WARD Report

Theodore Ando, Yanjun Yang, Eitan Zlatin, Jacob Edelman, Alexandra Palocz

Reflections

Overview

Timeline and Goals

In our design document at the beginning of the project, we created a timeline that described our minor goals, major goals, and what we intended to complete each week. Our schedule was mostly shaped by the fact that Room Draw starts in mid-April, and we aspired to have a minimal viable product by the time Room Draw begins.

This worked well for guiding us during the first few weeks, but after that, it became apparent that our original goals were not in line with our aspirations of what we wanted to provide to our users. Although we had basic features, such as searching and favoriting, done by mid-April, we decided not to release the application for public use during the draw and instead to refine what we wanted the user experience to be. There are two big examples of this, which we shall briefly mention here, and expand upon in later sections.

First, we thought that having an interactive map of campus was going to be an integral part of our application, and we needed to leave room for it on the main page. We wanted it to have nice animations that really showcased the integration. However, we later realized that the primary advantage that would distinguish our app from others relied on leveraging data sources, so we decided to shift focus away from implementing the map feature to focus on polishing other parts of the application.

Second, we had initially wanted to have a nice system for our users to form groups on our website. But after a few weeks, we came to the conclusion that many of our features, such as likelihood prediction and automatic sharing of favorites, are only applicable after groups and draw times have already been announced by Housing, and the step of creating groups on our website unnecessarily complicates the user's experience. So, we decided to remove many features related to the creation of groups.

Despite these setbacks, in the end, we managed to achieve our minimum viable product, as well as our alpha and beta test checkpoints, almost on schedule.

Planning Practices

One of our biggest challenges that we had run into during the project was in our planning. As we mentioned above, we would have benefited from having a more refined plan of what we wanted to provide to our users. We perhaps also would have benefitted from doing more research on the comparative advantages of frameworks, such as Bootstrap and jQuery vs. React or Angular, rather than diving into the project initially.

Design Decisions

User Interface Design

We wanted to create an intuitive and easy to use design. We added on autocompleting multiple-select fields and a generalized search field, as well as sorting by relevant statistics, such as room area and likelihood, because we believed these features are important to help users easily find rooms and refine their searches. Overall, while we are very satisfied with our final design, we found it to be much more of an iterative process than we had originally imagined. Below, we highlight two examples of this.

We made a number of modifications to how we wanted to manage a user's groups and favorites over the course of the project. Initially, we planned to have users manually add their groups members to their groups, but this became more complicated once we considered how students often have multiple draw groups. Eventually, we settled on a system where we would automatically fill in all the draw groups based on the information published by Housing. This has the drawback of only allowing users to use the product's collaborative features after draw groups are released, but the benefit that it handles students in many overlapping groups gracefully and requires minimal input from the user.

The design of the search page likewise changed as the project progressed. Originally, we visualized having a map that the user could use to filter rooms by building or see where a building or room is located. We realized, however, that this would take too much work for little benefit, so we chose instead to focus on having a streamlined and easy-to-understand text-based filtering system rather than incorporating graphics. We also originally thought to display search results as a list of room cards, allowed a user to collapse the card to view more information on each room. However, this caused our app to be less responsive, as it was difficult to make it look good on different screen sizes. Ultimately, we realized that being able to easily compare rooms across the list was more important for the search results than seeing details on individual rooms. With this in mind, we used a data table structure that would let you sort based on each of the room's features (likelihood, area, etc.) and would scale better for different screens.

Database and Information Storage Design

To store information for our website, we also went through many iterations of what we information had wanted to store and how we wanted to store it to match the features that we support.

Initially, we thought that we could use a simple SQLite database to hold our data because we did not have that much information, so we did not think that speed would be an issue. However, relatively early on we realized that our app needed to seamlessly support many simultaneous users due to the nature of Room Draw. SQLite turned out to be an inadequate means of storing data for multithreaded access, and so we made the choice to switch to a MySQL database server.

We have many tables in our database for keeping track of the multitude of many-to-many and one-to-many relations in our data, such as users to favorited rooms, users to groups, rooms to reviews, etc. Part way through the project, we were constantly thinking of new features and data that we might want to store. This meant that the the backend team was having to perform many database migrations per week, which was wasting a great deal of time. To avoid this, we tried to design a general schema for our project that could accommodate many different desired features via the application logic layer. For example, we debated whether we should have lists be related to users, or groups, or both, whether people should be able to create lists, or have a fixed number, whether you should be able to share lists with people outside your draw group, etc. We recognized that all of these could be implemented in the logic layer, if we had a general enough schema for our database. So we settled on make an abstraction of a ranked list of rooms that could be arbitrarily shared between users and groups. In the logic, we ended up enforcing that a user has a fixed number of lists, and they are shared only within their groups. However, this could be easily changed by future developers.

