

# ReCourse -- a reinvented course search engine

**Project Leader:** Jessica Zheng

Bill Zhang ([wyzhang@princeton.edu](mailto:wyzhang@princeton.edu)),

Natalie Diaz ([nhdiaz@princeton.edu](mailto:nhdiaz@princeton.edu)),

Julie Zhu ([juliez@princeton.edu](mailto:juliez@princeton.edu)),

Elizabeth Tian ([etian@princeton.edu](mailto:etian@princeton.edu)),

Jessica Zheng ([jz9@princeton.edu](mailto:jz9@princeton.edu))

## 1. Overview

Choosing the right courses can make or break a semester. Unfortunately, the present lack of a straightforward way to sift through Princeton's many classes can make it difficult to find the perfect fit. We plan to develop a tool that addresses some of the issues with Princeton's current course selection process and facilitates this procedure for all Princeton students. We proudly present **ReCourse**, a simple yet comprehensive alternative to the Registrar's Course Offerings page.

ReCourse is intended to be an interactive course search engine. CAS authenticated users can search for courses or groups of courses that satisfy chosen filters. ReCourse will offer an easy-to-use interface and a wider range of filtering options than the Registrar's page, (such as pdf/npdf, pages of reading, and certificate—to name a few). Results from a given search will be linked directly to the Registrar's information page for the course, as well as its Course Evaluations. We hope that ReCourse will make it easier for students to discover courses that match their specific needs and interests, and we are looking forward to testing our project for the upcoming semester!

## 2. Requirements and Target Audiences

The current method for course search revolves around the Registrar's Course Offerings page. Through various filters, users can search for courses, then click on a given course's page for more information and for evaluations. However, Course Offerings is rather limited on what users can filter by. Though the current filters make up an essential backbone to the course search process, these minimal features do not consider some of the more unique concerns of distinct students. Furthermore, information relevant to a search is not revealed until a user clicks on the course page, meaning that users might have to click through a whole list of courses before, for example, finding a course that is PDF-able. Finally, despite all of the information on a course's page, there are some key details that are missing, such as whether or not a course counts as a departmental or for a certificate. Currently, this information can only be found on the department/certificate's page, making the search all the more complicated.

ReCourse will hopefully solve all of these issues. First, with a much wider range of filters, users will be able to further customize their searches, so that the returned results are better matches for each specific case. For example, we hope to implement course conflict detection, where users can input courses with which the returned results do not conflict. Second, by allowing users to search by things like PDF, there is no longer any need to click on each result in a list, as all of the returned results will satisfy those criteria. Finally, by directly incorporating major and certificate requirements into our search

engine, we remove the extra effort necessary to search a department's homepage. By expanding the search options, we hope to hone in on those perfect courses, while reducing the time wasted on courses that don't match.

With ReCourse, we are targeting any and all Princeton students, but our system will also be accessible to any CAS authenticated faculty and staff. We hope that our system will make the search for courses much simpler and more adaptable to the individual specifications of users. Course selection can be a hassle, but with ReCourse, it doesn't have to be.

### **3. Functionality**

The main purpose of ReCourse is to make it easier to find "the perfect" course by enabling inclusive filtering (courses that fit certain criterion) and exclusive filtering (remove courses that fit certain criterion). The use cases below demonstrate using these filtering capabilities.

The filters described in the use cases may or may not be in our final implementation. At minimum, we wish to implement the filters already done by the Registrar, as well as a simple PDF/no PDF filter. The other filters are more difficult to implement, but the ones we currently plan on prioritizing are: pages of reading per week, grading breakdown, and major/certificate-fulfilling courses. In addition, the display of results post-filtering will be more informative than the course offerings page, including information like course description and average evaluation score (this would require CAS authentication). Clicking on a particular result will bring the searcher to the original course offerings page if they wish to see more details.

#### **Use Case 1: finding "relevant" courses (inclusive search)**

Bob is a sophomore who has decided that he must take COS 333, ORF 309, COS 340, and a 300-level MAT course in a 5-course semester. His wiser friends who care about him insist that Bob at least take an easy 5th course. Bob relents, but only to some extent. He wants something relevant to one of the 5 certificates he's pursuing because otherwise, he won't be able to fulfill all the requirements. He initially attempts to pan through all 5 certificate websites to see what courses are relevant and then opens many tabs to search for all the courses in course offerings to see if they are easy enough. He eventually seeks recourse through ReCourse (ba-dum-tss). He can set filters for all 5 certificates and no papers (that's his definition of easy). Bob finds a course and is satisfied, but his friends don't know what to do with him.

