yaMUN

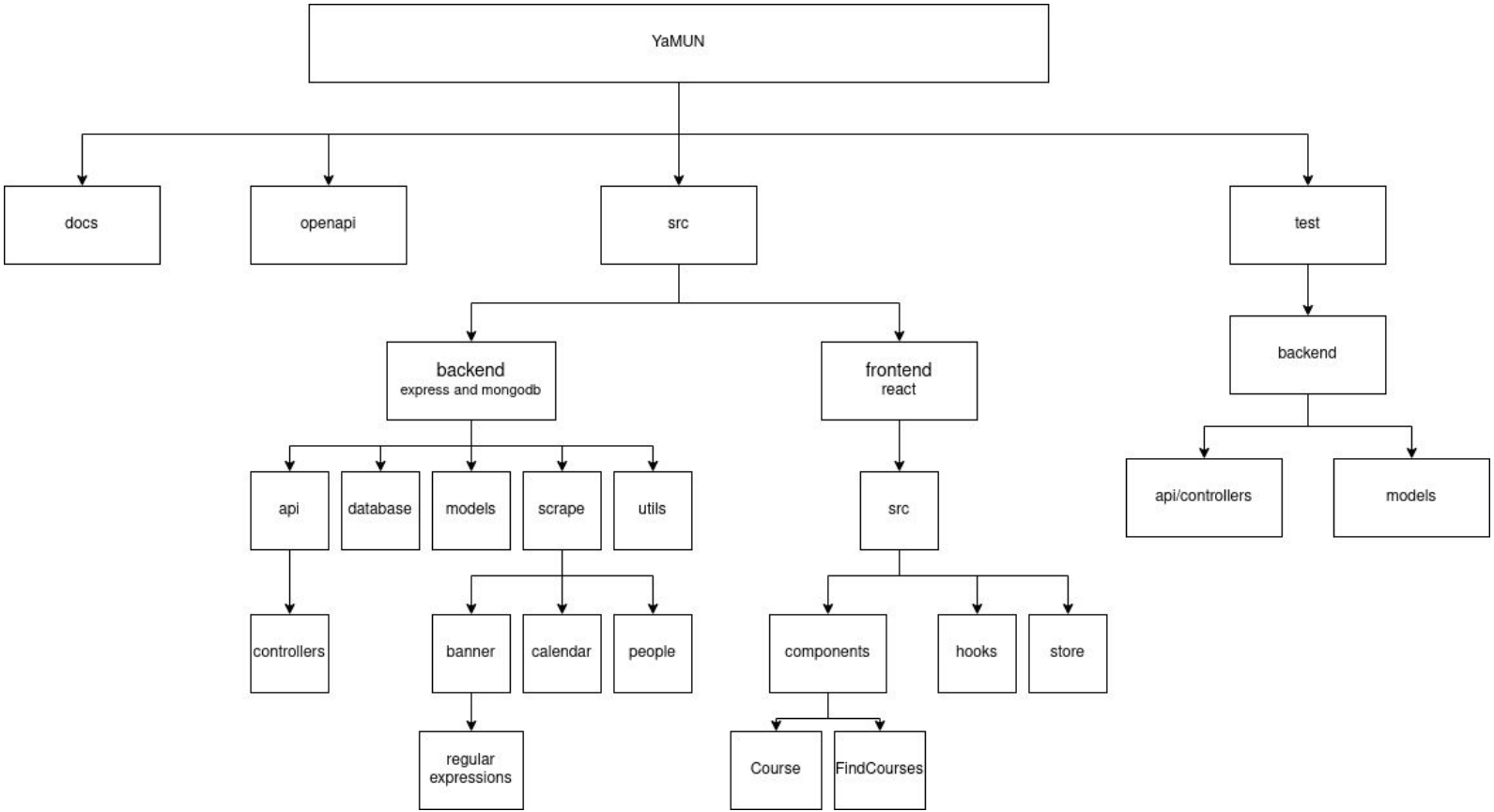
a refined interface to courses

yaMUN

- Reasoning/goals behind project
- Tooling
  - TypeScript, type safety
  - Prettier, code style
  - Vite, local hot reloading development server
- Backend
  - File structure
  - Mongoose, object modeling for mongodb
  - REST design, proper status codes, informed errors
  - Scraping different sources of data, jsdom, regular expressions
- Frontend
  - File structure
  - React, component structure, single page application design
  - Redux, state store and actions, wiring connecting to backend api
- Application in action!

