

TRABAJO EN GRUPO

Tutorial de Svelte

Ángel Cardoso Parreño

David Martínez Ruiz

Valiera Catalá Khudzhadze

¿Qué es Svelte?

Svelte es un marco de JavaScript que se utiliza para desarrollar aplicaciones web. A diferencia de otros marcos como React o Angular, que se basan en el patrón de "virtual DOM" (un árbol de elementos que representa el HTML final a renderizar), Svelte se encarga de la mayoría de la optimización de **rendimiento en el momento de la compilación**, en lugar de dejar que el navegador lo haga **en tiempo de ejecución**. Esto puede resultar en aplicaciones más rápidas y con menos código JavaScript necesario para ejecutarlas.

Además, Svelte es **conocido por** tener una sintaxis más simple y concisa que otros marcos, lo que lo hace más fácil de aprender y utilizar para desarrolladores principiantes o que ya tienen experiencia en otras tecnologías. Sin embargo, como Svelte es un marco relativamente nuevo y aún en desarrollo, tiene menos **recursos**, **tutoriales** y **bibliotecas** disponibles a comparación de otros marcos más establecidos.

Ventajas de Svelte

Tras investigar sobre Svelte hemos encontrado varias utilidades y ventajas de utilizar este marco para desarrollar aplicaciones web:

1. **Optimización de rendimiento**: Svelte se encarga de la mayoría de la optimización de rendimiento en el momento de la compilación, en lugar de dejar que el navegador lo haga en tiempo de ejecución. Esto puede resultar en aplicaciones más rápidas y con menos código JavaScript necesario para ejecutarlas.
2. **Sintaxis simple y concisa**: Svelte tiene una sintaxis más simple y concisa que otros marcos, lo que lo hace más fácil de aprender y utilizar para desarrolladores principiantes o que ya tienen experiencia en otras tecnologías.
3. **Amplia compatibilidad**: Svelte es compatible con la mayoría de los navegadores modernos, lo que lo hace ideal para desarrollar aplicaciones web que deben ser compatibles con una amplia gama de dispositivos.

Desventajas de Svelte

Sin embargo, también hay algunas desventajas que se deben considerar al elegir Svelte para un proyecto:

1. **Menos recursos y bibliotecas disponibles**: Como Svelte es un marco relativamente nuevo y aún en desarrollo, **puede tener menos recursos y bibliotecas** disponibles que otros marcos más establecidos.
2. **Mayor curva de aprendizaje**: Aunque Svelte tiene una sintaxis más simple y concisa que otros marcos, **puede tener una mayor curva de aprendizaje** si eres nuevo en el desarrollo de aplicaciones web o si no estás familiarizado con los conceptos de compilación y optimización de rendimiento.

En conclusión

En general, Svelte es una **buena opción** para desarrollar aplicaciones web rápidas y con una sintaxis simple y concisa.

En resumen, Svelte es un marco de JavaScript que ofrece muchas **ventajas** para desarrollar aplicaciones web, como una optimización de rendimiento mejorada y una sintaxis simple y concisa. Sin embargo, también hay algunas **desventajas** que debes considerar al elegir Svelte para un proyecto, como menos recursos y bibliotecas disponibles y una mayor curva de aprendizaje.

Para decidir si Svelte es la opción correcta para un proyecto, se deben considerar las **necesidades y objetivos** de desarrollo que se tengan, así como la **experiencia previa** en el desarrollo de aplicaciones web. Si lo que buscamos es optimizar el rendimiento de la aplicación y tenemos **tiempo para aprender** una nueva tecnología, Svelte podría ser una buena opción. Si, por otro lado, queremos un marco con una mayor cantidad de recursos y bibliotecas disponibles o tenemos un plazo de tiempo apretado para desarrollar la aplicación, otra opción sería más factible.

Arquitectura de Svelte

El formato que tienen las componentes básicas de las aplicaciones de Svelte contiene tres secciones, script, estilos y marcado.

```
<script>
    // logic goes here
</script>

<!-- markup (zero or more items) goes here -->

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

Sintaxis de la plantilla

Bucles

Svelte proporciona una sintaxis de plantilla llamada `{#each}` que nos permite iterar sobre una lista de valores. Si queremos recorrer una lista podemos utilizar la siguiente expresión:

```
{#each expression as name}...{/each}
```

Esto nos servirá para conseguir cada valor de una lista. Veamos un ejemplo.

