Planning and Designing

I want to use a framework because it makes the app easy to scale when we have events or complicated relationships. I have used Vue.js in the past. So I thought using v-for was suitable for this. I also considered using Svelte but decided to stick to Vue because of time.

This sort of interface reminds me of eventbrite.ca. My first thought was to use bootstrap cards for each event. That should make things easy. But recently, I watched a video from fireship talking about Tailwind CSS, and I have always wanted to try it out. So I went to their website, played around, and ended up managing all the styles in my UI with Tailwind. It was so easy and intuitive to use. I am happy I made this decision.

I wanted to try some other new things with this project. So I decided to use Vite as a build tool and use GraphQL to get data. I'm glad building this project finally pushed me to learn these things.

Implementing

My first goal was to implement all the UI components and styles with mock data. So I hit the GraphQL API with an external client and copied some of the data to an events array, so I could work on the front end as if I got the data from hitting the endpoint. In the design phase, I already decided what the website should look like, so I modified some Tailwind styles and made the data display properly. I also made it responsive easily with the help of Tailwind.

It went smoothly, so I was ready to connect to GraphQL and get real data from the backend. However, integrating the Apollo client into my app got me stuck: I could not get data from useQuery. So I opened the inspector to check network calls: the call has already returned data successfully. From that, I know something is wrong with my usage of apollo. This could be caused by asynchronous behaviour, so I added extra checks for loading status and error. However, it still didn't work. Ultimately, I discovered I was following the procedure for an older version: when I decompose the response, the data

is in “result” instead of “data.” I could have saved lots of time if I had read the documentation from the beginning instead of rushing through.

Then I just had to implement the details like sorting and filtering. I thought about using a computed property for visible events. I also wanted to maintain a map of id and events to look up details for a related event efficiently. Otherwise, I need to call the API again or do a linear scan of all the events every time. It wasn’t working, although the idea is straightforward, so I filtered events in the parent and implemented a find helper on the same event list instead. Toward the end, I realized I wasn't defining Component data properly. I have used Vue Options API before, but this time, I decided to try Composition API, and I assumed data() is equivalent to simple variables in the Composition API. Again, I could have saved so much time if I had read the documentation carefully.

Future Work

- Separate the <article /> code for each event into a separate <EventLlist> component.
- Add a search bar.
- Add dark mode.
- Mode filtering/sorting options. Filter by tags.
- Complete authentication. Separate login page. Use OAuth too.
- For the GraphQL endpoint, consider returning names and links for related events. Alternatively, maintain a map in the front end to look up an event in constant time.

Trying it out

You can try it directly at https://keen-dieffenbachia-96044a.netlify.app/. For more information, see the README.md file in the project.