whatsRoaring

Project Evaluation

Through working on this project, we learned a lot about how to design a web app, divide and share tasks efficiently, and prioritize some functionalities over others. We also developed a much better sense of how long certain tasks take, which will allow us to plan similar projects more easily in the future.

Evaluation of Our Original Planning

Our original planning was not ideal. None of us had ever contributed to a similar project before, so we did not have a good sense of how long everything would take. Our initial plans were very eager, with ambitions like obtaining data from Facebook and making event suggestions. However, it turned out that just making an app with basic functionality was a lot more difficult than we had expected.

As a result of our overly ambitious planning, we got a little behind in our milestones. We fell behind in the beginning because it took a long time to familiarize ourselves with the frameworks, since none of us had used Django or React before. It was also difficult because the project started ramping up right when we were busy with midterms and we spent a lot of time trying to figure out how to deploy Django and React together on Heroku. The fact that we spent about a month working on CAS authentication also caused a big delay, since we could not implement favoriting or different permissions for admins without that. Our milestones said that we aimed to finish CAS authentication on November 21, but we did not get any form of authentication working until the start of reading period. Without CAS, we also could not achieve many of our stretch goals that required authentication like email/text notifications and event suggestions. Filtering and interface design also took longer than we originally expected.

Evaluation of Our Design Decisions

We managed our database with PostgreSQL, which worked nicely because it is well integrated with Python and Django. We were able to access and make changes to the

database when running our app locally or in Heroku, which allowed us to add and edit objects in our database before we added this functionality to the app.

Django, which we used for the back-end, had a learning curve. However, the model abstraction was very helpful because it enabled us to edit the database without accessing it directly. It was therefore much easier to add tables (or models) and columns (or model fields) without breaking our code. We took advantage of this feature on several occasions, since we realized throughout the project that we needed to save additional information. It was also very useful when we decided to create our own authentication system, since we had to build a new User model, and Django also had built-in methods for encrypting passwords.

We used React for the front-end, which was also had a learning curve, especially since most of us had not used Javascript before. However, it also offered many advantages. We chose it because it has a nice calendar user interface, which saved us a lot of time. Additionally, one of the defining features of React is that it has a hierarchical system of apps, pages, and components. Users can make components and then insert them into a page. The modularity was useful because it enabled us to change our website easily, especially when we needed to move buttons to different pages. It also made it easier to have multiple people working on the front-end simultaneously. Another convenient feature involved changing the state of a component without refreshing the application. This characteristic was useful for many aspects of our project, such as dialog boxes and button changes upon users pressing them.

That said, we found it challenging to facilitate communication between our front- and back-ends. In particular, it was difficult to learn how to use RESTful APIs and how to make GET and PUSH requests. Debugging code involving this communication was initially grueling because we did not fully understand how the system is designed.

Heroku does not easily support deploying an app that uses both React and Django, and after spending a week trying to deploy our project, we decided to create two separate apps for the front- and back-ends. We are thankful that we decided to deploy our app early in the project because we were able to split our app before we had implemented most of the functionality. Additionally, splitting the app proved helpful while many teammates were working on the app simultaneously because it enhanced modularity. In particular, it helped us avoid merge conflicts as we developed our application. Modularizing our app like this was especially helpful when two people were simultaneously working on the front-end and back-end for the same function.

Despite the positive effects derived from enhanced modularity, having two apps made it almost impossible to implement CAS authentication. As mentioned previously, this hurdle delayed our project significantly because it took a long time to realize that it was not going to work. However, making our own authentication system was helpful in that the app is now accessible to community members who are interested in events but not affiliated with the University. Furthermore, dividing our app into two caused it to rely more on React. Specifically, if we had not split our application into two, then we would have used React primarily for aesthetic purposes. However, in our case, we also had to rely on React to execute POST and GET requests.

While React and Django certainly afforded us many tools and abstractions that we found useful, we found that getting help when needed was somewhat difficult. In particular, the course staff is not as familiar with Django and React as they are with Flask and AJAX, so we had to solve many of our problems on our own. Having a large team was useful in this case, as described in Surprises.

Surprises

We certainly encountered many surprises as we were developed this application. Initially, we were startled by how long it took to become familiar with our back-end and front-end frameworks. However, as we became more acquainted with our technologies (Postgres, Django, and React), we were pleasantly surprised with how quickly we could implement different features.

A number of tasks turned out to be surprisingly difficult, including using sticky toolbars, dealing with asynchronous problems in filtering, and implementing CAS authentication. Furthermore, we revised our aesthetic design many times, and this ended up taking more time than we had originally expected. One particularly unpleasant surprise involved Twilio, which we tried to use for automatic text notifications. Specifically, our Twilio API key was accidentally published on GitHub – this unfortunately resulted in fraudulent purchases on Becky's credit card. From this, we learned to be very careful with secret keys and to be mindful of the information we publish in public repositories. Ultimately, we decided to allocate our time to other features of our application. Specifically, our users indicated that they preferred Google Calendar functionality over automatic text notifications, so we shifted our focus to that feature.

Initially, we doubted our decision to have a five-person group. Having five members with busy schedules made it difficult to schedule times to work together. However, on the flip side, having a large group enabled us to have multiple people familiar with the back-end

and front-end frameworks. Therefore, when someone was stuck on a bug or confused about how to implement a feature, it was possible to work together with teammates. Additionally, because we had a large group, we each developed our own niches within the project. Eventually, we were each so familiar with our own corners of the application that implementing new functionality seemed almost trivial. That said, dividing the project like this made communication even more paramount. As a result, working in the same physical space was especially helpful. Between our combined knowledge and ease of communication, we were able to work much more efficiently in this setting.

Changes We Would Make Next Time

If we redid this project, we would have worked harder to find times to work together throughout the semester. As explained in Surprises, working in the same physical space allowed us to work much more efficiently. In particular, during reading period, we spent days working together because we did not have full schedules. During this time, we tweaked or added almost all of our functionality. Our ability to do so was a tribute to the importance of communication, and we realized that if we had worked together in this way every week throughout the semester, we would have been able to accomplish more tasks.

We are conflicted about whether or not we would choose the same back-end and front-end frameworks if we redid our project. Although Django and React individually offer several useful features, combining them caused us to split our project into two heroku applications. This ultimately caused severe problems implementing CAS authentication. Using different frameworks might have saved us time in the long-run based on how much time we wasted trying to solve this problem. However, by the time we realized that we could not implement CAS authentication, it was too late to change our frameworks. Along the same lines, we should have moved away from CAS and implemented our own authentication system much sooner; this would have enabled us to add additional features that relied on user authentication before reading period.

Future Work

If we had more time, we would work on inputting events into our Database by screen scraping Facebook events, implementing text notifications via Twilio, and adding event suggestions for each user. Originally, we listed all of these features as stretch goals of our project, but we were not able to implement them in the allotted time frame. In particular, we did not have time to include these functions because we faced many challenges, as described previously, and we focused more on including features

suggested by our users. For instance, we concentrated on implementing an Edit Event function, as proposed by our users. Our users also suggested adding a search bar and color-coding events by category, and we plan to incorporate these features in the future.