yaMUN

Project Structure



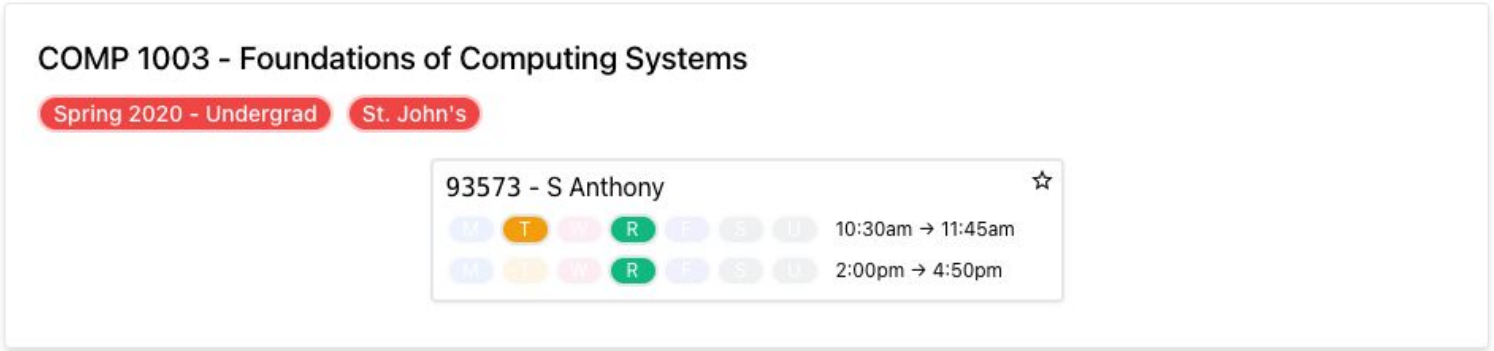
yaMUN

yaMUN was created to improve the MUN course selection and registration experience for all students.

How is it better than MUN's existing system?

- More **advanced searching and filtering**.
- Pleasant interface. Choose courses on yaMUN, and register using the CRNs.
- **Uninterrupted flow.** You will no longer need to constantly press the back button in your browser to select different subjects.
- **Bookmark system.** Not completely convinced that you want to do a course? Having a hard time choosing between multiple courses? Bookmark them and take your time to think! You can always come back and decide later.
- **Multiple schedules.** You can construct more than one schedule for a semester and see which one fits your better.
- **Schedule exporting.** You can now download your schedule and import it directly into Google Calendar or iCloud Calendar.

yaMUN



vs

Campus: St. John's

Session: Full Term

COURSE	SEC	CRN	SLOT	*** DAYS *** M	T	W	R	F	S	U	BEG TIME	END TIME	ROOM	SCHED TYPE	ASSOC LAB/ LEC SECT	PHON	WAIT LIST	PRE CHK	RESV LFTD	ATTR	CRED HR	BILL HR	INSTRUCTOR
Subject: Computer Science																							
COMP 1003 Foundations of Computing Sy	001	93573			T		R				1030	1145		L&L		Y	Y	Y		Q	3	3	Primary - S Anthony
							R				1400	1650											
Delivered remotely.																							

Tooling

https://www.typescriptlang.org

TypeScript

- **Strict typing.** Less room for mistakes and stricter data validation.
- **Easier to read.** Easy to read and understand existing code.
- **Easier to refactor.** Compiler catches errors faster than humans.
- Compiles into **regular Javascript**.

