Assessment 3

1. Before you start

1.1. Background & Motivation

Web-based applications are becoming the most common way to build a digital capability accessible to a mass audience. While there are modern tools that help us build these rapidly, it's important to understand the fundamental JavaScript-based technology and architectures that exist, both to gain a deeper understanding for when these skills may be needed, but also to simply understand the mechanics of fundamental JS. Even when working with a high level framework like ReactJS, understanding (in-concept) the code that it is transpiled to will ensure you're a more well rounded web-based engineer.

This assignment consists of building a **frontend** website in Vanilla JS (no ReactJS or other frameworks). This frontend will interact with a RESTful API HTTP backend that is built in JavaScript (NodeJS express server) and provided to you.

A theoretical background on how to interface with this API can be found the "promises & fetch" lecture.

The web-based application you build is required to be a single page app (SPA). Single page apps give websites an "app-like feeling", and are characterised by their use of a single full load of an initial HTML page, and then using AJAX/fetch to dynamically manipulate the DOM without ever requiring a full page reload. In this way, SPAs are generated, rendered, and updated using JavaScript. Because SPAs don' t require a user to navigate away from a page to do anything, they retain a degree of user and application state. In short, this means you will only ever have index.html as your HTML page, and that any sense of "moving between pages" will just be modifications of the DOM. Failure to implement your site as a single page app will result in significant penalties.

3. Getting started

3.1. The Frontend

Stub code has been provided to help you get started in:

- frontend/index.html
- frontend/styles/global.css
- frontend/src/helpers.js
- frontend/src/main.js

You can modify or delete this stub code if you choose. It's simply here to potentially provide some help.

To work with your frontend code locally with the web server, you may have to run another web server to serve the frontend's static files.

To do this, run the following command once on your machine:

```
$ npm install --global http-server
```

Then whenever you want to start your server, run the following in your project's root folder:

```
$ npx http-server frontend -c 1 -p [port]
```

Where [port] is the port you want to run the server on (e.g. 8080). Any number is fine.

This will start up a second HTTP server where if you navigate to http://localhost:8000 (or whatever URL/port it provides) it will run your index.html without any CORs issues.

3.2. The Backend

You are prohibited from modifying the backend. No work needs to be done on the backend. It's provided to you simply to power your frontend.

The backend server can be cloned by running git clone git@nw-syd-gitlab.cseunsw.tech:COMP6080/[term]/ass3-backend.git where [term] is the current term (e.g. 24T3). After you clone this repo, you must run npm install in the project once.

To run the backend server, simply run npm start in the backend project. This will start the backend.

To view the API interface for the backend you can navigate to the base URL of the backend (e.g. http://localhost:5005). This will list all of the HTTP routes that you can interact with.

We have provided you with a very basic starting database containing two users and one public channel with messages. You can look in backend/database.json to see the contents.

Your backend is persistent in terms of data storage. That means the data will remain even after your express server process stops running. If you want to reset the data in the backend to the original starting state, you can run npm run reset in the backend directory. If you want to make a copy of the backend data (e.g. for a backup) then simply copy database.json. If you want to start with an empty database, you can run npm run clear in the backend directory.

Once the backend has started, you can view the API documentation by navigating to http://localhost:[port] in a web browser.

The port that the backend runs on (and that the frontend can use) is specified in frontend/src/config.js. You can change the port in this file. This file exists so that your frontend knows what port to use when talking to the backend.

Please note: If you manually update database.json you will need to restart your server.

Please note: You CANNOT modify the backend source code for bonus marks.

3.3. Taking the first steps

This is how we recommend you start the assignment:

1. Read the entire spec, including a thorough read of section 2 so you know what is ahead of you!
2. Try to load up the index.html on your browser with a simple "Hello world" text just to sanity check you know what page you're trying to load.
3. Plan out your UI by thinking about all of the key screens and what information they rely on
4. Try to load up the backend and verify you've got it working by navigating to the API documentation - Swagger http://localhost:5005 and testing some of the routes.
5. Good luck!

3.4. Making a fetch request

Here is some helpful starter code to make a POST request (for non-authenticated routes). Note: there are many other ways (and some cleaner than this) to do this, so don't assume this is perfect code. It will just help you get started.

```
const apiCall = (path, body) => {
   return fetch('http://localhost:5005/' + path, {
      method: 'POST',
      headers: {
         'Content-type': 'application/json',
      },
      body: JSON.stringify(body)
   })
   .then((response) => response.json())
   .then((data) => {
      if (data.error) {
         alert(data.error);
      } else {
         return Promise.resolve(data);
         // or handle the data here
      }
   });
};
```

Here is some helpful starter code to make a GET request (for authenticated routes). Note: there are many other ways (and some cleaner than this) to do this, so don't assume this is perfect code. It will just help you get started.

```
const apiCall = (path, token, queryString) => {
   return fetch('http://localhost:5005/' + path + '?' + queryString, {
      method: 'GET',
      headers: {
         'Content-type': 'application/json',
         'Authorization': `Bearer ${token}`
      },
   })
   .then((response) => response.json())
   .then((data) => {
      if (data.error) {
         alert(data.error);
      } else {
         return Promise.resolve(data);
         // or handle the data here
      }
   });
};
```

4. Constraints & Assumptions

4.1. Javascript

- You must implement this assignment in ES6-compliant Vanilla JavaScript. You cannot use ReactJS, JQuery, or other abstract frameworks. You cannot, for example, use a popular Javascript framework such as Angular or React.
- You may **NOT** directly use external JavaScript. Do not use NPM except to install any other development libraries without prior approval from course authority.

