



SVELTE

Programación Reactiva



Nicholson Ochoa
Juan Ruiz
David Eslava
Juan Solano
Brian Barreto

INTRODUCCIØ



Que es Svelte?



- Framework reactivo para construir interfaces de usuario.
- Utiliza lenguajes conocido como JS, HTML y CSS
- Diseñado para ser precompilado.
- No utiliza tecnicas como DOM diffing.



Es un compilador que ayuda a obtener código en Javascript optimizado y muy liviano

Historia



Creado por Rich Harris teniendo en cuenta 3 características:

- Excelente rendimiento y fácil desarrollo.
- Curva de aprendizaje corta.
- Cambiar librerías o frameworks por JS puro.

Equipo encargado de mantenerlo: Central Svelte

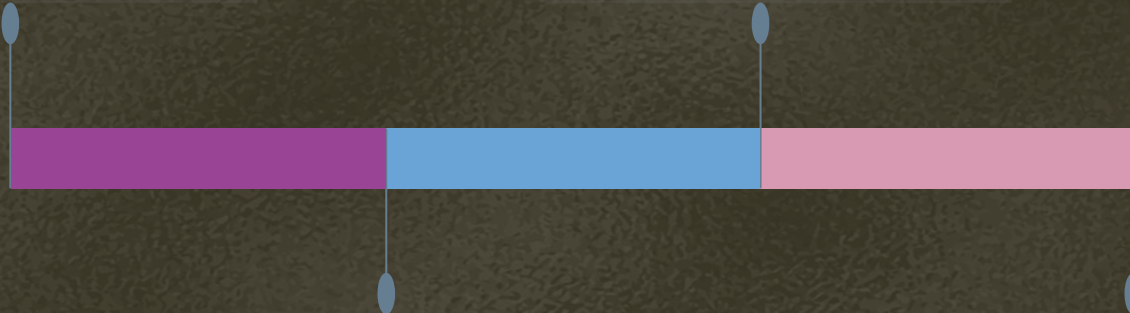
Historia

29 de Noviembre del
2016
Version 1

21 de Abril del 2019
Version 3

19 de Abril del 2018
Version 2

Presente



Características

1

No utiliza DOM

2

Propio manejo de estados (Stores)

3

Sintaxis sencilla

4

Reducción de tamaño de las apps

5

Buena documentacion

6

Es TOTALMENTE REACTIVO

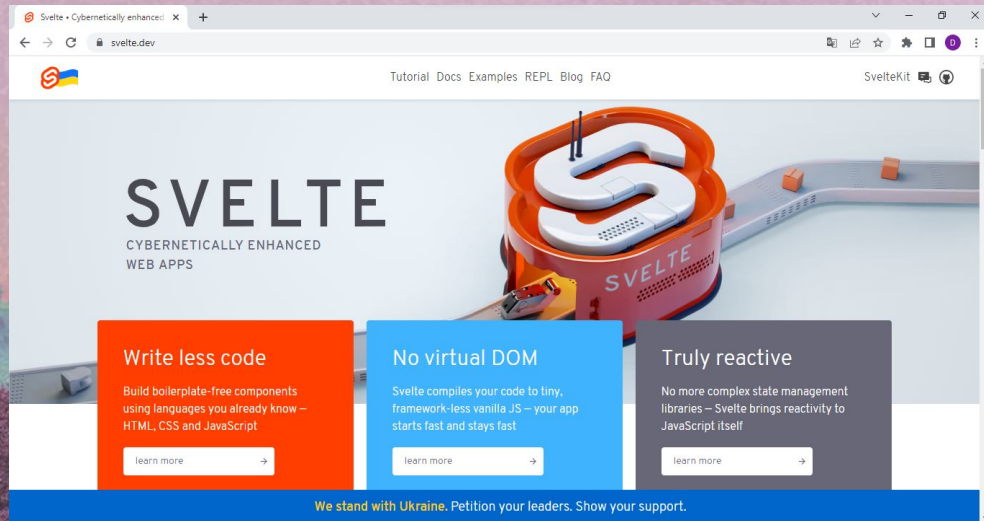
	React	Preact	Vue	Svelte	Web Components	Glimmer
Developer Productivity	High	Normal	High	Normal	Low	High
Build Pipeline	Good	Fair	Good	Fair	Poor	Good
Industry Adoption	High	High	High	Low	Very Low	Low
Community Support	Good	Good	Good	Growing	Poor	Poor
Low Memory Footprint	Good	Better	Good	Best	Good	Good
Performance	Good	Good	Good	Best	Good	Good
Conceptual Integrity	✓	✓	✗	✗	✓	✓
Learning Curve	Low	Low	Medium	Medium	High	High



