ReCourse
Final Report

Jessica Zheng (Project Leader)
Natalie Diaz
Elizabeth Tian
Bill Zhang
Julie Zhu
A. PLANNING AND MILESTONES

1. TIMELINE

When creating our design document, we wrote out a detailed timeline that described our goals, “stretch goals”, and tasks for each week; this proved to be very practical, as it gave us a high-level overview of our progress on the project throughout the semester and a measure of how on-track we were every week. Something that was useful was incorporating time-sensitive events into our timeline -- for example, the new course offerings were released on April 6, and updates to certificates would follow afterward; this allowed up to plan around these events accordingly. Overall, we were very satisfied with our progress as it related to the timeline we set out for ourselves; often, we met and worked ahead of our timeline, allowing for us to implement the “stretch goals” for our application, such as a certificates filter and course conflict detection.

Ultimately, we were able to achieve all of our milestones on time or earlier than anticipated, including creating our Minimum Viable Product, completing our project prototype, finishing our “stretch goals”, and beginning to beta test with real users after deploying.

2. PLANNING PRACTICES

As mentioned earlier, we found it crucial to have a clear plan set out before beginning the execution of the project. Furthermore, identifying a minimum viable product that we could build and then classifying various “stretch goals” within this plan allowed for great flexibility, as once we finished our MVP we were easily able to build onto our project and improve it by adding new filters. Furthermore, we were pleasantly surprised by the ease with which we could add new filters using this process, as we were able to quickly add filters that we had not planned for after receiving feedback from users (e.g. trips, no grad courses). We would suggest to future groups to select a project that allows for this flexibility -- if the project group is unable to reach the stretch goals, there would still be a completed and usable product with the MVP, and if there is more time after the MVP is completed, the group can still improve on the product by completing the stretch goals.

We also chose to split the project among our group in a way such that we were able to work one part of the project without needing to wait on another function to be implemented. We chose to divide the project between frontend and backend, but first set up the middleware between the two as soon as possible, so
that each side could easily make changes while also ensuring that nothing on the other side would break.

In addition to meeting/working individually and in teams (frontend or backend), we also met as an entire group weekly, in order to ensure that everyone was on the same page. In these weekly meetings, we kept “Meeting Minutes” that detailed our progress since the last meeting, the current concerns, and the to-dos and goals for the upcoming week. This allowed our group to stay organized and understand what was being done, as well as what needed to be done to keep up with our timeline.

B. DESIGN DECISIONS

1. USER INTERFACE DESIGN

From the very beginning of the project, having a clean, intuitive, and aesthetically pleasing user interface was of great importance to us. In our goal to make ReCourse a single-page application, we had to evaluate the necessity of each UI component and only keep the ones that we prioritized. Questions we faced:

- Do we show all the potential search options and risk cluttering the page?
- How much information should we display for each course in the results?
- Where and how should we display favorited courses? How often would a user want to see their favorite courses?

Our solution was the use of dynamic UI components, which we think worked out quite well! Advanced search options, detailed information about a course, and favorited courses would only appear when a user specifically requested to see them. By hiding favorited courses in a side drawer, we had more space on the page to display search options and results.

Programming the User Interface

A major coding style decision was whether to move each input component (e.g. checkbox, text field) into its own separate file, or keep all input components together in assets/js/index.js. We decided on the latter option. While the former option may have yielded more modular code, we deemed that the overhead complexity of passing data states down to each component and creating callback methods for each component was more trouble than it was worth. Furthermore, we already had quite a few external javascript files that we used to store data for input fields with menu and autocomplete options. Our chosen approach was ultimately more straightforward and made our code easier to debug.

If we were to rewrite ReCourse, we would have prioritized making a responsive website (as in responsive to different screen sizes) earlier. We only decided to enable responsive design over a week after we had started, and as a result we had to spend a whole Saturday rewriting our UI from scratch using a different Grid layout system.
3. DATABASE AND INFORMATION STORAGE DESIGN

Near the start of our project, we decided that we wanted to host our app on Heroku, so we decided to use Heroku Postgres for our database. Although Django defaults to using an SQLite database, it was fairly straightforward the change the database.