#### **Use Case 2: finding "easy" last course that is not STEM (inclusive/exclusive search)**

Jill is a history major looking for that final class to balance out her reading/writing heavy schedule. She would like it to be pdf-able, in the case her grade by the midterm is not that great (Jill needs a good GPA since she wants to go to law school). She would also like it to be an exam course with not so much reading (since all her history classes already have a lot papers and readings). She searches for a course via ReCourse since it is able to filter by PDF-ability, amount of reading, and if the final is exam/paper. When she first looks at the results, she is dismayed by all the STEM courses. Luckily, ReCourse has exclusive filters too so all she has to do is add not MAT/COS/MAE/MOL, etc.

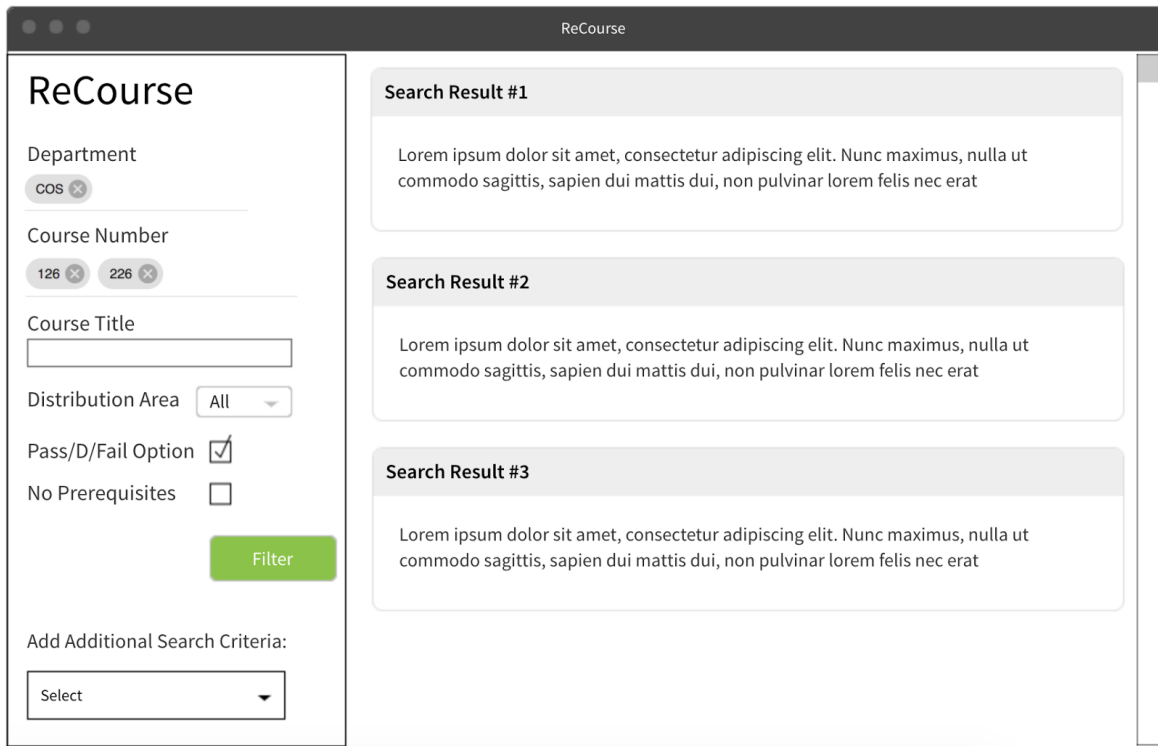
### Use Case 3: finding balance in the semester (exclusive search)

John is a COS major who really likes COS. However, John knows that if he takes more than 2 programming-intensive courses, he will start disliking COS and change his major to ECO (which is problematic for multiple reasons). But he's struggling to find that last course. John typically chooses his last course by going to the Course Offerings page and pressing search with no filters (essentially looking at all courses). He clicks into a course if the title sounds interesting. This is time consuming because there are so many courses to look through and sometimes, the course he clicks into turns out to have either a lot of reading (something no COS major likes) or is programming-heavy. If instead, John decides to use ReCourse, he is able to filter out the courses he definitely will not take and save some time. He can choose to filter out COS courses and choose a reading cap. He can also quickly see the grade breakdown of a course just from the results page, without having to navigate to a separate page.

## 4. Design

### USER INTERFACE

We plan to implement the frontend user interface with React JS. Our web app will be on a single page where search criteria and search results are both displayed. This makes it easy for users to modify their search criteria. Our current plan is to split the page with the course search options in a sidebar on the left, and have the returned search results take up the majority of the page on the right:



### Course Search Options

We plan on using a sidebar to display the different filtering options that users can select. The searches will not be “dynamic”, i.e. each time the filters are changed, the user must click a “Filter” button to

update the results. Within fields, the search will compute the union of queries, and between fields, the search will compute the intersection of queries.

