Proyecto SVELTE

Video 2

Instalación:

- instalar npm (brew install npm)

- instatar degit (npm install -g degit)

Crear un proyecto Svelte:

degit sveltejs/template <nombre-del-proyecto>

en el directorio del proyecto:

npm install

corremos el server por primera vez

npm run dev

Video 3

Alright then, gang, so before we write any code whatsoever, let's have a little look at how everything works in Svelte underneath the hood.

As I said in the last video, we create all of our source code inside the `src` folder. This is where you're going to be writing 99% of your code, and where we create all of our different Svelte components. We have one already created for us, `App.svelte`. Now, components are like building blocks of a website, which can be put together and then output to the DOM or the browser to create a whole website. For example, we might have a component for a header, a contact form, a modal, a footer, a pop-up, etc. At the end, each of these components can come together and be injected into the DOM to make a complete website.

Svelte has already created a root component for us called `App.svelte`, and by the way, whenever we create a component, it must have the `.svelte` extension so Svelte knows that this is a component and can compile it accordingly.

Inside each component, we have three different things. We have a script at the top for any kind of component logic, we have an HTML template which is the content that is ultimately injected into the DOM, and then we also have a style tag at the bottom to style the template of this component.

Now, moving on to the `main.js` file. This is the file that kick-starts the app. It contains the code that runs first and sets everything up. First of all, it imports the `App.svelte` component that's created over here. When we create a component, it's automatically exported, and we can import it into another file. So, we're importing `App.svelte` first, then we create a new instance of that component. Inside that instance, there's this object with a `target` property. This is basically saying, "Okay, we have this `App` component instance now, where do you want to inject it into the DOM?" We're saying we want to grab the `body` tag and put this app component—so all of this template—inside it.

This is the selector for that tag, but we could use a query selector if we wanted to target a different tag. Generally, though, we're going to use the `body` tag because that's where all the content is kept in an HTML document. We have this `index.html` file in the `public` folder, and that's the file served to the browser. If we inspect the element, we can see the HTML content.

What the `main.js` file is doing is grabbing the `body` tag from the `index.html` file and injecting all of the app templates into that tag. The `props` property is used to pass data into a component. For now, we're passing in a `name` property with a value of "world." We’ll talk more about props later, but essentially, if we go to `App.svelte`, we can see we're grabbing that `name` value here. The `export` denotes that the `name` is being set from a prop outside of the component.

I don't want to talk too much about props for now, so I'll delete this `export` statement and we'll revisit them later. Hopefully, now you understand the general idea of how this works: we create components, and the component is then injected into the DOM. In the future, when we create more components, we're generally not going to inject them directly into the DOM this way. Instead, we'll nest them inside this root component, which is the top-level component that is injected directly into the DOM. After that, we can nest other components inside the root component.

When we build our project, Svelte looks at all of our components, compiles them into a single JavaScript bundle, and outputs it into the `build` folder, along with a `bundle.css` file. All of the scripts and styles from our different components are bundled together into these single files. From our `index.html`, we link to these files, and that's how it all works.

Now that we know how it works under the hood, let's try writing some code. We've already seen this where we're outputting a `name` variable that is defined here in the script. Initially, it was passed as a prop, but we deleted that export statement, so if we save it now, it would be `undefined` and would output nothing. Let's give it a value instead. I'll say `let name = 'yo'`, for example, and then to output a variable inside the template, we just use curly braces like so. If we save that, we'll see "Hello yo."

I could create another one. Let's call it `beltColor = 'black'`. Then, to output that, I'll delete the current content inside the paragraph tag and output `beltColor` instead, with the word "belt" after it. Now, if I save that and preview it, we'll see "black belt."

This is how we output any variables we define in the script inside the template. Now, what if we wanted to change the data later, or react to a user event, like a button click? We can do that by creating a button and setting up an `on:click` event handler.

For example, I’ll say:

```html

<button on:click={handleClick}>Update belt color</button>

```

Then, we define the `handleClick` function above:

```javascript

const handleClick = () => {

beltColor = 'orange';

};

```

This means when the button is clicked, the `beltColor` will change to orange, and the page will react to that change by updating the displayed color. If I save it now and click the button, it will update to orange.

So, this is Svelte syntax. It makes it easy to output dynamic data, change data, and react to user events like click events on buttons. When we build our project, Svelte compiles this syntax into regular JavaScript, and that’s what gets output in the bundle.

Now that we’ve learned the basics of Svelte—how to define data in a component, how to output it, and how to handle click events—let’s explore imports and data binding in the next video.