For our database table structure (full structure in our GitHub README), we started by looking at the ReCal GitHub. Many of the more straightforward fields in our Course and Meeting tables were identical to ReCal's (e.g. storing the registrar id as a character field). We also had a Profile table to hold the favorites for all users since the User table and other authentication-related tables were abstracted away and managed by the django-cas-ng package.

One key design decision was to use boolean fields for pdf- and audit- ability, as well as grading requirements (e.g. papers, exams). This was done to speed up filtering: instead of having to parse the grading fields of all the courses, the filter method would just need to look at the boolean field. Similarly, we used an integer to store the pages of reading per week instead of parsing the reading/writing assignments upon each filter request. The values for each of these fields was populated while importing data from our scraper to the database.

Another design decision we made was to use a Text field to hold favorites, rather than a relationship with a Course object. This was because we often had to rescrape to populate new fields or add new courses and meeting times, and re-scraping occasionally messed up relationships with Course objects.

If we were to change one thing, we would either add a term field or add a Professor table to keep track of ratings and past classes taught by a professor. We had initially planned for our site to just search classes in the current semester, but several users expressed interest in historical data.

C. LANGUAGES AND SYSTEMS

1. FRONT-END: REACTJS, MATERIAL-UI, REACT-BOOTSTRAP

Although ReactJS did have a bit of a learning curve, its component-based structure provided advantages over a template-based structure that one might find in Bootstrap. In particular, React's approach -- in which a single piece of functionality is defined in a single place -- helped facilitate code reuse. This in turn made it easier to maintain and grow our code base. React also had a great set of Developer Tools that simplified debugging. We highly recommend ReactJS to future COS 333 groups, especially if they would like to display a constantly changing data set.

Material-UI was great because it allowed us to break free from the standard Bootstrap form components and really customize how our site looked. In addition, the Material-UI library offered components that Bootstrap did not directly have, such as Autocomplete text input, chips, and multiple-select dropdown menus. For the most part Material-UI was straightforward to use, but at
times required a very close reading of the documentation to accomplish a specific task.

React-Bootstrap allowed simple implementation of our Grid Layout system, thanks to its thorough documentation. Although we did not base much of our UI on React-Bootstrap components, we think React-Bootstrap would be a great choice for future COS 333 students who would like to try out the React design philosophy, but also want the look and feel of Twitter Bootstrap.

2. **BACK-END: DJANGO, SERPY, BEAUTIFUL SOUP**

We wanted to use Python in our web application because of its simplicity and familiarity (after using it for a few COS 333 assignments). Although Flask may have been simpler to use, we ended up using Django mostly because of its database interfacing features, i.e. its object-relational mapper (ORM) layer. Our project required some complex database queries and Django's built-in filter/exclude made these queries easier to carry out. Django's ORM layer also provides built-in protection from SQL injections and makes the database queries safer overall.

Although we had originally used Django’s built-in serializer to convert our data into JSON, we soon discovered that serialization was a performance bottleneck when filtering courses. So we changed over to serpy (over other packages) because it focused on speedy serialization.

Beautiful Soup is a commonly used package for web scraping, but we mainly used it because it is the package used by the Assignment 4/ReCal scraper (which we built off of) as well as the USG Labs’ course evaluations scraper, which we used in conjunction with the Assignment 4/ReCal scraper.

3. **DEPLOYMENT: HEROKU, WHITENOISE**

We used Heroku to deploy our project because Heroku allowed us to get our application up and running quickly and simply. Heroku also had extensive documentation for their built-in tools and architecture.

Part of Heroku's extensive documentation included instructions specifically for Django apps, which allowed for very easy setup and deployment. In order to make Django work together with Heroku, we chose to make use of the WhiteNoise project, which to serve static assets in production, as Django does not support this by default.

