RoomMeet: Report

Brian Chen, Dorothy Chen, Michael Danielczuk, Leonidas Tolias, and Campbell Weaver

Introduction

This project was definitely a great learning experience for all of us–our collective previous COS career had only consisted of the prerequisite courses for COS333. Thus, everything that wasn't Java or C was a new realm for us. Learning about all the different aspects of an application in class, implementing them in our project, and then finally seeing it all work together was very satisfying. Overall, it has been a successful journey!

Milestones

We had mixed success with our milestones. The earlier ones were difficult to meet–and we missed the first 2 or 3 mostly due to the lack of a solid project idea at that point in time and due to our general lack of experience with the technologies we used. Just to get started, we had to spend a good amount of time reading documentation, learning, and experimenting, and that made it hard to get something substantial done for our earlier milestones. Another issue that we had with meeting the earlier milestones was that we didn't know the order in which to do things–should we work on the database first? Or should we aim to get a working website up as soon as possible?

After the first few, however, we did well with meeting the remainder of our milestones. We had finished learning about the technologies and actually started programming, so things progressed much more quickly and smoothly. Once we got our core functionality up and running, it was more of a matter of continually adding features that we thought would be helpful and making the website more visually appealing.

It would have been better and easier if we had several smaller milestones in addition to a moderate number of large ones–perhaps two or three small goals that would help us achieve each major milestone. This would have guided our progress better, and additionally made it easier to gauge how we were doing in terms of time. Furthermore, doing this would have split the project into even smaller chunks of things to do, which is generally easier to tackle and manage than having larger and fewer blocks.

One other thing that we wish we had done better in terms of milestones is more properly defining the alpha and beta tests. Our descriptions could be more specific and better phrased–they were actually a bit contradictory, which we think is because we didn't have the knowledge or prior experience yet to make a realistic vision of what we would be able to accomplish by that time.

Experiences with design, interfaces, and testing

While implementing our design, it was crucial to have a working website in order to help determine whether or not things were working correctly. You could always make changes to code and check function outputs, but only when testing on the website were we able to find bugs and things to improve upon. Once we saw something wrong, we would always go through the same thought process for debugging: Is there a problem with our Javascript? Is it the html formatting? Is it the data processing? Or is it our database? Tracking backwards through these steps was helpful in pinpointing the location of the bug and fixing it.

Testing with a working website, as stated above, needed to be done frequently because we had to make sure that each new feature we added worked and didn't break any of our previously existing ones. However, pushing to Heroku each time would be inefficient and could lead to conflicts if people were experimenting with new code at the same time. Thus, we used our localhost to test out our changes before merging to GitHub and pushing to Heroku. Once we integrated CAS, however, two of our group members' localhosts no longer worked because they could not handle patch through to CAS login and authentication. (Why the rest of us were able to do it is still a mystery.) They couldn't disable the CAS login either, because our major features revolved around storing user information based on their CAS netid. Thus testing was made much harder for them, and so they often had to work side by side with other group members when making major changes that needed to be tested before committing/pushing.

Throughout our development phase, we constantly tried to break the website and find out what would happen for unexpected user behavior: clicking different combinations of buttons, entering invalid input, shrinking the browser window to different sizes, etc. We found most of our bugs this way, and going through these checks each time we made a major change was crucial to having a working site at the very end.

We also invited our friends to log in and explore the site throughout different stages of our development process. They gave us feedback on aesthetics, functionalities, and things that were confusing to a first-time user. This led us to add instructions, error checking, and much of the jQuery used for smooth user interactions. We also wanted our page to look clean and simple, which we accomplished using user feedback, both from us and our friends.

Surprises

Unpleasant surprises: for something that's so commonly used, it's harder to get a hang of Github than expected. We ended up having issues with it once or twice a week throughout the entire project period, even at the end. Sometimes, things would be bad enough that we'd just have to delete our local files entirely and reclone the repository–trying to manually resolve conflicts was either not worth the amount of effort required, or somehow managed to make things worse. This, needless to say, set back progress (though thankfully, we never lost large amounts of progress this way). Besides this, syncing also failed to work sometimes, which means that we don't get the most up-to-date files. This caused more conflicts, which only made things worse. (Future COS333 Project–make a better Github desktop client and website?)