Languages and Systems

Frontend: jQuery, Bootstrap

We used jQuery to dynamically load content into our website, as well as to program all of the user interactions in our code. Although there are other means of assigning interactions to DOM elements (such as through the “onclick” attribute), we thought that it would be best to use jQuery exclusively with the goal of making our code base easier to understand. However, we found that there were also significant tradeoffs with using jQuery. Although jQuery allowed us easily manipulate any element on the DOM we wanted, it provided very little structural support for our project: It did not facilitate a separation between the HTML, styling, and functionality, especially when elements were being dynamically inserted into the webpage. Often times the CSS and HTML were simply part of the javascript file, such as when we were creating the “big card” that displayed reviews.

If given the chance to redesign our frontend, we would have considered using a web framework such as Angular 4.0. This would improve maintainability by enforcing a separation of concerns, making it easier to reuse code and abstracting away unnecessary details. While such a framework would necessarily have a much steeper learning curve than jQuery—which we were able to pick up in a matter of hours—it could have payed off in the long run.

We used Bootstrap to make our website more responsive and look good on a variety of different screen sizes and resolutions. We found that Bootstrap was useful in making our website look more streamlined and uniform; however, we also found that Bootstrap did not provide a few features that we wanted, such as multiple-select inputs, prompting us to use extra modules like Select2. If given the chance to do it over again, we would have considered React and MaterialUI instead, as we witnessed other groups using them to great effect for a smooth look, and because they have a larger set of features and components than Bootstrap.

Backend - Flask, Pony

We decided on Python and Flask as our backend language and framework because everyone in the group was relatively familiar with Python, and some people had good experience with Flask. Python code is also easy to write, which allowed us to get a running server up quickly and facilitated our testing. We also thought about using Django, but decided against it because it seemed too heavy duty for our small application.

Flask does not include its own ORM, so we decided to use PonyORM to manage our interactions with the database layer. We chose this because the syntax of its queries and objects is more pythonic than any other ORM, which meant that we spent almost no time worrying about how we stored the data.

Deployment

We wanted to avoid the hassle of setting up AWS or Azure hosting, buying a domain, etc, so we opted to request web space from the Computer Science department. They gave us our domain name, rooms.cs.princeton.edu, as well as server space and MySQL database access. This proved to be useful because we did not have to worry about setting up the server, Apache, domains, or anything else related to server maintenance.

The only challenge we ran into was that by default, the Apache instance was set to use CGI scripts, which meant that every session with our website started up a new running application, and at the end of every session it was destroyed. This led to long load times, and we had to spend a while figuring out how to get around this. We ended up using Phusion Passenger to serve our application in a robust multithreaded manner.

Version Control : Github

We decided to use user branches to house experimental changes, feature branches like frontend and backend to house well tested pieces of code, and the master branch to house the final product, which we made an effort to always keep in a clean and working state. Overall, Github was a great convenience in making sure we could coordinate our efforts. However, along the way, we did run into issues with binary files. We would have benefitted from keeping to the practice of adding all of our binary files to our ".gitignore" file as soon as they are created, so that we do not run into merge conflicts with these files.

Testing

Self-Testing

We initially discovered bugs principally by accessing the webpage ourselves, making sure that everything was displaying correctly, and iteratively editing our code until it appeared to function well. To give an example. we would search for rooms on the search page and favorite them, make sure they appeared on the favorites page, then unfavorite them and make sure they were deleted. We knew that we wouldn't be able to find all the usability issues this way, as we were already intimately familiar with the system, so we decided to open up the website to beta testing as soon as it was ready.

Beta Testing with Other Students

Unfortunately, our application was not ready for beta testing during Room Draw, so we had to rely on asking our friends and associates to click through the website and give us feedback afterward. We gave them minimal instructions to make sure they could discover confusing or unintuitive features, and they helped expose a few bugs, such as incorrect redirects, on our website.

We would have liked to have had a working version of the app before Room Draw ended so that we could have had more real users. We hope that during Room Draw next year we can deploy the app to a wider audience, and we installed Google Analytics so that it will help us improve the app when that time comes around.

Group Organization

Overall, one of the most difficult parts of the project was coordinating people. We tried many different strategies throughout the semester: holding weekly meetings before meeting with our advisor to discuss what to talk about and what we wanted to get done in the following week, holding frontend/backend only meetings to discuss specific features, and holding work sessions where we all collaborated. Unfortunately, even now, we are still unsure about what we could have done differently, besides being more consistent with our meeting times from the very beginning, to improve our organization. Perhaps in the future, with more experience with the dynamics of working together with a team of this size, we would be able to improve.

Advice to Future 333 Groups

A 5 times larger project than your typical COS assignment can much more than 5 times the work, because you can't simply roll over any mistakes or poor decisions, either in terms of structure, design, or choice of tools, that you made at the start--they continue to compound until you either fix them or waste endless hours working around them. Plan and do lots of research before you write your first piece of code!

Try to have shorter whole-team meetings where the agenda is well-defined, and longer work-sessions with smaller subgroups to get work done. We found that we were most productive when smaller groups of 2 or 3 split off, discussed one issue, and then presented the result to the group - remember that communication is O(n^2)!