inPerson

Final Report

Alice Gao

Michael Peng

Anna Qin

Ioana Teodorescu

FINAL REPORT

PROGRESS

TIMELINE

In retrospect, our original timeline was a bit ambitious. However, the stages of development that we laid out provided a benchmark for overall progress in the project, and scheduling our progress goals in this way provided motivation for us to keep working and accomplish our tasks in a timely manner. Because our original timeline had a significant amount of built-in buffer time, we were able to complete a viable product by the deadline.

In terms of flaws with the original planning, we did not expect many of the various setbacks that we encountered at each stage. On the back end, testing was delayed due to problems setting up a local database using Windows Subsystem for Linux and PostgreSQL. Deployment was also hampered due to difficulties configuring the Relational Database Service (RDS) instance correctly. Alpha testing for the application was delayed because we had initially separated development on the front end and the back end, and it took several attempts to integrate React with Django. We also did not anticipate the UI issues to be so abundant and difficult to solve.

However, the original planning was effective in that it provided an accurate picture of what should be prioritized as soon as a given stage was completed. In this way we were able to get the most basic working version of our app by the end of the semester.

DOCUMENTATION

While working on this project, we kept an in-depth updating documentation and progress log. This served as a unified road map, allowing us at any given time to keep track of what members were currently investigating or working on, the specifics of any issues that had been encountered, and what needed to be accomplished next, as well as miscellaneous info like reference links and necessary login information. Adding dates to the log also allowed us an easy way of tracking the speed of our development and whether we were keeping up with our timeline. In addition, we also kept notes of our weekly meetings with our TA, which helped track our progress, goals, and questions on a more macroscopic level.

WEEKLY MEETINGS

As this is a group project, we realized that scheduling group work sessions dedicated to this project would help greatly. We ultimately decided on establishing three such work sessions through the week, each ranging from 2.5 to 3.5 hours, and every member needed to attend at least two a week, although we usually ended up attending at least part of all three. These meetings helped ensure that everyone was on the same page in terms of what had been and what needed to be done, and we could also help each other make progress in certain areas if needed.

DEVELOPMENT EXPERIENCE

DESIGN AND INTERFACES

Early on in the project, we got together to discuss what we wanted our product to do and look like, and from there we made our choices about which systems to use.

By narrowing down our target audience to Princeton students, we could develop Princeton-specific features like course schedules. Given that schedules contain private information tied to individual users, it made sense to use Princeton's Central Authentication Service for our login system. This means that users don't need to create another login and password just for our system, and we can ensure that every student has only one account in our database.

For the calendar itself, we drew inspiration from the commonly-used calendar apps like iCal and Google Calendar, as well as from the Princeton course scheduling apps that have calendars built in, like Recal. As such, we chose to use MaterialUI because we were already using React, and because the framework has clean design and abundant documentation. After we tried using the DevExpress library for the calendar, we discovered that the library was not fully implemented, so we switched to using the Scheduler widget from DevExtreme. This widget allowed us to easily display multiple people's schedules in different colors. However, when a user chooses to display someone else's schedule, the calendar must be refreshed (which may be accomplished by navigating to a different week and back, or switching the calendar view) so that it may be displayed. Additionally, adding a custom event to the calendar requires a page refresh to display correctly.

There were a number of tradeoffs regarding the features we wished to incorporate. One of our biggest debates was whether or not friendships should be bidirectional or one-way. In the end, we decided that it was best to have a one-way follower-followee relationship because sometimes a user may want to see someone else's schedule, but may not want that other user to see their schedule. For example, the leader of a group might not find it necessary to share their own schedule when arranging a meeting among members; only they themselves would need to see everyone's schedules. Additionally, we decided that a user should be able to remove followers in case they decide they do not want another user to follow them any more. Overall, a one-way implementation gives more control to the user.

Another design decision was our choice to store each section of a class (such as COS126 precept vs COS126 lecture) as different objects in our database. This better suited our needs to add "recurring events" to the calendar. However, one drawback is that when a user searches for their classes, COS126 lecture and its precepts pop up as different options instead of as one option like on Recal or Princetoncourses. Still, we believe that this would not greatly impede usage.

We considered adding mobile support for our app, but after talking to and observing the habits of other people, we decided to prioritize other functionality. Many people, when planning out schedules, do so on their desktop computers, simply because the larger screen makes it easier to view everything.