Tour por Svelte

Un recorrido por el framework y su sintaxis

~~Una imagen~~
ejemplo dice
más que mil
palabras

A screenshot of a web browser displaying the Svelte website. The browser's address bar shows 'svelte.dev'. The website features a large 'SVELTE' logo in the background, which is a stylized 'S' on a red and orange structure. Below the logo, the text 'CYBERNETICALLY ENHANCED WEB APPS' is visible. Three main feature boxes are present: 'Write less code' (orange), 'No virtual DOM' (blue), and 'Truly reactive' (grey). Each box contains a brief description and a 'learn more' button with a right-pointing arrow. At the bottom of the page, a blue banner reads 'We stand with Ukraine. Petition your leaders. Show your support.'

Svelte • Cybernetically enhanced: x +

svelte.dev

Tutorial Docs Examples REPL Blog FAQ

SvelteKit

SVELTE

CYBERNETICALLY ENHANCED
WEB APPS

Write less code
Build boilerplate-free components using languages you already know – HTML, CSS and JavaScript
[learn more](#)

No virtual DOM
Svelte compiles your code to tiny, framework-less vanilla JS – your app starts fast and stays fast
[learn more](#)

Truly reactive
No more complex state management libraries – Svelte brings reactivity to JavaScript itself
[learn more](#)

We stand with Ukraine. Petition your leaders. Show your support.

<https://svelte.dev/>

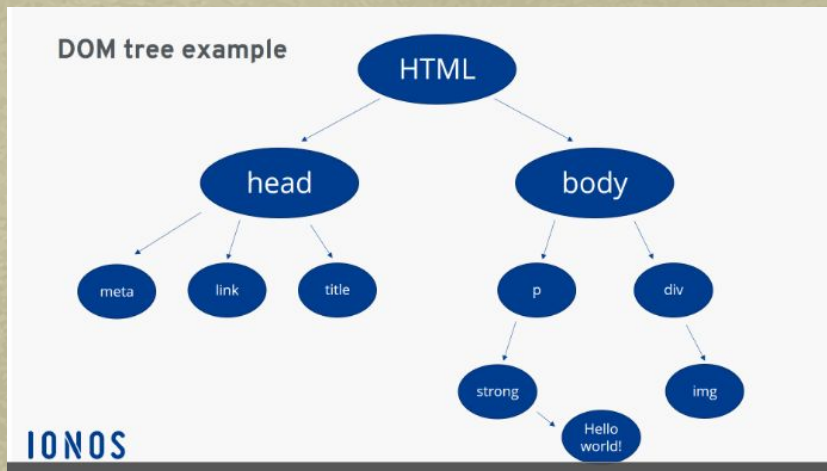
Repaso:

Componente:

Es un bloque de código autónomo reutilizable que encapsula HTML, CSS y JavaScript que van juntos, escrito en un archivo .svelte.

```
App.svelte +  
1 <h1>Hello world!</h1>
```

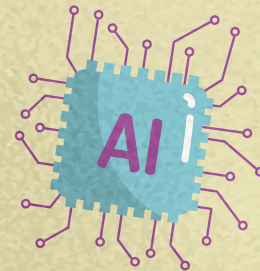
DOM





Reactividad

Asignaciones



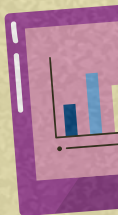
Hay un poderoso sistema de 'reactividad' para mantener el DOM en sincronización con el estado de la aplicación, por ejemplo en respuesta a un evento.

```
App.svelte +
1 <script>
2   let count = 0;
3
4   function handleClick() {
5     count += 1;
6   }
7 </script>
8
9 <button on:click={handleClick}>
10   Clicked {count} {count === 1 ? 'time' : 'times'}
11 </button>
```