4. **VERSION CONTROL: GITHUB**

Like most other COS 333 groups, we used Github to help facilitate collaboration. We found it useful to keep multiple branches of our code base, especially when the application was live and had active users. The master branch contained the currently deployed version of our code, while in-progress features were kept in other branches until we were ready to merge them back into master and deploy. This way, if someone found an issue in our live application, we could quickly make the according fixes and redeploy the master branch without deploying half-finished features.
D. TESTING

1. SELF-TESTING

We discovered most of our bugs through extensive self-testing (especially with regards to course conflict detection). We would try a variety of test searches and check to ensure the returned results made sense. Many times we would chance upon an odd edge case which we hadn’t thought of. We would then go back to the code and fix what was causing the issue, and then perform more tests to make sure we hadn’t broken anything else.

However, given that we were all COS majors who had worked closely with the code, there was likely an implicit bias in the kinds of classes we would search for and the search criteria we would use. We knew it was important to open up testing to other users as soon as possible.

2. BETA TESTING WITH OTHER STUDENTS

Fortunately, with course selection coming up, it was easy to find friends and classmates who were willing to test our web app. The testing process was simple. We wanted to see if the website design was intuitive enough that users could figure out most of the features without guidance. We gave little instruction and let the users play with the site on their own.

Once we deployed our web app on Heroku and publicized it to the Princeton community, we were able to test our product on a broader scale. Having a large number of users was useful for general case testing and stress testing, as many users might be on the site at once and could come up with search criteria we hadn’t tried before. We also added an anonymous feedback form to our site, which proved to be very valuable, as students could voice their genuine opinions.

We did our best to incorporate user feedback as much as possible. As our web app ultimately was intended to be a tool for Princeton students, we wanted to cater our services to what would be most useful for our users. In fact, several current features were actually the result of beta tester feedback: (multiple) exclusion of distribution areas, showing and hiding graduate-level courses, keyword description search, classes with trips/excursions, and the “I’m Feeling Lucky” randomized course easter egg.

3. AUTOMATED TESTING

To test the filters, we used Python’s unittest module since Django’s built in way of testing required creating/destroying databases, which was something Heroku did not allow for. In addition, we only used automated testing on a few more complex filters. It felt a bit unnecessary to test the simpler filters since we would basically be testing the functionality of Django’s filter/exclude methods rather than the correctness of our code. The few filters we tested were days, course conflict, and pdf- and audit-ability.

In general, testing worked as follows. We would create a ‘test’ user and call Django’s force_login to get into our login protected views. Next, we made the
appropriate API call and converted the result into JSON. Finally, we would go through each course and make sure the course satisfied the filter used in the API call. This ensured there were no incorrect results returned. It would have been nice to also test that we returned all courses that satisfied the filters, but we could not think of a good way to obtain such a list of courses that we could compare our results to.

FINAL THOUGHTS

Although we’re very satisfied with what we have achieved, there are some areas of ReCourse that we would love to improve, given more time. We’d like to include course data for multiple semesters, rather than just for the Fall 2017 term. Another addition that was highly requested by beta testers is displaying enrollment information for each section of a course. In order to keep the data as accurate as possible, we would periodically rescrape the course offerings. Finally, we want to explore alternate ways of displaying returned courses such that more courses can be displayed on the page at once.

GENERAL PIECES OF ADVICE TO FUTURE 333 GROUPS

1. Getting real users early serves as an excellent source of motivation to constantly improve your product, and it improves team morale.
2. It’s never too early to start thinking about how your user interface will look and work -- even as early as the design document! Once you start programming the front-end, it can be very time-consuming to make major layout changes.
3. Holding regular (weekly) group meetings was key to keep everyone on the same page. In particular, keeping meeting minutes that detailed the progress since last week, the current concerns, and the goals for the upcoming week made meetings organized and efficient.
4. We found splitting our project and team into front-end and back-end work very intuitive. It also made it easier to find time to meet up and work together with fewer people.
5. Frameworks can have bugs too. If you have a really bizarre bug, don’t forget to check if the framework you’re using has any open/unresolved bugs that might be the source.
6. Always keep a backup copy of your database -- we learned this the hard way.