Ethan Keshishian

Paul Eggert

CS97

20 December 2000

Pollify

1. Introduction/Purpose:

My group worked on a project called Pollify. Pollify is a web app that allows users to vote on a new poll every day, comment on those polls, and vote on the following day's poll. It also allows a user to view all previous polls, their participation in them, and demographic data about the people who voted. The purpose we set out to achieve when we first came up with the idea was to create a fun and engaging platform that would also be a tool for those seeking to study correlations between users' identities and their opinions.

2. Architecture and technologies:

We used React to build our frontend UI, and AWS Amplify in the backend, using Cognito to handle authentication. We used DynamoDB as our database, which allowed us to store all user votes, comments, and more. There were several technologies we used in our project, for many reasons, detailed below:

- TypeScript – TypeScript has many benefits over JavaScript. Its main feature allowed us to add types to our variables, which prevented many type errors. In addition, it allowed us to easily manage props and helped our team quickly understand each other's code.
- Redux – Like TypeScript, there are many benefits to Redux. The main benefit to Redux is a single app-wide state, which was useful for our project because we had many components that relied on the same data.
- Thunk – Thunk was used to allow asynchronous database calls with Redux. This was helpful in connecting our database to our frontend, but did pose a couple challenges, detailed in the improvements section.
- react-calendar – This component was used to display the calendar on the account page. It was useful because it was very customizable and gave us an interactive method of viewing previous polls.
- styled-components – This was a useful component when working with complex CSS. In particular, for stylizing the demographics and the poll button animations, styled-components allowed the use of incorporating JS variables in the CSS, so things like

the animation playback could easily be paused and the locations of the colors could match the poll data.

3. **Features:**
    - The ability to vote on a poll every day that automatically resets at midnight.
    - Users can leave comments and engage with others with other users' comments.
    - Users can suggest and vote on the next day's poll. At midnight, the poll ends, and the poll with the most votes becomes the next day's poll.
    - Allow users to view the results of previous polls through an interactive interface. Queries are easily made by selecting the day.
    - Display demographic data about users and their votes. This is done through login with Google OAuth and some additional information that our users provide voluntarily when they sign up.
4. **My contributions:**

I acted mostly as a frontend developer, while also concerning myself with some full-stack development. My first contribution to the source code was initializing the repository; I created the repository using 'create-react-app' and initialized it with TypeScript. TypeScript was helpful for our team, and I believe the benefits of TypeScript became more evident as our project grew. Our components eventually came to have many props with many different types, some of which were sent by the backend and were a mystery until displayed. Additionally, group members had to be aware of the changes made by other developers. TypeScript helped developers quickly understand the functionality of components and this in turn saved a lot of time and confusion. After adding my groupmates to the repository, I set up a Git workflow where developers would make changes on a branch named "develop", and these changes would periodically be merged with the master branch. To ensure the workflow was respected, I added some branch protection rules which disabled pushing to master and required pull requests and a pull request review. This helped us manage our workflow as well as adding a component of code review to our project. Another task I had related to setting up the repository was adding a deploy key from AWS so AWS could manage deploying what was on the master branch. I also made most of the updates to our readme file to describe how to set up the repository.

My first task after setting up the repository was scaffolding for the homepage. I think my first commit helped set an example of what our project would look like as it grew; I demonstrated an example of TypeScript being used and showed how to split up the project into components. All our components were functional components too, as React moves away from class-based components. I then implemented the navbar at the top of the page and later contributed to the routing to the three pages. One of the components I worked on extensively was the main poll button and its animations. When logged out, animations were paused, and once logged in, animations were determined by the state. The first type of animations starts at the left side of the button, and the second type starts from the previous state of the poll after the results

are updated. I talk more about this in the challenges section. Another component I worked on was the comment section. I had an interesting challenge here, detailed in the challenges section.

When the designs for the other pages were finalized, I built out the UI for those pages as well. When working on the UI for the account page, I benefitted from being able to reuse components from the poll and modified them to function differently if they were the current day's poll or old polls. I imported react-calendar and used it for implementing the calendar on the left side of the page. To make it work, it calls a function that fetches the results of a particular day when a day is selected. It also prevents users from selecting the current day or future days, and defaults to the previous day. Later, I worked on adding some style to the demographics section of the page, which includes a component to show the percentages of votes by gender, which again uses styled-components to dynamically update the location of the colors. After building the about page, I wrote up the text that describes Pollify.

5. Difficulties/challenges (personal and team):

- One challenge I faced directly was implementing the separator for the comments in the comment section. I noticed that when using a standard HTML
 tag, the separator would disappear when changing the zoom on the page because of how the browser calculates pixel size. I tried some other solutions that had similar problems, until figuring out that using a canvas would allow a 1-pixel size that displayed regardless of the zoom.
- I faced another challenge when implementing the animations on the poll buttons. I spent a lot of time trying to animate the buttons, trying out many different solutions. The solution that worked was using styled-components to dynamically update the CSS with variables determined by the poll data. These variables and the state of the web app were able to determine when an animation should run and where the animation should start/end.
- One challenge was getting our calendar component to work with the same time zone as our backend. We noticed this issue while testing and made some changes to our local component and the backend. Once this was fixed, I discovered another problem with our days being out of sync, which we eventually fixed.
- In general, connecting our frontend and backend was a challenge. Sometimes, features in the frontend were ready but didn't function because the backend wasn't ready, and vice versa. One nice way of solving this was scheduling sessions where we would work with a live call and simultaneously build a feature. VSCode LiveShare was helpful for this since we could all make edits on the same document.

6. Improvements:

- One update I want to make is to fetch information from the database periodically. This would serve two purposes. The first would allow for the poll results to constantly be up-to-date, and would result in a live animation on the frontend. This would be very engaging for a user. The second reason is because it allows for the

animations to be smoother. Currently, our use of Thunk allows us to use Redux asynchronously, but this can result in hiccups in the animation when poll information isn't fetched fast enough. There are also other benefits to this. I think this change would make our app more engaging and have it look even more polished.

- I would like to add some polish to the demographics section and make a slight change to our color scheme to be more accessible. The light pink color can be hard to read and the demographics section can be improved to make the information easily digestible at first glance. We could also add a lot of functionality to the demographics section, which is something we initially set as a dream goal. We would like to be able to present more nuanced information about the polls.
- This is small, but I'd personally enjoy doing some code cleanup and reorganizing some code. I'd especially like to group components with their stylesheets. I like the current structure for smaller projects, but it becomes more tedious with larger projects.
- Lastly, one idea we initially had with our project was to convert it into an extension or even a homepage for our users. This would enable users easier access to the app and could help make it a routine for our users to vote every day.

7. Conclusion

I really enjoyed working on Pollify. I'm thankful for the opportunity to work on it and look forward to seeing where it goes.