```
<h1>Shopping list</h1>
<ul>
    {#each items as item}
        <li>{item.name} x {item.qty}</li>
    {/each}
</ul>
```

Este código realizará una lista de la compra en la cuál recorre todos los ítems de la lista y nos mostrará el nombre y la cantidad de cada ítem.

También podemos realizar un bucle con el podemos recorrer la lista y utilizar un índice, la expresión es la siguiente:

```
{#each expression as name, index}...{/each}
```

Esto nos servirá para conseguir cada valor y su posición, empezando desde cero, de la lista. Veamos un ejemplo.

```
{#each items as item, i}
  <li>{i + 1}: {item.name} x {item.qty}</li>
{/each}
```

Esperas

Svelte proporciona una sintaxis de plantilla llamada `{#await}` que te permite trabajar directamente con la promesa sin necesidad de preocuparte en guardar diferentes estados para trabajar con la promesa, lo que **simplifica mucho el código**. La sintaxis es la siguiente:

```
{#await expression}...{:then name}...{:catch name}...{/await}
```

Para ver cómo nos podría ayudar, vamos a ver un ejemplo de forma que haga uso de esta sintaxis.

```
<script>
  const fetchImages = async () => {
    const response = await fetch('https://digitalnet/items/images/')
    return response.json() // directly return the promise
  }
  // store the promise directly in a variable
  const imagesPromise = fetchImages()
</script>

{#await imagesPromise}
  <p>...loading</p>
{:then data}
  <img src={data.image} alt="Digitalnet" />
{:catch error}
  <p>The error is {error}</p>
{/await}
```

Se realiza una llamada a la función `fetchImages` la cual realiza una petición hacia una API para obtener una imagen. Si no obtiene los datos aún entonces se mantiene en el `await`, si obtiene los datos entonces entrará en `then`, pero si salta un error en la petición entrará en el `catch`

Constantes

Svelte proporciona una sintaxis de plantilla llamada `{@const}` que te permite definir una constante local en ejecución del código. La expresión es la siguiente:

```
{@const assignment}
```

Veamos un ejemplo para entender el uso de esta sintaxis.

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

{#each boxes as box}
  {@const area = box.width * box.height}
  {box.width} * {box.height} = {area}
{/each}
```

Crea una constante llamada *area* en la cual almacena el resultado de multiplicar la anchura por la altura del cuadrado para mostrar su resultado en la siguiente línea.

Directivas

on:eventage

Svelte proporciona una directiva de elemento llamada *on* que permite escuchar eventos DOM. Las sintaxis son las siguientes:

```
on:eventname={handler}
```

```
on:eventname|modifiers={handler}
```

Veamos un ejemplo de su uso.

```
<script>
  let count = 0;

  function handleClick(event) {
    count += 1;
  }
</script>

<button on:click={handleClick}>
  count: {count}
</button>
```

Como podemos entender, cada vez que se pulse el botón llamará a la función *handleClick*, la cual incrementará el valor en uno del mismo modo que un contador.

Están disponibles los siguientes modificadores:

- `preventDefault`— llamadas `event.preventDefault()` antes de ejecutar el controlador
- `stopPropagation`— llamadas `event.stopPropagation()`, evitando que el evento llegue al siguiente elemento
- `passive`— mejora el rendimiento de desplazamiento en eventos táctiles/de rueda (Svelte lo agregará automáticamente donde sea seguro hacerlo)
- `nonpassive`- establecido explícitamente `passive: false`
- `capture`— dispara al manejador durante la fase de *captura* en lugar de la fase de *burbujeo*
- `once`— quitar el controlador después de la primera vez que se ejecuta
- `self`— solo activa el controlador si `event.targetes` el elemento en sí
- `trusted`— solo activa el controlador si `event.isTrusted` es `true`. Es decir, si el evento se desencadena por una acción del usuario.

bind:property

Los datos normalmente fluyen hacia abajo, de padre a hijo. **Svelte** proporciona una directiva de elemento llamada *bind* que permite que los datos fluyan al revés, de hijo a padre. Las sintaxis son las siguientes:

`bind:property={variable}`

Veamos un ejemplo de cómo se usa:

```
<input type="number" bind:value={num}>
```

Realizamos un input tag en el que guardamos el valor de un número que hemos definido.