Unpleasant surprise part 2: installing things, really, really sucks. We were never expecting this stage to be a walk in the park, but we also didn't expect it to be an unbelievably frustrating, draining ordeal. A part that was exceptionally annoying was when it would install perfectly on one person's computer but not another's, even though we were all following the exact same steps.

Pleasant surprise: Django is extraordinarily easy and nice to get started with, and the way it works (views, urls, templates, etc.) is intuitive and helpful. There's a free online immersive tutorial book that explains the basics very well, and the documentation everywhere is great. Django also made Postgres database interaction super easy so that we would not have to deal with writing SQL. Django <3

Pleasant surprise: CAS was much easier than expected to get working. We heard several horror stories about how challenging it was to implement CAS from multiple people, both past and current students of COS333, but we managed to finish our implementation in a single day. Django-cas, the library that we used to do this, was a little outdated but fixing it wasn't too bad. The Princeton Security and Data Protection people were also super responsive and helpful for getting our initial Heroku domain approved, and then later switching to our current domain. We also heard some stories of CAS being unusable with Google App Engine due to constantly changing IP addresses (or something) and requiring roundabout fixes, but it happily ended up working perfectly fine with Heroku.

Major choices and decisions

We chose to use the Django framework for our project, since most of us were familiar with Python. Django also allowed for easy implementation of PostgreSQL as our database, as well as straightforward hosting with Heroku. Bootstrap allowed us to easily style our web-based application, and we used the jQuery/Ajax Javascript library for interactivity to dynamically update the site so that the page does not need to reload for each changed input. Finally, we used the Google Maps API for our map. Since people looking for summer housing/roommates would most likely be doing it on their laptop in their room and not on the go on a mobile device, we decided that it was not necessary to make a mobile application or customize our site to run on mobile devices.

We chose to make our website accessible only to Princeton students, requiring CAS login. We did this because people would probably be more comfortable sharing their information, meeting, and rooming with strangers if those strangers were also Princeton students as well. Our Princeton community is (relatively) safe and free of creepers.

An option that we were considering was allowing users to also log into their Facebook accounts in addition to CAS. The main motivation behind this was to tap into Facebook's messaging app, so we could have on-site live messaging between users to help them discuss summer plans without having to turn to an external site or application. Connecting with Facebook would also open the site to lots of new possible features, such as seeing where your Facebook friends would be staying over the summer, and even adding new people you meet on RoomMeet as a Facebook friend. However, this ended up not being possible because the Facebook API doesn't play well with CAS, since they both wanted to set the user profile to their own variables.

The next option we considered was to then have users connect to Facebook instead of CAS, so that we could still have the messaging feature easily implemented, and users could see what their friends were up to. The benefit of using Facebook is that it has a much larger network, so our site would be quickly populated with lots of users. However, doing so would compromise the security and privacy aspect that we wanted from the beginning. In the end, we chose to ditch Facebook; the tradeoff of in-site messaging for less security was not worth it.

Ever the stubborn visionaries, we still wanted in-site messaging, so we looked into writing our own messaging client. While it would take a lot of work and debugging, it was definitely a doable option to consider. However, we realized early on that it would require storing a lot of data, and that the more people chatted, the more space we would need. Since we were using Heroku to host the website and database for free, hosting the messaging application would exceed the hosting limits (and make us pay money), so we finally chose not to do messaging.

Around the time of our beta test, we noticed flyers on campus for a very similar summer housing website called spot.ac, which also received the backing of Career Services and USG. We wanted to make sure that our product was clearly different, so we decided to implement a housing feature that would allow users to drop available housing options on the map for others to see and rent. The housing option also allows the contact email to be a non-CAS email, so people offering housing such as Princeton alumni can contact current students and have them list their housing offers and contact information on our website. We believe that this housing option greatly facilitates the search for summer housing in addition to finding a roommate through RoomMeet.

If we had more time...