LANGUAGES AND SYSTEMS

For the back end, we chose Django as a framework because of the plentitude of documentation and support that it had. This proved to be crucial as we learned to use the system. Django is widely used and, thanks to the large community, there are many useful packages available, such as the django-friendship library.

Additionally, the Django REST Framework made it easier to support the REST API and API endpoints, which were crucial for the requests that we needed to send.

For the front end, we chose to use React.js largely because one of our members had significant experience with it; we knew beforehand that it met our requirements. React is good for building user interfaces with single page applications, and data can be changed without needing to reload the page. This made it suitable for the kind of app we wanted to have, which we knew would consist primarily of a view of a calendar. It is overall a fast and simple system to use.

Like many other COS333 projects, we chose to deploy our app to Heroku. Assignment 4 for this class had already given us some experience in doing so, and so we didn't encounter any significant problems in that regard.

For database hosting, we initially planned to create a Postgres Heroku database because it was the simplest option, but we found that it was of insufficient size to hold all of the class information and user information that we needed. Since we wanted this app to be used by Princeton students in the future, our app would not have been scalable if it we continued to use a Heroku Postgres addon. Therefore, we instead hosted our database on Amazon Web Services, which met our size requirements without charge, and provided useful documentation on connecting to a Heroku app. Although initially tricky, it did not take too long to correctly configure the system so that our database was successfully connected.

We used Github to collaborate on the project and version control our web app. Our workflow on the project involved checking out a new branch each time we tested or implemented a new feature. When this feature was done, or a component of it worked properly, we then merged these changes into the master branch. This allowed us to avoid conflicts in the code and enabled us to always have a working version of the web app. We also made sure to clear out branches that were completed or no longer relevant.

TESTING

Self-testing formed the majority of our testing as we developed our product, allowing us to catch numerous problems before sharing our application on a wider scale. As is the case with all such projects, implementing one new feature often caused something else to break, so it was crucial to test everything thoroughly ourselves.

In terms of the back end, we hosted our database locally and wrote many independent Django tests for each Django app to ensure that all of the database queries were functioning as expected. Most of our front end testing was completed by building the full app and manually experimenting with features to determine the correctness of their functionality.

After finalizing the functionality of our web app, we solicited feedback from our friends. Much of this took the form of UI feedback, and we took these comments into consideration when we tried to make the UI more intuitive and overall easier to use. The inclusion of the tutorial on first login was one such feature that originated from user feedback. As of now, the feedback link on our web app remains open.

WHAT WE LEARNED

As this was the first time any of us had ever worked on such an involved project, we learned a lot about how to think about design on a large scale. Our earliest meetings were about how we wanted to structure the back end and how to represent the users, schedules, events, relationships, and so on. There were many design decisions that had to be made, as described in sections above. We've learned how to think of the big picture

about the connections between all of the components of our system, and build in such a way that doesn't limit us later on and allows for scalability and changes. This was, for example, why we built the front and back ends separately instead of connecting them at the beginning, because we wanted the freedom to change if we ran into a dead end.

A large part of this project involved learning all of the technical skills necessary for the various components of this project. For almost all of the languages and technologies we used for this project, we had to learn from scratch, relying on copious amounts of Google and asking for help from other, more experienced people. In a way, from this project, we've learned how to learn.

Many surprises were encountered in the process of developing this web app, most of the unpleasant variety, as we kept running into roadblocks and errors. The most common thing was the never-ending cycle of making many small scale changes to try and fix some issue, only to break something somewhere else and have to fix that. There were a few pleasant surprises, when something was completed faster than anticipated, was much easier to implement than anticipated, or was already implemented in some convenient public library. As a whole, for example, MaterialUI was fairly straightforward to use, and the django-friendship library saved us from having to implement the entire following system from scratch.

Although somewhat unavoidable, developing the back end on Windows as opposed to a Unix-based system made the task more difficult than it otherwise might have been. It was extremely difficult to set up a Postgres database locally, and at one point the local database got corrupted and development had to continue on another machine entirely. Due to the time and cost constraints, and the desire to keep our testing environment as close as possible to the production environment, we had no other options for this project. Unfortunately, we could not simply buy a Mac to avoid using Ubuntu on Windows entirely, but in the future when a new laptop is needed, this experience will likely play a factor when choosing a replacement.