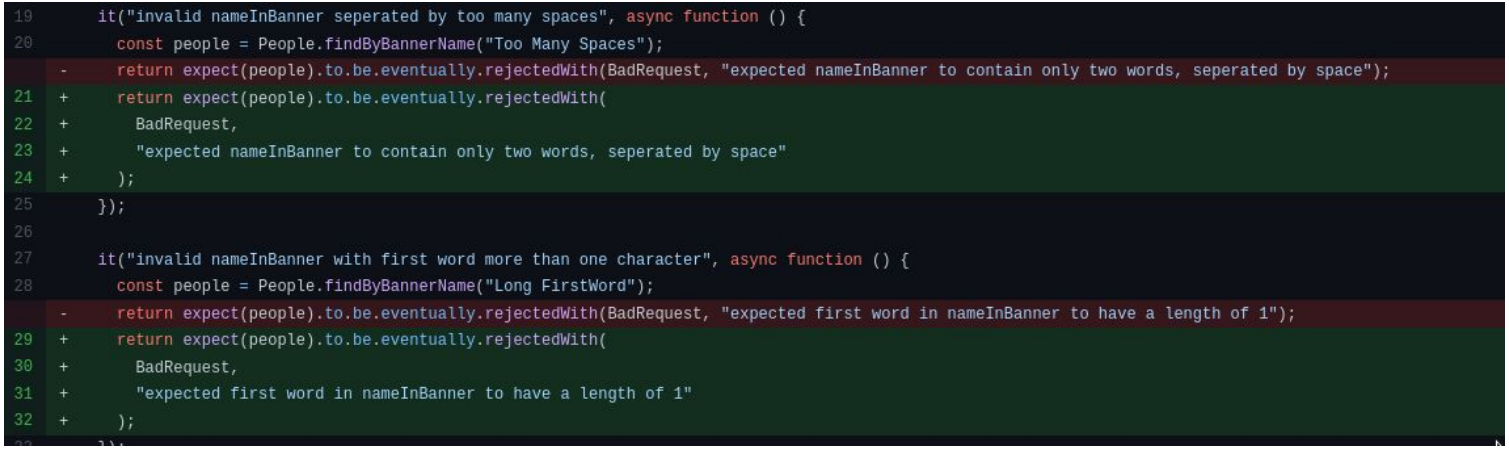
```
export interface IBookmarkDocument extends Document {
  owner: IUserDocument["_id"];
  courses: string[];
  resolvedCourses: ICourseDocument[][];
  addCourse: (sid: string) => Promise<void>;
  removeCourse: (sid: string) => Promise<void>;
}

export interface IBookmarkModel extends Model<IBookmarkDocument> {
  findByUserId: (userId: Types.ObjectId) => Promise<IBookmarkDocument | null>;
  findOrCreateByUserId: (userId: Types.ObjectId) => Promise<IBookmarkDocument>;
}
```

https://github.com/prettier/prettier

Prettier

"... an opinionated code formatter. It enforces a consistent style by parsing your code and re-printing it with its own rules that take the maximum line length into account, wrapping code when necessary." (taken from https://github.com/prettier/prettier)

- **Consistent code style,** even when working in teams**.**
- **Easy to use.** Can be used directly from an IDE or as part of a CI environment.

