almacen.js

Import {writeable} from ‘svelte/store’

Export const name = writeable(0);

OtroComponente

<script>

Import name from ‘almacen.js’‘’

</script>

<input bind:value={$name} />

# Ciclo de vida

Son hooks que se van a llamar para interactuar con varios momentos del ciclo de vida

onMount, onDestroy, beforeUpdate, afterUpdate, tick

## onMount

en el estan los momento de montaje y desmontaje

import {onMount} from ‘svelte’

onMount( ()=>{

//codigo que se ejecuta

Return ()=>{

//codigo cuando se desmonta

}

})

### Asyn

Tambien puede ser una función de este tipo, pero en este caso no hay que especificar el return

onMount( asyn ()=>{

//codigo que se ejecuta

})

## onDestroy

import { onDestroy} from ‘svelte’

onDestroy ( ()=>{

//codigo que se ejecuta

})

## beforeUpdate,afterUpdate

antes y después de actualizarse, igual que los anteriores

## tick

import { tick } from ‘svelte’

// se queda esperando a que pase algo, supongo algo del ciclo?

await tick();

// luego se ejecuta

# Store

Componente Almacen.js normalmente llamado ‘Store’, fijarse que la extensión es js

Supongo que la idea es que sean como variables staticas disponibles a toda la app

## Writeable

Estos tienen dos métodos para editar vW.update((oldValue)=>newValue) y vW.set(newValue) este ultimo redefine la variable y para suscribirse vW.suscribe((value)=>{\\despues de que cambia el valor})

### Editar

Almacen.js

Import {writeable} from ‘svelte/store’

Export const count= writeable(0);

### Suscribirse

Const suscripcion= vW.suscribe((value)=>{\\despues de que cambia el valor})

#### $

Una mejor manera es solo importando la variable del store (Almacen) y usándola poniéndole un $ delante

import { variableTipoWriteable } from ‘nuestro\_almacen.js’

<div>$ variableTipoWriteable<div>

### Desuscribirse

import { onDestroy} from ‘svelte’

onDestroy(suscripcion)

### custom

la idea es empaquetar una funcionalidad

almacen.js

Import {writeable} from ‘svelte/store’

Function createCount(){

Const {subscribe,set,update } =writeable(0);

Return {

Subscribe,

Incrementar: ()=>update((n)=>n+1),

Resetear: ()=>set(0)

};

}

Export const count= createCount (0);

OtroComponente

<script>

Import count from ‘almacen.js’‘’

</script>

<div>$count</div>

<buttom on:click={count.incrementar}> +</>

<buttom on:click={count.resetar}> resetar</>

## readable

solo permite lectura

Import {readable} from ‘svelte/store’

export const variable\_leible =readable(valor inicial o null,

(set)=>{ //start, set es una function con la que se puede cambiar el valor de esta variable, pq la idea es que no tenga valor hasta que se suscribe alguien, que es cuando se llama este metodo, además tambien es la idea que internamente usando el set y algun método de ejecución automática se cambie el valor de la variable

//lo que se ejecuta cuando obtiene su primer suscriptor

return ()=>{//stop

//lo que se ejecuta cuando el ultimo suscriptor se desuscribe

}

})

## Derived

Subvariables declaradas dentro del store que dependen de otras variables del store

almacen.js

Import {writeable, derived} from ‘svelte/store’

Export const name=writeable(‘word’)

Export const greeting = derived(name,($name)=>’hola’+$name);

OtroComponente

<script>

Import greeting from ‘almacen.js’‘’

</script>

<div>$greeting </div>