We now know the importance of carefully choosing libraries to use. For example, when deciding on what kind of scheduling library to use, we had initially chosen the DevExpress React-MaterialUI Scheduler because it was one of the first ones we found. Only later did we discover that it didn't include all of the functionality that we would like. We ended up switching over to the DevExtreme library, but this wasn't ideal either because the documentation was for jQuery when we were using React.js. If we had more time, we might have chosen something else. It's difficult to predict exactly what is needed when initially choosing what systems to use, but we now have a better sense of the kinds of considerations to take into account when making our choices - documentation, support, functionality, etc. Documentation, in particular, is extremely important in learning how to use a system; a system with many functionalities is of no use if the functions are too difficult to learn. If we redid this project, we would definitely be more careful with our preliminary research.

One unfortunate fact we ran into was that the domain "inperson.herokuapp.com" was already taken by another heroku project. By that point, we had been calling our app inPerson for too long to change the name, and so we could only choose "nperson.herokuapp.com" instead. If we could do this over, we would probably think about our name more carefully – even though it may not be as important as a technical aspect, branding and distinguishability is still important for a marketable product.

Integrating the front and back ends (React.js and Django) ended up taking a significant amount of time, almost a week, to figure out. This was longer than we had initially expected. Ultimately, however, we feel that it was a good idea to build them separately before connecting them, because as stated above, doing so would allow us the flexibility to switch out a system we were using or use a different framework if needed.

FUTURE WORK

There were a number of features that we would have liked to implement given more time. Although the semester is ending, we believe that our product has great potential to help the larger Princeton community, and we plan to continue development and maintenance of this project past the conclusion of this class. Any additional feature we add has the risk of overcomplicating the system and user interface, but we have identified a few that we think would be particularly helpful.

One such useful feature we wished to include was groups. This would allow a user to add people to a custom and named "group," and the schedules of everyone in a group could be toggled on and off at once. This would be helpful if meetings need to be repeatedly scheduled for the same group of people such as those in a problem set group or a club.

Although many things are scheduled on a weekly basis, we would also like to add support for one-time events in the future, and events that recur at a different frequency than simply every week. In addition, sometimes a user may wish to make public the time slots during which a user are busy, but not the name or details of the specific event a user has in their schedule at that time. In the future, we want to add the ability to make events "private" in such a manner.

The django-friendship library that we used included a system for blocking other users. The system would work as follows: if a user wished to stop all interaction with another user, they could block that user, which would break any follow connection they might currently have and prevent any more follow requests from being exchanged between them. If a user no longer wishes to block another person, they can unblock them. We implemented blocking on the back end, but we decided not to include it in our current version of the product, because we wanted to prioritize perfecting other features.

Lastly, because of the widespread use of Google calendar, we hope to add support for it in the future, so that users may import a current calendar from an outside source without having to manually fill in everything.

ADVICE

We try to distill our experiences described above into the following points of advice. Although some of these statements may seem obvious, we feel that they bear repeating.

- **Start early, plan well, aim high.** Time is always of the essence. If you set high goals and fall short of them, you'll likely still have a good product; if you set low goals, then you might not make it to a working product. Figure out all the intermediate steps of what needs to be accomplished.

- **Choose your tools wisely.** Do your research! Ten minutes of planning can save you hours of future work and changes.

- **Choose your partners and roles wisely.** Our team benefited immensely from a strong, focused, organized, and intelligent leader. Whoever you choose as the project manager needs to be able to handle that responsibility. For the team as a whole, try to balance your personalities and skill sets/levels - people have all different backgrounds, and are suited for different things.

- **Ask for help.** Don't feel like you have to do everything alone. You may think it's faster to figure things out on your own rather than spend time catching up your partner on what you're struggling with so that they can help you. Or, you may feel ashamed that you haven't succeeded at a task that was assigned to you. But that's not the way to go - even if you might be separating tasks, this is not a

single-player game, but a team effort. Don't let yourself fall behind. Don't be afraid to ask for outside help, either.

- **Stay organized.** For a large project like this, there is a lot going on. Having a documentation log was very helpful for us. Use Github well.
- **Communicate!** All of your team members need to know your current status, including what you've accomplished or are currently working on. If you fall sick, miss a meeting, etc., your teammates need to know that as well. Delegate tasks appropriately, so that everyone has something to work on that they are reasonably capable of accomplishing.
- **Take care of yourself.** Eat, sleep, take breaks. Please don't give yourself carpal tunnel from COS333 (and no, this is not baseless advice).
- **Take pride in what you're doing.** It's good work.