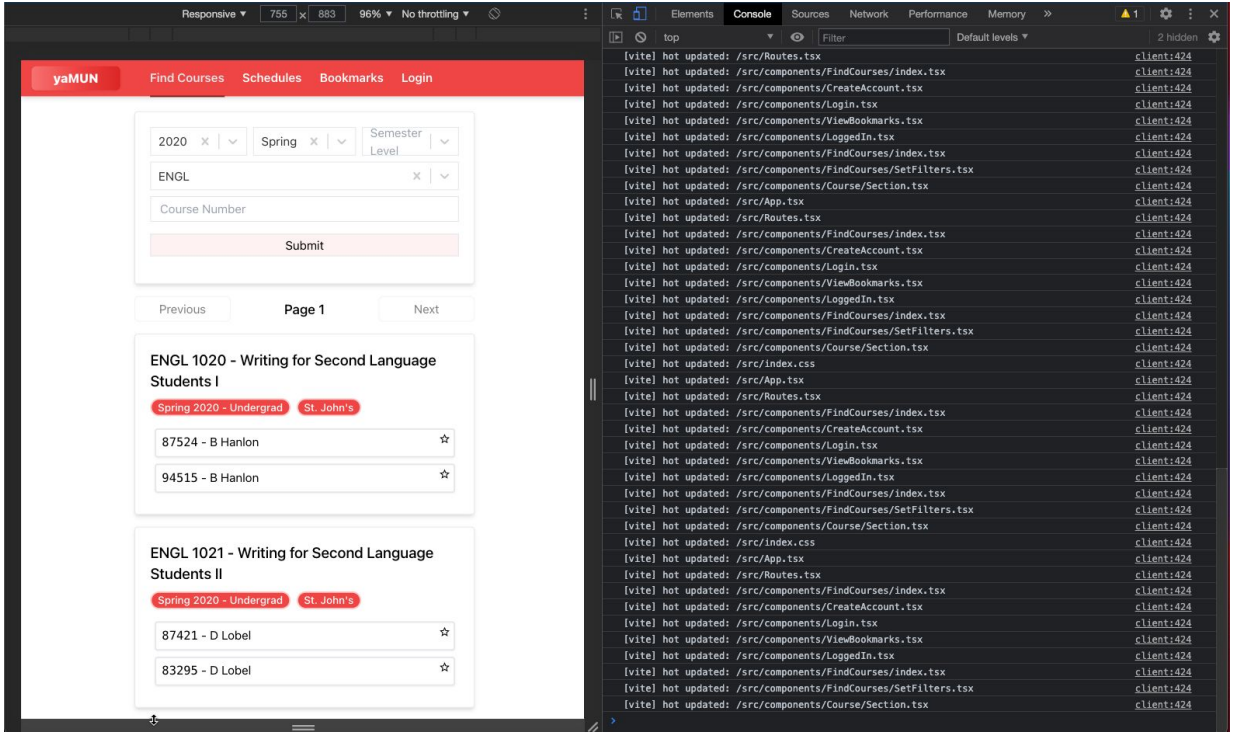


```
19      it("invalid nameInBanner seperated by too many spaces", async function () {
20        const people = People.findByBannerName("Too Many Spaces");
-         return expect(people).to.be.eventually.rejectedWith(BadRequest, "expected nameInBanner to contain only two words, seperated by space");
21  +     return expect(people).to.be.eventually.rejectedWith(
22  +       BadRequest,
23  +       "expected nameInBanner to contain only two words, seperated by space"
24  +     );
25      });
26
27      it("invalid nameInBanner with first word more than one character", async function () {
28        const people = People.findByBannerName("Long FirstWord");
-         return expect(people).to.be.eventually.rejectedWith(BadRequest, "expected first word in nameInBanner to have a length of 1");
29  +     return expect(people).to.be.eventually.rejectedWith(
30  +       BadRequest,
31  +       "expected first word in nameInBanner to have a length of 1"
32  +     );
```

https://vitejs.dev/

Vite

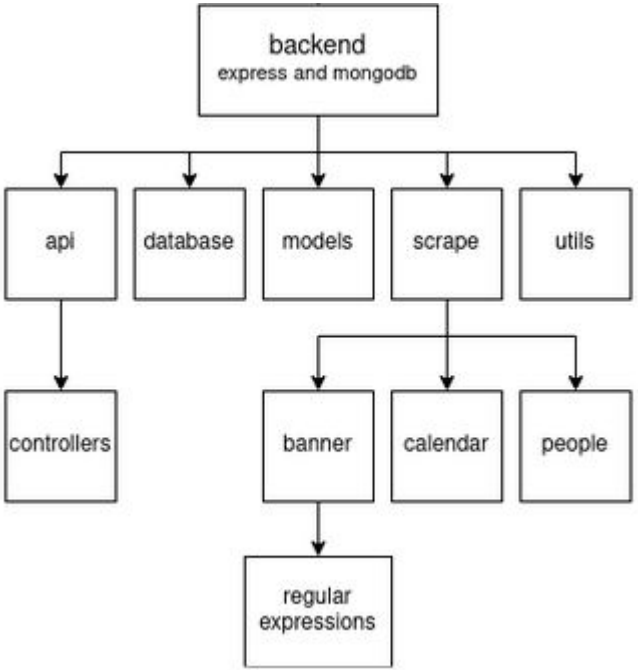
- **Local development server.** Quickly test your application.
- **Hot-reload.** Always the latest version of the code without manually restarting.



