Hack the North 2022 Frontend Developer Challenge Writeup

Jennifer Lu

GitHub: https://github.com/jennifer-lu/htn-dev-chl
Figma: https://www.figma.com/file/K4mOPjIYXukgQQ5AgtRi83/htn-dev-chl?node-id=0%3A1

Question 1

Walk us through your development process as you worked on this project. How did you plan out the structure and design of it? How did you decide on the tools you've used? Did you encounter any problems? And if so, how did you solve them? Are there any areas of your code that you're particularly proud of or want to point out?

Workflow:

1. Design: made wireframes for the user interface in Figma
2. Tools: decided on which languages and libraries to use
3. Development: built a basic working app with the main functionalities, then added styling and additional features, finished by cleaning up code
4. Documentation: created a readme and this writeup

Step 1: Design

My first step in this project was brainstorming layouts for the web app. I chose a card-based layout where each card contains all of the information for an event. This way, users can easily get an overview of the events and also directly access the details of each event. I decided to limit the size of the cards so that many events can fit on the screen at the same time. However, I still made sure that the cards were large enough for the name, event type, start time, end time, and url to be visible without scrolling. I also emphasized these important details by changing the font size, font weight, and italics. Above the cards, I added a floating navigation bar to allow for easy access to functions like login, logout, and search. To match the name “Hackathon Global”, I used a graphic of Earth from space as the web app’s background. I created the background graphic in a flat style to complement the clean design of the web app. I also picked a color palette to use throughout the web app for consistency. Finally, I made a wireframe in Figma based on these design choices.

Step 2: Tools

I chose React and TypeScript as the frontend JavaScript libraries for this project because I have the most experience with them and they were mentioned in the frontend developer position description. I used Chakra UI for styling because it’s a component library, and thus allows for faster development since I don’t need to worry about building out and styling basic components like forms, modals, and buttons. I picked GraphQl and Apollo Client as my query

language and API client library because the necessary GraphQL queries were provided in the challenge document, which made the remaining setup very simple. I also have more recent experience with GraphQL and Apollo than REST. I used ESLint and Prettier for linting to keep my code consistent and ensure I followed best practices. For the search feature, I used Fuse.js because it's an easy-to-use library that allows for fuzzy search.

Step 3: Development
I began development by building out a basic app. I started with the boilerplate code from create-react-app, and added the rest of my tools and dependencies. I implemented major functionalities one by one, going in order based on the requirements listed in the challenge document. I made sure to set up a scalable directory structure, with separate folders for categories like assets, api, components, contexts, and styling. I also broke everything down into modular components. I used states to keep track of things like inputs, button presses, and query results. I used contexts to control more global information like authentication, color mode, and device type. To reduce duplicate styling code and maintain a consistent style, I implemented extended themes with Chakra UI. I also added alt texts and aria labels for better accessibility. I ensured that my web app was responsive by using responsive components like flex boxes and also built out features to accommodate mobile users like a hamburger menu. To ensure best practices and code maintainability I used the linters ESLint and Prettier, and also made sure to refactor and document my code. I thought carefully about my implementation of each feature and kept my code as minimal and straightforward as possible.

Step 4: Documentation
I added a readme with some external documentation about how to get started, what the dependencies are, and a basic overview of the code architecture.

Problems:
In my initial designs, I didn’t consider a separate layout for mobile devices. When testing my website for responsiveness, I quickly realized that this was an issue. The two main problems were that the navigation bar was too crowded and the event cards were too large for the screen. In order to reduce the space taken up by components in the navigation bar, I moved all of them into a hamburger menu. To fix the issue with the event card sizing, I tried to make them shorter to conserve vertical screen space (and display more cards at the same time). Unfortunately, this hid a lot of the information in the cards. Users would have to scroll within the event cards to see any details. A related issue is that nested scrollbars are not friendly on mobile devices. I resolved this problem by removing the scrollbars and adding an expansion toggle. By clicking an expand button, the user could expand a card to view the event details. To control when the mobile layout is shown versus when the desktop layout is shown, I added a device context that is set based on the screen size. To get the screen size, I used a media query (from react-responsive) to check the viewer width.

Another problem I encountered was how I would implement search functionality. I struggled to decide on a way to display search results (i.e. the events that matched the search keywords). At first, I thought of displaying the resulting event names as menu items in a drop-down menu under the search bar. But then, a user wouldn't be able to easily view the event details. Also, adding the event details to the drop-down menu would make the menu far too long and crowded. I eventually decided that the search bar should filter the currently displayed event cards. That way, users could easily see the event details for all of the search results. Unfortunately, this meant that users could not return to viewing all of the events (since they were replaced by the search results). Thus, I added a clear button to the search bar, which would clear the search input and restore all of the previously filtered event cards.

Question 2

Given additional time, how would you extend your application to become a fully functional product that thousands of hackers and the general public would use at Hackathon Global Inc.™'s next event? Would you add more features and performance metrics? If so, what would they be?

Given additional time, I would add features such as reservation, calendar view, filters, and token authentication. The reservation feature would allow users to RSVP for events with attendee limits, and also provide headcount data to help organizers plan for the size of an event. Implementing this feature would require a table where event ids are mapped to event headcounts. A GraphQL mutation could be used to update the table when a user registers and unregisters for an event. Mapping event ids to an array of registrants might be an even better solution, since it would make it easier to check that a user has not registered twice in the same event, or for two events that overlap in terms of time. Organizers could also use this data to send reminders and follow-up emails to registrants.

I would also add a weekly calendar layout as an alternative to the existing card layout. The layouts could be toggled using a button in the navigation bar. In the calendar view, events would be displayed on a calendar based on their start times and end times. This would allow for users to easily plan their day and schedule events (especially in tandem with the reservation feature, since they would be able to easily see what times they are free). This could be implemented by adding an appropriately styled calendar view component.

Another useful feature would be filters. Filters would complement the existing search feature by allowing users to filter on specific parameters like event type, date, and speakers. The filters could be implemented as drop down menus with multi-selectable menu items.

With access to the backend, I would also add authentication with JWT and OAuth 2, since my current implementation just involves saving a boolean in local storage.

Question 3
Any other thoughts you have (not limited to the previous questions).

In addition to the required functionalities, I implemented a search feature and color modes. For the search feature, I added a search field to the navigation bar so that users can filter events based on the input. When keywords are entered in the search bar, the event cards are filtered and reordered based on relevance. Additionally, I weighed search parameters like name and description heavier than parameters like start time and end time in an effort to improve the relevance of the search results. To accomplish this I used Fuse.js, which is a library that provides fuzzy search functionality.

In my implementation of color modes, I created two Chakra UI extended themes to control the web app's global color styles and color mode configuration, one for light mode and one for dark mode. I also made light mode graphics (initially, I only had dark mode graphics). To enforce the color mode across all app components, I implemented a color mode context that can be toggled by a button in the navigation bar.