Ejemplo con SvelteKit

Ahora pasaremos a realizar un pequeño tutorial en el cual usaremos el lenguaje Svelte para crear una aplicación web.

Para poder realizar el tutorial, necesitaremos:

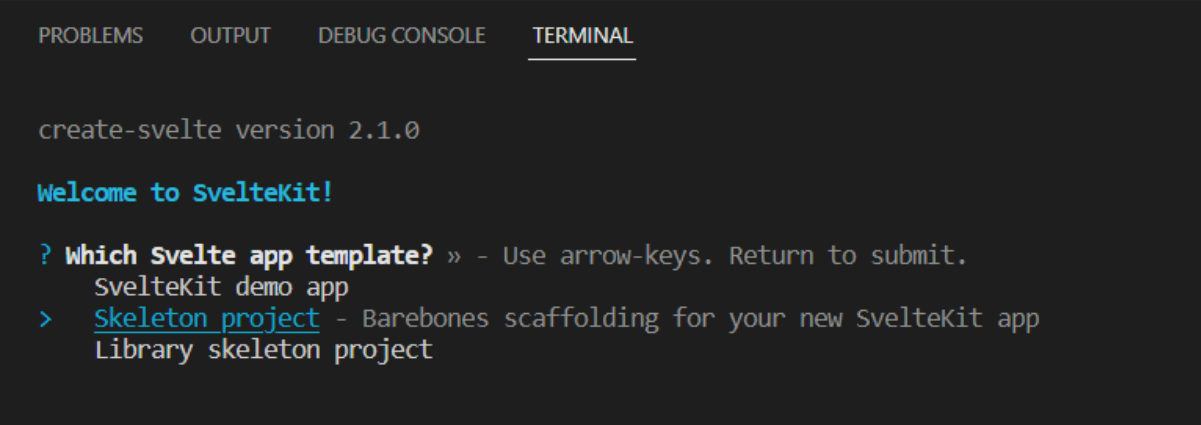
- **Visual Studio Code**
- **SvelteKit**¹ (Se podrá instalar durante el tutorial)

1. Creación y preparación del proyecto

Para comenzar, abriremos Visual Studio Code y abriremos la carpeta que queramos que contenga nuestro **proyecto**. Una vez abierta, abriremos la consola y escribiremos lo siguiente:

```
npm create svelte@latest myapp
```

Este comando creará un proyecto, con el nombre de “myapp”, usando SvelteKit, si no lo tienes instalado aparecerá un mensaje indicando de que falta dicho framework y **te permitirá elegir si deseas instalarlo o no**. Le daremos ‘y’ para aceptar la instalación y procederá a mostrar la siguiente pantalla:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

create-svelte version 2.1.0

Welcome to SvelteKit!

? Which Svelte app template? » - Use arrow-keys. Return to submit.
  SvelteKit demo app
> Skeleton project - Barebones scaffolding for your new SvelteKit app
  Library skeleton project
```

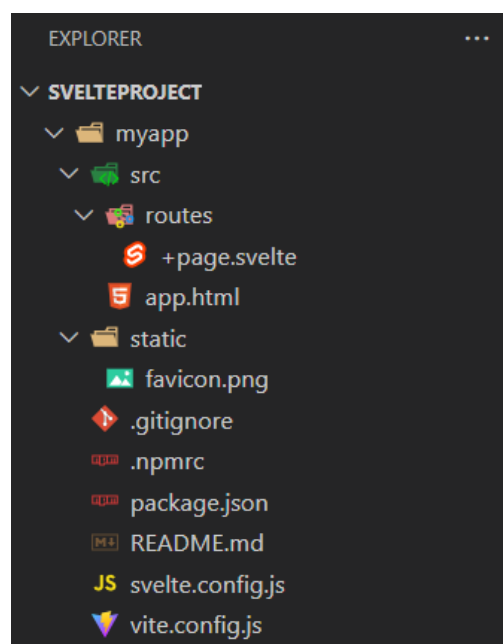
Seleccionamos la opción **Skeleton project** y le daremos a que “No” a las siguientes opciones que nos ofrezca como se muestra a continuación:

¹ SvelteKit es un framework que hace uso de Svelte y Vite que nos permite crear una aplicación web.

Welcome to SvelteKit!