We would implement a few more features. The first would be an on-site messaging feature, provided that we get funding to pay for hosting on Heroku. This would be more convenient than contacting other users via email, as it contains everything within the site. It also makes it easier to track the conversation, as well as the location of the person you're talking to. Furthermore, on-site messaging would better preserve privacy because users could not directly email each other, thus eliminating the need to share email addresses with strangers. While this privacy concern is not that large of an issue now (because users are all from Princeton and could just look up someone's email using their name), it would definitely be more of an issue if we were to, say, switch to logging in using a Facebook account.

Another feature we would potentially add is a more comprehensive housing page to better connect students to available housing near where they're staying. Right now, we just have a user-inputted description of each available housing location. While that is useful, it would definitely be more helpful to add additional fields, such as: cost, square footage, occupancy. Furthermore, it would make searching for suitable housing easier if we were to implement filters based on these fields.

Right now, students are able to add housing for other people. This system technically works, but is not really convenient or efficient at all, since most students will not have a house to offer, and people potentially offering housing such as Princeton alumni would need to know a current student to enter their housing listings. Something we could add that would fix this is some sort of new user type for people (not necessarily students) who are just offering housing; these users could add their houses but would not be allowed to see where students are staying. This approach preserves students' privacy while also increasing the amount of available housing options. This increased amount of housing would also warrant another filter for people vs. houses to hopefully de-clutter the map.

How would we do things differently?

We would allocate more time for installing things, which, as we mentioned before, was an unexpected massive time-sink and headache-inducer. Aside from giving it more time, however, there doesn't seem to be anything else we could do besides hope for the best.

We would also be much more modular with our code from the beginning; we weren't careful and ended up with massive files, which made it impossible to find anything easily (functions, variable names, bugs...). We would either have to ask other members if they remembered where certain functions were, or manually search through all the files ourselves.

In a similar vein, we would also comment our code more carefully and thoroughly. It was hard for one person to debug something that someone else wrote or to expand the functionalities of someone else's code, which hindered progress.

Also mentioned before was having more numerous but smaller milestones. These would help (or, from another perspective, force) us to stay on track; it also creates a more straightforward and defined course of action and progress, instead of adding code or making changes whenever or wherever we thought was good.

Finally, we would delegate tasks and designate people to do specific things more clearly. We ended up having everyone do a little bit of everything, and that got a bit chaotic sometimes. While this meant that we all had the opportunity to learn all of the technologies we used, it would be more structured and better to have specific people delegated to work on the backend or the front end, etc. That way, when some specific part of the code goes wrong, there is one (or two) expert(s) to turn to, instead of having everyone scratching their heads and trying to find the bug at the same time.

Advice

Definitely allow yourselves lots of time for things generally going wrong, especially in the beginning (cough cough, installations)–this is where we had the most large issues with our code (as opposed to small bugs and minor fixes).

If, in the beginning, you find that something's just not working well, don't hesitate to switch and use something else instead! You probably shouldn't do this later on unless you have to, though. Choose a technology that you're comfortable with, or else make sure that you and your team are very dedicated to learning something new!

Start with the basics and make sure those are working before expanding, even if that means you're falling behind schedule. Keeping up with your milestones and staying on track is important, but you can't build anything without a strong foundation and understanding what you're working with. Don't rush ahead to later find that you have to rebuild something because it was hastily done the first time!

Find a group and brainstorm ideas early on. This way, you can hit the ground running once the second half of the semester starts (or even earlier!), meaning you can spend time actually building your project instead of wasting it figuring out what your project is.

After the weekly TA meetings, set a goal of what you want to accomplish before the next meeting and make sure to write it down. The TAs have valuable insight and input, and writing things down ensures that it won't be forgotten. It also serves as a reminder throughout the week, which makes sure that some amount of progress is always made, no matter how little. (Baby steps!)

Don't forget to update your timeline! You'll need to turn it in at the end anyway, and failing to update it as you go means that you'll later forget the exact details of what you did. It also forces the group to communicate about what everyone's been working on, which helps tremendously in keeping everyone up to date (...and feeling guilty about not working).