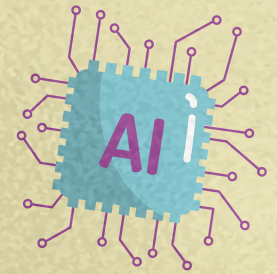
Result JS output CSS output

Clicked 1 time

Controlador de eventos



Declaraciones



SVELTE actualiza automáticamente el DOM cuando cambia el estado de su componente. A menudo, algunas partes del estado de un componente deben calcularse a partir de otras partes (como un nombre completo derivado de un primer nombre y un último nombre), y recomputados cada vez que cambian.

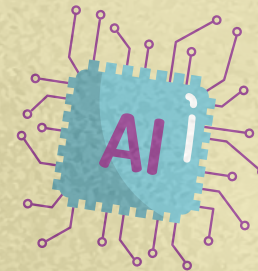
```
let count = 0;  
$: doubled = count * 2;
```

Referenciados



Statements

sentencias



```
$: console.log('the count is ' + count);
```

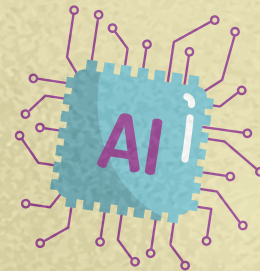
```
$: {  
  console.log('the count is ' + count);  
  alert('I SAID THE COUNT IS ' + count);  
}
```

```
$: if (count >= 10) {  
  alert('count is dangerously high!');  
  count = 9;  
}
```



No estamos limitados a las declaraciones

Arreglos y objetos



```
function addNumber() {  
  numbers.push(numbers.length + 1);  
}
```

La reactividad de Svelte se desencadena por las asignaciones. Los métodos que mutan matrices u objetos no activarán las actualizaciones por sí mismos.

```
function addNumber() {  
  numbers.push(numbers.length + 1);  
  numbers = numbers;  
}
```

```
function addNumber() {  
  numbers = [...numbers, numbers.length + 1];  
}
```

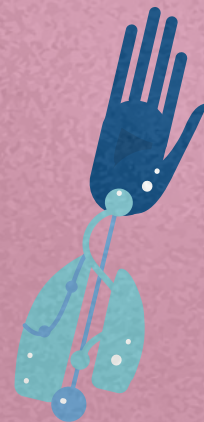
La misma regla aplica para métodos de arreglos como `pop`, `shift`, and `splice`, y a métodos de objetos como `Map.set`, `Set.add`, etc.



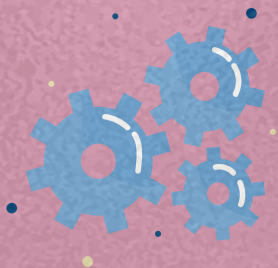


Props

Declaración de Props



```
<script>  
  export let answer;  
</script>
```



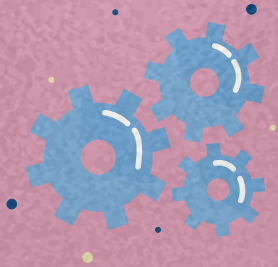
Abreviación de Properties, nos permiten pasar datos de un componente a otro sobre el estado interno del mismo.

Valores por defecto



```
<script>
  export let answer = 'a mystery';
</script>

<p>The answer is {answer}</p>
```



Podemos especificar un valor por defecto para la prop.