```
✓ Which Svelte app template? » Skeleton project
✓ Add type checking with TypeScript? » No
✓ Add ESLint for code linting? ... No / Yes
✓ Add Prettier for code formatting? ... No / Yes
✓ Add Playwright for browser testing? ... No / Yes
✓ Add Vitest for unit testing? ... No / Yes
```

Una vez creado el proyecto tendremos la siguiente estructura en la ventana **Explorer**:



Desde la consola entramos en la carpeta “myapp” usando el siguiente comando:

```
cd .\myapp\
```

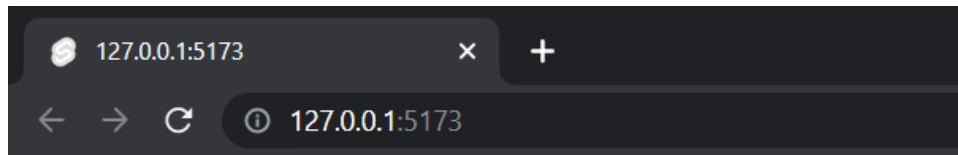
Y le instalaremos las dependencias necesarias usando el comando:

```
npm i
```

Ahora podremos ejecutar el proyecto web usando el comando:

```
npm run dev
```

Nos mostrará el enlace para acceder a la aplicación web, lo seguiremos y en la página veremos lo siguiente:



Welcome to SvelteKit

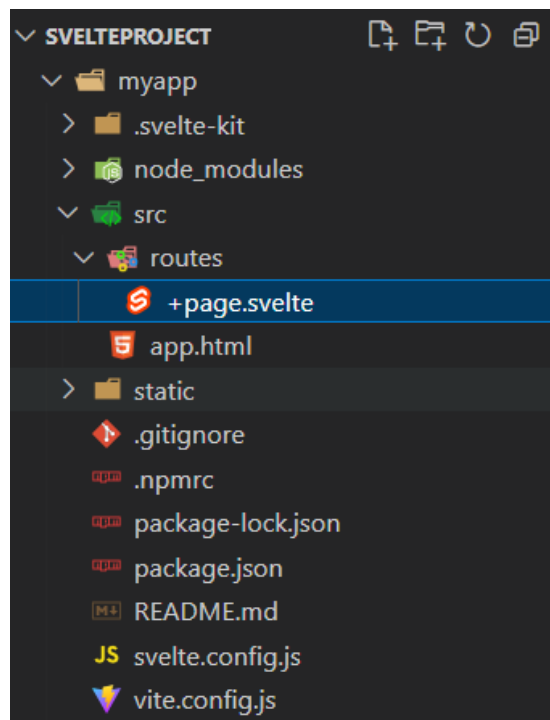
Visit kit.svelte.dev to read the documentation

¡Enhorabuena! ¡Ya tenemos el esqueleto del proyecto de forma totalmente funcional!

Para cerrar el servidor usaremos la combinación de teclas **Ctrl+C** en la consola para frenar la ejecución del servidor.

2. “¡Hola Mundo!”

Ahora que ya tenemos la básico, podemos empezar a picar código. Lo siguiente que haremos será mostrar un “¡Hola Mundo!” en la página. Para ello nos desplazamos a la página que contenga la información a mostrar. En nuestro caso será “**+page.svelte**” (la página principal) que se encuentra dentro de src/routes como se muestra a continuación:



Borramos todo el contenido de esta página ya que comenzaremos desde cero. Añadiremos lo siguiente:

```
<script>

</script>

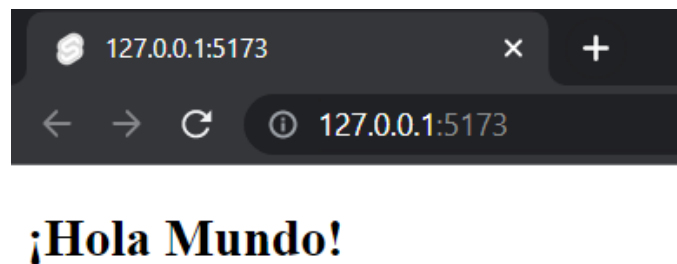
<h1>¡Hola Mundo!</h1>

<style>

</style>
```

Podemos escribir directamente lo que queramos que se muestre en la web sin meterlo dentro de algún tag. Notamos los siguientes tags: **script**² y **style**³.

Cuando ejecutemos el servidor con **npm run dev** veremos en la web lo siguiente:



Ya tenemos un **¡Hola Mundo!** mostrándose en la página principal de la aplicación.

3. Tratando con variables y estilos

Ahora aprenderemos a tratar con variables y a estilizar componentes HTML.

De primeras, dentro de el tag **<script>** crearemos una variable “name” y le asignaremos la string “**Juan**” de la siguiente forma:

```
<script>
  let name = 'Juan';
</script>
```

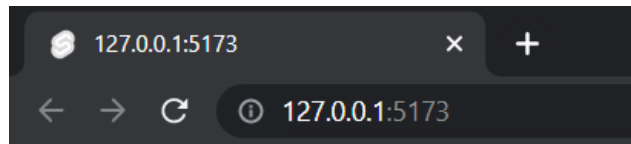
² El tag **<script>** contendrá la lógica/JavaScript que se ejecuta cuando se crea la página.

³ El tag **<style>** contendrá el CSS/estilos que se necesitan en la misma página en la que se encuentra.

Ahora que tenemos la variable “name” para la página, modificaremos el tag **<h1>** que teníamos anteriormente y lo dejaremos así:

```
<h1>¡Hola {name}!</h1>
```

Se puede obtener la información de las variables haciendo uso de los corchetes (**{ }**) como se muestra anteriormente con **{name}**. Una vez se ejecute la página veremos que se muestra:



¡Hola Juan!

Ahora, modificaremos el estilo del **h1** para que su texto sea de diferente color, para ello dentro de **<style>** añadiremos lo siguiente:

```
<style>
  h1 {
    color: blue;
  }
</style>
```

Lo cual modificará el color del texto de cualquier **h1** que se encuentre en esta página a azul. Si por algún casual quisiéramos solo modificar un solo elemento, podríamos, como de costumbre en CSS, editarlo directamente cuando se declara el **h1**:

```
<h1 style="color: blue;">¡Hola {name}!</h1>
```

O podríamos dentro de la etiqueta **<style>** asignar como una clase personalizada:

```
...
<h1 class="abc">¡Hola {name}!</h1>

<style>
  h1.abc{
    color: blue;
  }
</style>
```

Por último, si nos interesa que la variable **name** esté en mayúsculas, se puede modificar desde que se llama (en **h1** en nuestro caso) en vez de hacerlo desde **<script>**, para ello pondremos:

```
<h1 class="abc">¡Hola {name.toUpperCase()}!</h1>
```

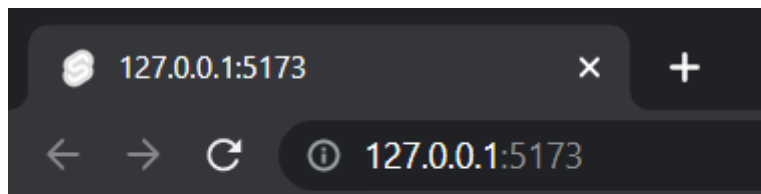
Podemos usar funciones de las variables justo cuando se llamen en el HTML. El resultado de toda la configuración aplicada será la siguiente:

```
<script>
  let name = 'Juan';
</script>

<h1 class="abc">¡Hola {name.toUpperCase()}!</h1>

<style>
  h1.abc{
    color: blue;
  }
</style>
```

Dejando la página así:



¡Hola JUAN!

4. Crear una página

Ahora que sabemos tratar con lo que se muestre por pantalla, **crearemos una página** que contendrá un futuro contador (como todo tutorial de creación web que se aprecie).

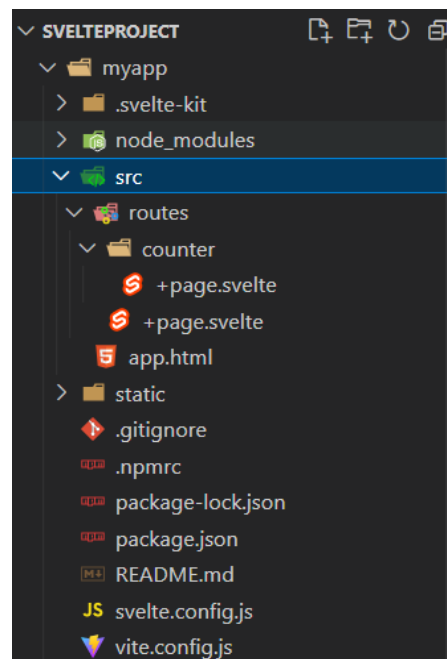
Para ello crearemos una carpeta llamada “counter” dentro de “src/routes” y dentro de esta nueva carpeta crearemos un archivo llamado “**+page.svelte**” (que representará el contenido de la nueva página, recuerda que existen nombre). En este nuevo archivo por ahora solo pondremos lo siguiente:

```
<h1>Esta será la página contador</h1>
```

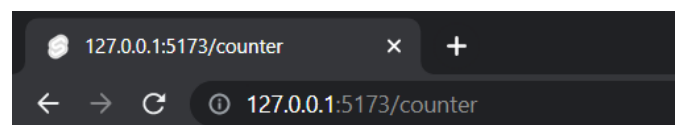
Ahora, volveremos al archivo “**+page.svelte**” contenido en la carpeta “routes” y crearemos un enlace a la nueva página escribiendo lo siguiente:

```
<a href="/counter">Acceso a contador</a>
```

En acabar, la estructura del proyecto será la siguiente:



Y a través de la página principal seremos capaces de llegar a la nueva página:



Esta será la página contador

5. Funciones y eventos

Ahora que ya tenemos una página nueva, crearemos el típico contador en dicha haciendo uso de la declaración de funciones y el uso de eventos.

Primero escribiremos lo siguiente en la nueva página “**counter**”:

```
<script>

</script>

<h1>Esta será la página contador</h1>

<style>

</style>
```

Ahora definiremos una variable (**counter**) y una función (**sumCounter**) para incrementar en uno dicha variable:

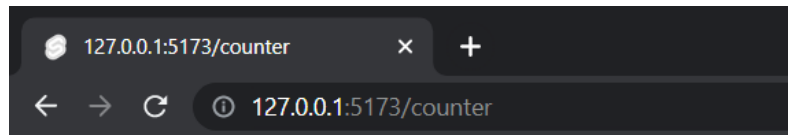
```
<script>
    let counter = 0;

    function sumCounter(){
        counter+=1
    }
</script>
```

Ya declarada la función, solo nos falta ejecutarla cuando se realice alguna acción, en este caso, crearemos un botón el cual llamará a dicha función cada vez que se clique:

```
<h1>Esta será la página contador</h1>
<h2>{counter}</h2>
<button onclick={sumCounter}>+1</button>
```

Como las variables, las funciones hacen uso de corchetes para ser declaradas. La página contador se mostrará de la siguiente manera:



Esta será la página contador

0



Cada vez que clickeamos en el botón **+1** se sumará 1 al contador

6. Reactividad

La **reactividad** que nos ofrece Svelte es muy potente, pues nos ofrece incluso reactividad en componentes compuestos de otros de forma rápida y simple.

Ya tenemos un contador, ahora imaginemos que quisiéramos tener una “string” que contiene el contador como parte de la misma para mostrarla en la página, lo que pensamos de primeras que tendríamos que hacer sería añadir lo siguiente:

```
<script>
  ...

  let info = "Esta es la cantidad de veces que has clickado: " + counter;
</script>

  ...

