Lab 5

Svelte Review

As a refresher of the concepts studied, make sure to check out the official tutorial (https://svelte.dev/tutorial) which has more information than you need in an interactive guide to practice the various features Svelte provides. You can also find more examples (https://svelte.dev/examples), and the full documentation (https://svelte.dev/docs) on the same website.

When it comes to actually writing Svelte code, you can use the Svelte REPL (https://svelte.dev/repl) which also allows you to inspect the output JS and CSS if you want. However, we are going to jump straigt to coding it standalone.
If you are using VS Code, install the Svelte extension (https://marketplace.visualstudio.com/items?itemName=svelte.svelte-vscode) to get syntax highlighting and IntelliSense support.

To create a new Svelte project that supports TypeScript, you can run the following commands (replacing `my-app` with your app's name):

```
npm create vite@latest my-app
```

In the interactive prompt, select Svelte first, then TypeScript (do **not** choose `SvelteKit`). Now, just run `cd my-app` and then `npm install` and then `npm run dev` to open a development server. We no longer need the "Live Server" extension!

Vite

Note that this time, we're using a new bundler instead of Rollup: Vite. Vite actually uses Rollup under the hood when building for production, and ESBuild during development to speed up boot times.

The main differences with Vite are that it is configured out-of-the-box with the following features (https://vitejs.dev/guide/features.html):

- NPM dependency resolution and pre-bundling
- TypeScript support (compilation only, not type-checking)
- Hot Module Replacement
- Importing CSS and static assets (images, JSON, …)
- and lots more…

Preprocessing

When you setup the Svelte+TypeScript template, you will notice a new file called `svelte.config.js` that calls `vitePreprocess()`. By default, the Svelte compiler supports only plain HTML/CSS/JS (with Svelte’s syntax, of course) in its single-file components. To add support for different languages, we used to use Svelte Preprocess (https://github.com/sveltejs/svelte-preprocess) package in the past, which allows to transform the content of the Svelte components from these other languages to the syntax that Svelte’s parser understand, but now this functionality is available in Vite itself.

These preprocessors include:

- TypeScript, Babel, CoffeeScript (in the `script` tag)
- Pug (for the markup)
- Sass, Less, PostCSS (in the `style` tag)
- and more…

It is also possible to define custom preprocessors if necessary.

Task

To practice some Svelte, let us rewrite the task of the previous lab (real-time chat app) using Svelte and SFCs.

First, we define a type for messages that will be used elsewhere: **`message.ts`**:

```
export interface Message {
  username: string;
  message: string;
};
```

Then, we encapsulate the message with its styles in a reusable component: **`Message.svelte`**:

```
<script lang="ts">
  import type { Message } from './message';

  export let message: Message;
</script>

<li>
  <span style="font-weight: bold;">{message.username}: </span>
  {message.message}
</li>

<style>
  li {
    padding: 0.5rem 1rem;
  }
  li:nth-child(odd) {
    background-color: #f8f8f8;
  }
</style>
```

And then define another component for the input bar at the bottom `Input.svelte`:

```
<script lang="ts">
  import { createEventDispatcher } from 'svelte';
  import type { Message } from './message';

  const dispatch = createEventDispatcher<{ send: Message }>();

  let message = '';
  let username = '';

  function send(e: SubmitEvent) {
    if (!message || !username) return;
    dispatch('send', { message, username });
    message = '';
  }
</script>

<form on:submit|preventDefault={send} action="">
  <input autocomplete="off" bind:value={username} placeholder="Name" style
  <input autocomplete="off" bind:value={message} placeholder="Type your me
  <button>Send</button>
</form>

<style>
  form {
    background: rgba(0, 0, 0, 0.15);
    padding: 0.25rem;
    position: fixed;
    bottom: 0;
    left: 0;
    right: 0;
    display: flex;
    height: 3rem;
    box-sizing: border-box;
    backdrop-filter: blur(10px);
  }
  input {
    border: none;
    padding: 0 1rem;
    flex-grow: 1;
    border-radius: 2rem;
    margin: 0.25rem;
  }
  input:focus {
    outline: none;
  }
  form > button {
    background: #333;
    border: none;
    padding: 0 1rem;
    margin: 0.25rem;
    border-radius: 3px;
    outline: none;
    color: #fff;
  }
</style>
```

And then, to put it all together:
`App.svelte`:

```
<script lang="ts">
  import { io } from 'socket.io-client';
  import MessageItem from './Message.svelte';
  import Input from './Input.svelte';
  import type { Message } from './message';

  let messages: Message[] = [];

  const socket = io('http://10.90.137.175/');
  socket.on('chat message', function (msg: Message) {
    messages.push(msg);
    messages = messages;
    // messages = [...messages, msg];
    window.scrollTo(0, document.body.scrollHeight);
  });
</script>

<main>
  <ul>
    {#each messages as message}
      <MessageItem {message} />
    {/each}
  </ul>
  <Input on:send={(e) => socket.emit('chat message', e.detail)} />
</main>

<style>
  ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
  }
</style>
```

And that’s all!

Homework

Take the same website you’ve been working on for the past assignments and transform it to use Svelte with TypeScript. The components themselves will depend on the nature of your website, but it is expected that a minimum of 3 components will be used.