Frontend design

Foreword

In this session, we are going to learn about how to build a modern web app frontend with `Svelte`, a Javascript framework that is most beloved by the community during the time of writing. This is probably the most important part of this course. One reason is because the frontend is what your users see and interact with, so by extension it is what your user care. No matter how great your package is, if the user cannot use it or do not want to use it, it is not a good software. I am over commiting the term "frontend" here, because I am not only talking about the user interface, but also the user experience. The user interface is what the user see, while the user experience is how the user feel when they are using your software. A lot of scientific software do not aim to have a web frontend to it, and that is completely legitimate since the targeted users are other scientists and researchers who will look into their code. Nonetheless, a good user experience is not excusable even for the packages that do not require a web frontend. Another reason, perhaps a more legitimate reason, is because there are simply too many options to build your frontend. There are many javascript frameworks to choose from, `React`, `Angular`, `Vue` are the top three javascript frames, and I have dealt with all three of them. You can also write your frontend with `rust`, and there are frameworks like `yew`, `leptos`, `dioxus`. Recently `python` make a big splash in frontend development because of `FastHTML`. This list goes on and on. Then once your are done with choosing your javascript frameworks, it is time to choose your CSS framework. There are `Bootstrap`, `Tailwind`, `Material-UI` just to name a few. **Then**, you can choose your **metaframework** to handle routing and bundle your website, popular choices are `Next.js`, `Gatsby`, `Nuxt.js`, and here we go again. The point is, there are way too many choices, and I have spent so many hours just crawling through this holy war battlefield to try to build my frontend for some projects. In the end, it does not matter to the customer what framework you choose, as long as it works and it looks good, it is a good frontend. The reason I think this session is the most important one, it is because it has the biggest margin of time saving for you. Out of all the experience, I find `Svelte` to be the most fun to work with, and can convert what I think into what I see the most efficiently, so we are going to stick with the `Svelte` framework.

After I made this note, Svelte 5 came out and change some of its syntax. I migrated some of the Svelte part to Svelte 5, but there could be bug here and there. If things go south, then consult the Svelte tutorial.

Key concept

There are 3 main components in building a frontend: HTML, CSS, and Javascript. HTML defines the structure and content of the web page, CSS defines the style of the web page, and Javascript defines the interactivity of the web page.

Basic HTML

HTML stands for Hyper Text Markup Language. It is where your content goes. You can just write in HTML and open it in your browser, and you will see your content. A minimal HTML file looks like this:

```
<!DOCTYPE html>
<html>
<head>
```

```
  <title>Page Title</title>
</head>
<body>

<h1>This is a Heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

Trying opening this in your browser, you will see a heading and a paragraph. There are many tags in HTML and you can go very far with just HTML. You can find references of HTML tags in W3Schools, but these days it is probably easier to just ask ChatGPT.

Styling with tailwind CSS

While HTML gives you the functionality of the webpage, it does not give you the style. Here is where CSS comes in. CSS stands for Cascading Style Sheets, and it defines the style of the webpage, which includes the color, the font, the size, the position of the elements, etc. A minimal CSS file looks like this:

```
body {
  background-color: lightblue;
}

h1 {
  color: white;
  text-align: center;
}

p {
  font-family: verdana;
  font-size: 20px;
}
```

You can include this CSS file in your HTML file like this:

```
<!DOCTYPE html>
<html>
<head>
  <title>Page Title</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
...
```

However, keeping track of CSS in a separate file means you have to go back and fro to change the style of a particular element. To make this easier, people have made a number of CSS frameworks that helps the process. I personally uses `Tailwind CSS` because it is very easy to use and it keeps the style in the HTML file, so you can see the style of the element right next to the element. Here is an example of how you can use `Tailwind CSS` (Add

`<link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css" rel="stylesheet">` to the head of your HTML file):

```
<h1 class="text-4xl text-center text-blue-500">This is a Heading</h1>
<p class="font-sans text-lg">This is a paragraph.</p>
```

Javascript with Svelte

Finally we arrive at Javascript, which is where the interactivity of the webpage comes from. You can add interactivity to our HTML example like this:

```
<!DOCTYPE html>
<html>
<head>
  <title>Page Title</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

<h1>This is a Heading</h1>
<p>This is a paragraph.</p>

<button onclick="myFunction()">Click me</button>

<script>
function myFunction() {
  alert("Hello World!");
}
</script>

</body>
</html>
```

This will create a button that when clicked, will show an alert box with the message “Hello World!”.

Fundamentally Javascript is a language, but really when you hear about Javascript these days, it is more a discussion around which Javascript framework people are using. `React`, `Angular`, `Vue` are the big three in terms of popularity. I used to use `React` and `Vue` for my website, but I always felt like I don’t like them as much as I want to, and there were something not quite right when I use them. Then I found `Svelte`, which is a Javascript framework that is most beloved by the community during the time of writing. `Svelte` is a compiler that takes your code and compiles it into vanilla Javascript, HTML, and CSS. So in a sense you are not strictly writing Javascript, but you are writing Svelte, and this can be seen as you will write file with `.svelte` extension instead of `.js` extension. A minimal Svelte file looks like this:

```
<script>
  let name = $state('world');
</script>

<h1>Hello {name}!</h1>
```

Isn't this simple? We will go through the basic of Svelte in the next section.

Buidling your website with SvelteKit

Setting up the project

Make sure you have `node` and `npm` installed on your machine. You can check if you have them installed by running `node -v` and `npm -v` in your terminal. If you don't have them installed, you can download them from here. Then you should be able to find the step by step guide for this tutorial on this repository.

Write a Hero page for yourself

In this session, we are going to write a simple hero page for yourself. A hero page is the first page that the user sees when they visit your website. We are going to write a simple hero page that explores a number of svelte features following the steps below:

Defining and using a variable

The first thing we are going to do is to checkout the basic of writing a svelte file such as defining variables and showing them to the user. Look at your `src/routes/+page.svelte` file, and you should see some bolier plate codes like:

```
<h1>Welcome to SvelteKit</h1>
<p>Visit <a href="https://kit.svelte.dev">kit.svelte.dev</a> to read the documentation</
p>
```

Replace them with the following code:

```
<script>
  let name = $state('world');
</script>

<h1>Hello {name}!</h1>
```

If you have `npm run dev` running in your terminal, you should see the changes in your browser. You should see a heading that says "Hello world!". You can change the value of `name` to anything you want, and the heading will change accordingly.

Importing components

Sometimes you may find yourself reusing the same code in different places. In this case, you can create a component and reuse it. And you may find yourself wanting to pass some data to the component. In this case, you can use props. Let's create a component that takes a `name` prop and display it. Create a new file `src/components/Hello.svelte` and write the following code:

```
<script>
  let {name} = $props();
</script>

<h1>Hello {name}!</h1>
```

Then you can use this component in your `src/routes/+page.svelte` file like this:

```
<script>
  import Hello from '$lib/components/Hello.svelte';
</script>

<Hello name="world" />
<Hello name="svelte" />
```

Logics

In some situation you may want to display different content or style based on some condition. You can use `if` statement to do this. Let's say you want to display a different message based on the value of `name`. You can do this like this:

```
<script>
  let name = $state('world');
</script>

{#if name === 'world'}
  <h1>Hello {name}!</h1>
{:else}
  <h1>I don't say hello to anyone else</h1>
{/if}
```

Try changing this snippet to work with the `Hello` component we created earlier. There are more logics you can use in Svelte, such as `each`, `await`, `then`, `catch`, etc. You can find more information about them in the Svelte documentation.

Events and binding

Now it is time to add a bit of interactivity to our page. Let's add a button that track the number of times it is clicked. You can do this like this:

```
<script>
  let count = 0;

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

<button on:click={handleClick}>
  Clicked {count} {count === 1 ? 'time' : 'times'}
</button>
```

This is a way to let data flow from the javascript to the html for display. However, sometimes you may want to go the other way around Another common use case is to bind the value of an input field to a variable. You can do this like this:

```
<script>
  let name = $state('world');
</script>

<input bind:value={name} placeholder="Enter your name" />
<p>Hello {name}!</p>
```

Frontend design

Now given the code above, try to make a page that says hello to the user when they enter their name in the input field, together with a poke button. If the poke button is poked more than 10 times, the page should say "You are annoying".

Setting up routes

Often we want to have multiple pages in our website. This is usually done through the concept of route. A route is a path that the user can visit to see a different page. In SvelteKit, you can define routes in the `src/routes` folder. If you want to create a new page, you would create a new folder in the `src` folder, then add a `+page.svelte` file in it. To test this out, let's try to add a simple navigation bar to our website.

Creating a navigation bar

Create a new file `src/components/Navbar.svelte` and write the following code:

```
<nav>
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/about">About</a></li>
  </ul>
</nav>
```

Then you can include this component in your `src/routes/+page.svelte` file like this:

```
<script>
  import Navbar from '$lib/components/Navbar.svelte';
</script>

<Navbar />
```

Now you should see a navigation bar at the top of your page. If you click on the "About" link, you should see a 404 page. This is because we have not defined the route for the "About" page yet. Let's do that now.

Defining the About page

Create a new folder `src/routes/about` and add a `+page.svelte` file in it. Write the following code in the `src/routes/about/+page.svelte` file:

```
<h1>About page</h1>
<p>This is the about page</p>
```

Now if you click on the "About" link in the navigation bar, you should see the about page. In your browser, you should see the URL change to `http://localhost:5173/about`.

Layouts

If you have navigate to the "About" page, you may notice that the navigation bar is missing. This is because the layout of the "About" page is different from the layout of the "Home" page. A layout file defines the layout of a page, such as the header, footer, and navigation bar. In SvelteKit, a layout file for a route can be defined with the `+layout.svelte` file in the route folder. And the layout file will be applied to all the child routes of the route folder. Let's create a layout file for our website. Let's move our navbar to the top level layout file.

Create a new file `src/routes/+layout.svelte` and move the navbar code to it:

```
<script>
  import Navbar from '$lib/components/Navbar.svelte';
</script>

<Navbar />

<slot />
```

The `<slot />` tag is where the content will be inserted. Now you can remove the navbar from the `src/routes/+page.svelte` file. If you navigate to the "About" page now, you should see the navigation bar at the top of the page.

Adding style

Now we are going to add some style to our website with `Tailwind CSS`.

1. Follow steps 4 –7 in the Tailwind CSS documentation to install Tailwind CSS in your project.
2. You should see the font becomes very different already once you run `npm run dev`.
3. Now let's try to change the font size of the hello world heading to 4xl. You can do this by adding the `text-4xl` class to the heading like this:

```
<h1 class="text-4xl">Hello {name}!</h1>
```

4. You can also change the color of the heading by adding the `text-blue-500` class to it:

```
<h1 class="text-4xl text-blue-500">Hello {name}!</h1>
```

5. The next thing we can do is to center the heading. You can do this by adding the `text-center` class to it:

```
<h1 class="text-4xl text-center text-blue-500">Hello {name}!</h1>
```

Adding a reactive plot

Now we are going to add a plot to the main page using D3.js.

1. Here is an example on Svelte's website. Copy this to your `src/routes/+page.svelte` file.
2. The next step is to install `d3` and `d3-scale` in your project. You can do this by running the following command in your terminal:

```
npm install d3 d3-scale
```

3. After this, add an input field to the page that allows the user to change the data of the plot. Bind the input to one of the birth rate and choose number input type.
4. Click and see if it works!

Other tools

Frontend design

UI/UX design is very much about finding inspiration from the internet. It is impossible for me to teach you everything about UI since that will take at least months if not years. However, here are some tools you should look into if you want to expand your knowledge about Web UI design

Figma / Penpot

shadcn/ui

Wix