Backend

yaMUN

Backend File Structure



https://mongoosejs.com/

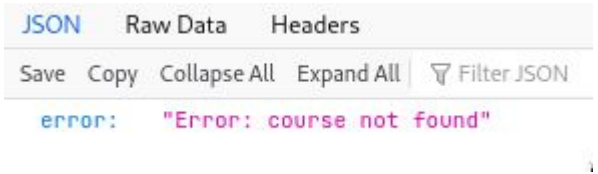
Mongoose

- **Object modeling for MongoDB.** Model objects for our data.
- **Schemas.** Stricter structure control for our Models.
- **Similar interface.** The mongoose API is very similar to the API of the node.js MongoDB driver.

```
export interface IUserModel extends Model<IUserDocument> {
  createUser(username: string, password: string): Promise<IUserDocument>;
  login(username: string, password: string): Promise<string | null>;
}

export const UserSchema = new Schema<IUserDocument>({
  username: { type: String, required: true, unique: true },
  passwordHash: { type: String, required: true },
});
const User = mongoose.model<IUserDocument, IUserModel>("User", UserSchema);

export default User;
```

REST Design

- **Single path for object.** Utilizes different HTTP request methods for actions.
- **Uses JSON.** Both for requests and responses.
- **Proper status codes.** Returns proper HTTP codes that let the client know exactly what is going on.
- **Informed errors.** If a request fails to produce something, the client will know exactly why.
- **Stateless API.** Every request contains all the data that is needed to handle it.



yaMUN

Scraping

The data we use is not offered through a public API, so we have to scrape it manually. Data we scrape: Banner course offerings, MUN Calendar course information, people & staff information.

Tools we use for scraping:

- **JSDom -** to parse the contents of the HTML pages we scrape and turn them into JS objects we can work with.
- **Regular Expressions -** to extract data from the objects we scrape.

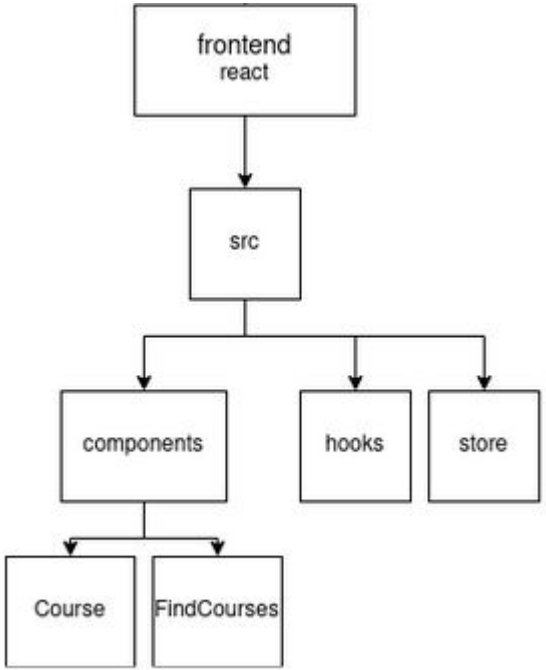
```
const dom = new JSDOM(data);

const divs = Array.from(
  dom.window.document.querySelectorAll(".course")
) as HTMLDivElement[];
```

```
export const COURSE_REGEX = /^(?<subject>([A-Z]|&|\.| ){4}) (?<number>(\d|[A-Z]){4}) (?<name>.{27})/;
```