4.2. CSS and other libraries

- You may use small amounts (< 10 lines) of general purpose code (not specific to the assignment) obtained from a site such as Stack Overflow or other publically available resources. You should clearly attribute the source of this code in a comment with it. You can not otherwise use code written by another person.
- You may include external CSS libraries in this assignment (with the <link /> tag). You must attribute these sources (i.e. provide the URL/author in source code comments). For example, you are permitted to use the popular Bootstrap CSS framework. Some Bootstrap functionality relies on accompanying Javascript. You are permitted to include this Javascript. The Javascript accompanying Bootstrap requires the popular general purpose Javascrpt library jQuery. You are permitted to include **jQuery** so bootstrap can use it. However you are not permitted to use **jQuery** in the code you write for the assignment.

4.3. Browser Compatibility

You should ensure that your programs have been tested on one of the following two browsers:

- Locally, Google Chrome (various operating systems)
- On CSE machines, Chromium

4.4. Other Requirements

- The specification is intentionally vague to allow you to build frontend components however you think are visually appropriate. Their size, positioning, colour, layout, is in virtually all cases completely up to you. We require some basic criteria, but it's mainly dictating elements and behaviour.
- This is not a design assignment. You are expected to show common sense and critical thinking when it comes to basic user experience and visual layout, but you are not required to be creative to achieve full marks.
- Your web app must be a single page app. This means that there is only one initial browser load of content on one html page, and all subsequent dynamic changes to the page are based on Javascript DOM manipulation, and not through any page refreshes. If you do not build a single page app (e.g. using links to multiple HTML pages), you will receive a 50% penalty of your mark.

What is non-SPA? Non-SPA is a multi-page application where each new page is loaded from the server, causing a full page reload with each navigation. This contrasts with SPAs, which handle navigation on the client side for faster interactions.

4.5. Allowed Usages

In this assignment, you are allowed to:

- Add static HTML/CSS to the stub website provided (i.e. you can put raw HTML/CSS as if it's a static page, even if you then later manipulate it with JavaScript).
- Build HTML elements and add CSS properties to the DOM via JavaScript.
- Use innerText properties/functions

4.6. Prohibited Usages

- You are not allowed to have more than 1 HTML file in your repo.
- You are strictly **not** allowed to use the async and await syntax in this assignment. You must use Javascript Promises. The use of any async or await will result in a 50% penalty of your mark.
- You are prohibited from using any string-to-DOM parser (e.g. DOMParser, or the innerHTML property, or insertAdjacentHTML or anything similar). The use of any of this will result in a 50% penalty of your mark. You can read more about

this https://www.dhairyashah.dev/posts/why-innerhtml-is-a-bad-idea-and-how-to-avoid-it/.

5. Marking Criteria

Your assignment will be hand-marked by tutor(s) in the course according to the criteria below.

Please note: When we test your UI we will use a pre-loaded database JSON that already has threads and users and watches added to it.

5.1. Compliance to task requirements (70%)

- Each milestone specified a particular % of overall assignment (summing up to 70%). Implement those components as required to receive the marks.
- You **MUST** update the progress.csv file in the root folder of this repository as you complete things partially or fully. The valid values are "NO", "PARTIAL", and "YES". Updating this is necessary so that your tutor knows what to focus on and what to avoid - giving them the best understanding of your work and provide you with marks you have earned. Failure to correctly fill in this file will result in a 5% penalty.

5.2. Mobile Responsiveness (15%)

- Your application is usable for desktop sizes generally, tablet sizes generally, and mobile sizes generally (down to 400px wide, 700px high).

5.3. Code Style (10%)

- Your code is clean, well commented, with well-named variables, and well laid out as highlighted in the course style guide.
- Code follows common patterns that have been discussed in lectures and as highlighted in the course style guide.
- If you do not complete at least 50% of the assignment, your code quality mark will be scaled down to some degree based on the limited contributions.

5.4. Usability & Accessibility (5%)

- Your application is usable and easy to navigate. No obvious usability issues or confusing layouts/flows.
- Your application follows standard accessibility guidelines, such as use of alt tags, and colours that aren't inaccessible.
- Describe any attempts you've made to improve the usability/accessibility in <code>usability.md</code>

5.5. Bonus Marks (5%)

- An extra 5% of the assignment can be attained via bonus marks, meaning a maximum mark of 105/100. Any bonus marks that extend your ass2 mark above 100% will bleed into other assignment marks, but cannot contribute outside of the 75% of the course that is allocated for assignment marks
- Your bonus feature(s) can be anything. You just have to think of something that could make your web app stand out in some minor or major way. Simple examples would include just making sure that your user interface and user experience stands out amongst other students, maybe through some user testing. They can be functional (added behaviour) or aesthetic (making things very pretty).
- You could also add extra features, such as some additional frontend form validations - the possibilities are limitless.
- If you do implement a bonus feature, describe the feature and its details in bonus.md in the root directory of this repository.