Extender props



```
const pkg = {  
  name: 'svelte',  
  version: 3,  
  speed: 'blazing',  
  website: 'https://svelte.dev'  
};  
</script>  
  
<Info {...pkg}/>
```

Esto

En lugar de esto:

```
<Info name={pkg.name} version={pkg.version} speed={pkg.speed} website={pkg.website}/>
```

Si tiene un objeto de propiedades, puede 'extenderlas' en un componente en lugar de especificar cada una:



Slots

Un Slot es un elemento que permite ser usado como "placeholder" y de ésta forma ser reemplazado por el marcado HTML que deseemos.

```
<script>
  import Box from './Box.svelte';
</script>

<Box>
  <h2>Hello!</h2>
  <p>This is a box. It can contain anything.</p>
</Box>
```

Hello!

This is a box. It can contain anything.


Slots nombrados

```
<script>
  import ContactCard from './ContactCard.svelte';
</script>
```

```
<ContactCard>
  <span slot="name">
    P. Sherman
  </span>

  <span slot="address">
    42 Wallaby Way<br>
    Sydney
  </span>
</ContactCard>
```


P. Sherman

 42 Wallaby Way
Sydney

 Unknown email



Stores



Un Store provee una utilidad similar a la de un Prop. Como su nombre lo indica, un Store es un componente que almacena información y que puede ser accedida por cualquier componente de la aplicación.

- Stores escribibles
- Stores de solo lectura
- Stores derivados
- Stores personalizados

Store Escribable

```
<script>
  import { count } from './stores.js';
  import Incrementer from './Incrementer.svelte';
  import Decrementer from './Decrementer.svelte';
  import Resetter from './Resetter.svelte';

  let countValue;

  const unsubscribe = count.subscribe(value => {
    countValue = value;
  });
</script>

<h1>The count is {countValue}</h1>

<Incrementer/>
<Decrementer/>
<Resetter/>
```

The count is 0

+ - reset

Store de solo lectura

```
<script>
  import { time } from './stores.js';

  const formatter = new Intl.DateTimeFormat('en', {
    hour12: true,
    hour: 'numeric',
    minute: '2-digit',
    second: '2-digit'
  });
</script>

<h1>The time is {formatter.format($time)}</h1>
```

**The time is 11:37:47
PM**

Store derivado

```
<script>
  import { time, elapsed } from './stores.js';

  const formatter = new Intl.DateTimeFormat('en', {
    hour12: true,
    hour: 'numeric',
    minute: '2-digit',
    second: '2-digit'
  });
</script>

<h1>The time is {formatter.format($time)}</h1>

<p>
  This page has been open for
  {$elapsed} {$elapsed === 1 ? 'second' : 'seconds'}
</p>
```

**The time is 11:39:01
PM**

This page has been open for 25 seconds

Store personalizado

```
<script>
  import { count } from './stores.js';
</script>

<h1>The count is {{count}}</h1>

<button on:click={count.increment}>+</button>
<button on:click={count.decrement}>-</button>
<button on:click={count.reset}>reset</button>
```

The count is 0





Logic



En Svelt es posible hacer expresiones lógicas dentro del mismo archivo

- If
- If-Else
- Each
- Await

If

```
<script>
  let user = { loggedIn: false };

  function toggle() {
    user.loggedIn = !user.loggedIn;
  }
</script>

{#if user.loggedIn}
  <button on:click={toggle}>
    Log out
  </button>
{/if}

{#if !user.loggedIn}
  <button on:click={toggle}>
    Log in
  </button>
{/if}
```

Log in

Else

```
<script>
  let user = { loggedIn: false };

  function toggle() {
    user.loggedIn = !user.loggedIn;
  }
</script>

{#if user.loggedIn}
  <button on:click={toggle}>
    Log out
  </button>
{:else}
  <button on:click={toggle}>
    Log in
  </button>
{/if}
```

Log in

Each

```
<script>
  let cats = [
    { id: 'J---aiyznGQ', name: 'Keyboard Cat' },
    { id: 'z_AbfPXTKms', name: 'Maru' },
    { id: 'OUtn3pvWmpg', name: 'Henri The Existential Cat' }
  ];
</script>

<h1>The Famous Cats of YouTube</h1>

<ul>
  {#each cats as { id, name }, i}
    <li>
      <a target="_blank" href="https://www.youtube.com/watch?v={id}">
        {i + 1}: {name}
      </a>
    </li>
  {/each}
</ul>
```