Frontend

yaMUN

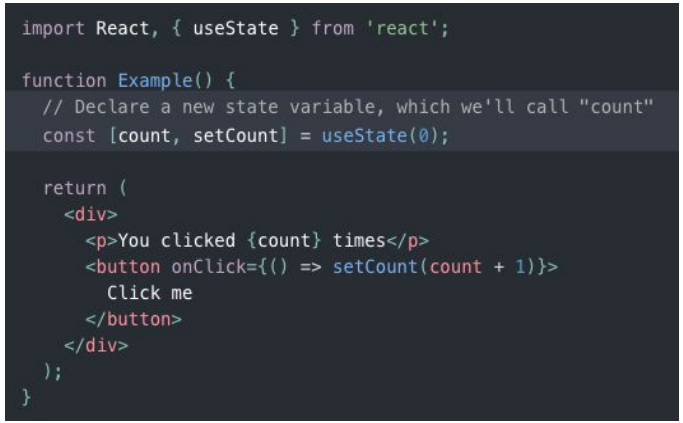
Frontend File Structure



https://reactjs.org/

React

- **Component Design.** React lets you break down your application into small chunks of logic called ‘Components’, which can be as simple as a function with arguments.
- **Simplifies Rendering Logic.** With a library like jQuery, you must handle how rendering and updating your frontend works; if you say render an array of objects, if you modify one of those objects, you must also figure out how to either patch the DOM manually, or just opt. to re-render everything, which is unperformant.
- **Code Reuse.** Within our application, the Course component is used in almost every screen, and since we’re using the above mentioned component design, using it is as simple as importing the component, handing it an array of courses, and placing where in the structure we expect the DOM elements to show up.



```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

yaMUN

```
function Course({ course }: { course: ICourseDocument }) {
  const name = course.info?.title ?? course.name;
  const semester = formatSemester(course.semester);

  return (
    <Box className="p-5 mb-4">
      <p className="text-xl font-medium mb-2">
        {course.subject} {course.number} - {name}
      </p>
      <Pill style={{ marginRight: "0.5rem" }} text={semester} />
      <Pill text={course.campus} />
      <div className="sections mt-4 mb-2">
        {course.sections.map((section) => (
          <Section key={section._id} section={section} />
        ))}
      </div>
    </Box>
  );
}

export default Course;
```

```
<Route path="/(schedules|create-schedule)">
  <ErrorBoundary>
    {loggedIn ? (
      <>
        <Route exact path="/schedules">
          <Schedules />
        </Route>
        <Route
          path="/schedules/:scheduleId"
          children={<ScheduleView />}
        />
        <Route exact path="/create-schedule">
          <Create />
        </Route>
      </>
    ) : (
      <>
        <DisplayError error="Login required!" />
        <Login />
      </>
    )}
  </ErrorBoundary>
</Route>
```

https://tailwindcss.com/

Tailwind

- **Quick CSS development.** A framework with a lot pre-defined CSS classes that can be used to quickly add style to your website.
- **Simple and extensive API.** A lot of utility that is very simple to access.
- **Only bundles what's used.** Even though Tailwind provides a lot of CSS classes, only those that are actually used are bundled for production.

```
return (
  <div className="m-auto w-auto pt-2">
    <Box className="pt-4 pb-5 px-5">
      <p className="text-xl text-center">Logged in as '{username}'</p>
      <input
        className="w-full py-0.5 mt-4 border bg-red-50 hover:bg-red-100 focus:outline-none focus:ring-2 focus:ring-red-200"
        type="submit"
        value="Logout"
        onClick={attemptLogout}
      />
    </Box>
  </div>
);
```

https://redux.js.org/

Redux

- **State Container:** The entire state of the application, is stored in a global object that every component can subscribe to, and once these properties change, only the components that need to know the updates will re-render
- **Separation of Concerns for State Management.** The components are not too aware of how to modify the state, only that they can invoke actions and receive the modified state; all of the actual logic to invoke the API and modify the state from the API response is defined outside of UI.

```
export interface IPeopleFields {
  people?: IPeopleDocument[];
  setPeople: Action<IStore, IPeopleDocument[] | undefined>;
  searchPeople: Thunk<IStore, { nameInBanner: string }>;
}

export const peopleFields: IPeopleFields = {
  people: undefined,
  setPeople: action((state, people) => {
    state.people = people;
  }),
  searchPeople: thunk(
    async (actions, { nameInBanner }) => {
      const params = new URLSearchParams({ nameInBanner });
      const resp = await api.peopleSearch(params);
      const json = await resp.json();

      if (resp.ok) {
        actions.setPeople(json as IPeopleDocument[]);
      } else {
        toast.error((json as ErrorResponse).error);
        actions.setPeople(undefined);
      }
    }
  ),
};
```

How would MUN use this?

- Likely exposing more data as APIs / json, such that no manual scraping has to occur
- Maybe removing quite a bit of the concept of a user within the application and moving it to something baked into whatever my.mun.ca uses
- Other unknown considerations that would require a rewrite of how the application functions

yaMUN

Time to show off!