<h3>{info}</h3>
```

Parece correcto pero NO lo es, ya que solo la variable “**counter**” es la que cambia mientras que “**info**” solo es declarada cuando se crea la página (por lo tanto solo será modificada cuando se inicialice la página). Si quisiéramos que “**info**” también cambie, tendremos que declararlo de la siguiente manera en **<script>**:

```
$: info = "Esta es la cantidad de veces que has clickado: " + counter;
```

“**\$:**” es una declaración reactiva, la cual obligará a actualizar (en nuestro caso) la variable “**info**” cada vez que algo que la compone sea modificado (“**info**” se actualizará cada vez que “**counter**” sea modificado). Ahora sí, la variable “**info**” será reactiva. Tenemos que entender que NO todo es reactivo de forma natural, sino que debemos de tener cuidado en este tipo de casos.

7. Condicionales

Con Svelte, también podemos usar condicionales que dependen de las variables definidas. En este caso, queremos un mensaje cuando hayan menos de 10 clicks, otro con 10 y finalmente otro cuando el contador lleve más de 10. Para ello haremos uso de expresiones `{#if ...}` las cuales nos permitirá seleccionar qué mensaje queremos que se muestre de la siguiente manera:

```
{#if counter < 10}
  <p>Has clickado menos de 10 veces</p>
{:else if counter == 10}
  <p>Has clickado 10 veces</p>
{:else}
  <p>Has clickado más de 10 veces</p>
{/if}
```

De esta forma se comprobará el valor de la variable “**counter**” y se mostrará el mensaje pertinente.

8. Animaciones

Por último, tenemos animaciones. Svelte nos lo pone fácil para añadirlas. Tan solo tendremos que añadir en `<script>`:

```
import {fade, fly} from 'svelte/transition'
```

Y ya podremos acceder a un par de animaciones de forma fácil y rápida. Para hacer uso de estas, tendremos que modificar el componente que queramos tenga animación.

```
{#if counter < 10}
  <p>Has clickado menos de 10 veces</p>
{:else if counter == 10}
  <p in:fly={{x: 1000, duration: 500}} out:fly={{x: 1000, duration: 500}}>Has clickado 10 veces</p>
{:else}
  <p transition:fade>Has clickado más de 10 veces</p>
{/if}
```

¡Ya le hemos dado animación al segundo y tercer mensaje!

9. Código final

Aquí podrás ver cómo queda el código de la página principal y de la página “counter” al final del tutorial con todos los cambios realizados:

src/routes/+page.svelte

(Página principal)

```
<script>
  let name = 'Juan';
</script>

<h1 class="abc">¡Hola {name.toUpperCase()}!</h1>
<a href="/counter">Acceso a contador</a>

<style>
  h1.abc{
    color: blue;
  }
</style>
```

src/routes/counter/+page.svelte

(Página “counter”)

```
<script>
  import {fade, fly} from 'svelte/transition'
  let counter = 0;
  $: info = "Esta es la cantidad de veces que has clickado: " +
counter;
  function sumCounter(){
    counter+=1
  }
</script>

<h1>Esta será la página contador</h1>
<h2>{counter}</h2>
<button on:click={sumCounter}>+1</button>
<h3>{info}</h3>
{#if counter < 10}
  <p>Has clickado menos de 10 veces</p>
{:else if counter == 10}
  <p in:fly={{x: 1000, duration: 500}} out:fly={{x: 1000, duration:
500}}>Has clickado 10 veces</p>
{:else}
  <p transition:fade>Has clickado más de 10 veces</p>
{/if}
```

REFERENCIAS

<https://svelte.dev/>

<https://es.wikipedia.org/wiki/Svelte>

<https://midu.dev/introducci%C3%B3n-a-svelte/>

<https://ed.team/blog/que-es-sveltejs-1>

<https://kit.svelte.dev/docs/introduction>

 Svelte 3 Reaction & QuickStart Tutorial