The Famous Cats of YouTube

- 1: Keyboard Cat
- 2: Maru
- 3: Henri The Existential Cat

Await

```
<script>
  let promise = getRandomNumber();

  async function getRandomNumber() {
    const res = await fetch(`/tutorial/random-number`);
    const text = await res.text();

    if (res.ok) {
      return text;
    } else {
      throw new Error(text);
    }
  }

  function handleClick() {
    promise = getRandomNumber();
  }
</script>

<button on:click={handleClick}>
  generate random number
</button>

{#await promise}
  <p>...waiting</p>
{:then number}
  <p>The number is {number}</p>
{:catch error}
  <p style="color: red">{error.message}</p>
{/await}
```

generate random number

The number is 78



Events

Svelte nos facilita hacer uso y definir listeners hacia eventos en el DOM usando `on:<event>`

- Mouse Event
- Event Modifiers
- Component Events
- Event forwarding

Mouse Event

```
<script>
  let m = { x: 0, y: 0 };

  function handleMousemove(event) {
    m.x = event.clientX;
    m.y = event.clientY;
  }
</script>

<div on:mousemove={handleMousemove}>
  The mouse position is {m.x} x {m.y}
</div>

<style>
  div { width: 100%; height: 100%; }
</style>
```

The mouse position is 8 x 8

Event modifier

```
<script>
  function handleClick() {
    alert('no more alerts')
  }
</script>

<button on:click|once={handleClick}>
  Click me
</button>
```

Click me

Component Events

```
<script>
  import Inner from './Inner.svelte';

  function handleMessage(event) {
    alert(event.detail.text);
  }
</script>

<Inner on:message={handleMessage}/>
```

Click to say hello

Component Events

```
<script>
  import Inner from './Inner.svelte';

  function handleMessage(event) {
    alert(event.detail.text);
  }
</script>

<Inner on:message={handleMessage}/>
```

Click to say hello

Event forwarding

```
<script>
  import Outer from './Outer.svelte';

  function handleMessage(event) {
    alert(event.detail.text);
  }
</script>

<Outer on:message={handleMessage}/>
```

Click to say hello

The image features a central window-like frame with a blue background. At the top left, there are three small white circles representing window control buttons. The frame is filled with a grid of thin black lines that converge towards the center, creating a perspective effect. The word "Bindings" is written in a large, bold, yellow font across the middle of the window.


Bindings

Un binding es una propiedad de las etiquetas de Svelte que permite asignarla a un valor definido

- Text input
- Numeric input
- Checkbox
- Multiple inputs with groups
- Files
- Select
- Multiple select
- bloques Each
- Dimensions
- Components

A stylized window frame with a blue background. The window has a dark blue header bar with three small white circles on the left. The main area of the window is filled with a light blue background and a grid of thin black lines that create a perspective effect, converging towards the center. The word "Lifecycle" is written in a large, bold, yellow font across the middle of the window.

Lifecycle

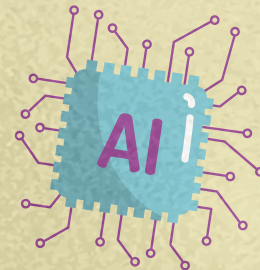


Cada componente tiene un ciclo de vida que comienza cuando es creado y se acaba cuando es destruido. Hay muchas funciones que permiten correr código en ciertos momentos durante el ciclo de vida. Las funciones del ciclo de vida se deben llamar mientras el componente se está inicializando para que la devolución de llamada esté vinculada a la instancia del componente.

Lifecycle Functions

- onMount
- onDestroy
- beforeUpdate
- afterUpdate
- Tick

Referencias



- <https://svelte.dev/tutorial/adding-data>
- <https://developer.mozilla.org/es/docs/Glossary/DOM>
- <https://github.com/sveltejs/svelte>
- <https://ed.team/blog/que-es-sveltejs-1>
- https://www.youtube.com/watch?v=Xsxm8_BI63s
- <https://midu.dev/introducción-a-svelte/>
- <https://svelte.dev>