At the top of the sidebar, we will include the filtering features that are already present in the Registrar's Course Offerings page:

- *Department, Distribution area, Course title, Instructor, Course number.*

We will also have a drop-down menu with which users can add and remove additional search criteria. We hope to eventually include the following filters, however we will complete as many as we can in the time we have:

- *PDF, Prerequisites, Grading criteria, Major/certificate, Conflict detection, Course evaluation, Term*

### Course Search Results

We plan to display each course in the returned result in a container. The top portion of the container would contain:

- *course department and number, distribution area (if applicable), days (section with the largest enrollment), time*

The larger bottom portion of the container would contain:

- *full course title*
- *professors (truncated if necessary)*
- *a description of the course (truncated if necessary)*
- *a course rating out of 5. This will likely be an average of the "Overall Quality of the Course" ratings obtained from the registrar's course evaluations.*
- *a link to the course's information page on the registrar*
- *a link to the course's evaluation page*



Sample results:

COS 333 | TTh | 11:00am-12:20pm

**Advanced Programming Techniques**  
Brian W. Kernighan, Christopher M. Moretti

This is a course about the practice of programming. Programming is more than just writing code. Programmers must also assess tradeoffs, choose among design alternatives, debug and test, improve performance, and maintain software written by themselves & others. At the same time, they ...

4.18/5

COS 217 | QR | MW | 10:00am-10:50am

## Introduction to Programming Systems

Aarti Gupta

Introduction to programming systems, including modular programming, advanced program design, programming style, test, debugging and performance tuning; machine languages and assembly language; and use of system call services.

4.06/5



### SCRAPING

In order to build our database, we will be scraping the Course Offerings pages. To do that, we will be using the Python program written by Alex Ogier '13 and kept in service by Brian Kernighan and Christopher Moretti as a basis, while adding on further code in order to store more extensive information. Currently, the database stores for each course: the professors (and their ids), the title, course id, listing department(s) and number(s), distribution area, prerequisites, description, and classes (class number, people enrolled, the enrollment limit, start and end time, section, building and room number, and days. We hope to add to this the PDF-ability of the class, the reading/writing assignments (stored as number of pages of reading), and the requirements/grading criteria (e.g. tests, take-homes, papers, etc.). As stretch goals, we would like to scrape the course evaluations and score for past semesters, as well as whether it fulfills any major/certificate requirements (however, this will probably involve crowd-sourcing data manually rather than scraping).

### DATABASE MANAGEMENT

We will be using PostgreSQL as a database to hold this information. The Django web framework provides an easy-to-use interface that transforms Python calls into the necessary database queries, so that the choice of database is less significant. If we are able to reach our stretch goals (specifically scraping course evaluations, which requires CASS Authentication), we will also represent “users” within our database for each Princeton student once they pass the authentication -- this will involve storing input from users, including their current schedules when implementing course conflict detection, as well as their major and/or certificate(s), so that filters can be tailored toward each student.

### PROCESSING

When the user makes a query, it sends an API call with a dictionary of all selected filters. We construct a QuerySet and iterate through all key/value pairs in the dictionary to make the appropriate filter conditions. This should take minimal processing if the database is set up so that all course data is pre-processed. We then serialize the data into JSON and pass it to the frontend.

This JSON data will be passed to the front-end via a RESTful API. This API will likely be constructed through the Django-Rest Framework since it provides both a simple way to create common API calls (e.g. when querying for a particular id/entry in a database) as well as ways to deal with more complex calls via overriding of method (something we will likely need to do to implement our filters).

Since our web page will be updated frequently (each time filters are applied), we have chosen to use React to display the page. Each time the filters are applied, an appropriate AJAX call will be made and the page updated. The querying and data retrieval will be done using jQuery in a React component.

## 5. Timeline

### March 20 - March 25 (Spring Break)

Goal: Begin setting up the frameworks and interfaces, ensure they work together smoothly. Start gathering the necessary information/data to write the application. Create project website.

Tasks:

- Begin learning the necessary languages/frameworks: React, Postgres, Python, Heroku
- Learn the Course Offerings scraping code provided in Assignment 4 and make changes
- Gather major and certificate data from students

Overhead:

- Send out a Google form/listserv blast to get major and certificate data
- Meeting with the registrar

### March 26 - April 1

Goal: Make progress on separate components of the application - this involves: completing the scraper, setting up database requests and processing with filters (specifically, complete filters that are already available on Course Offerings), and having a rough UI.

Tasks:

- Write/edit given scraping code to scrape more information on the Course Offerings page
- Set up a basic UI that pulls up course information when requested
- Write database processing code using dummy data, implement current filters

### April 2 - April 8

Goal: Make progress on separate components of the application - this involves: continuing to create our own filters (non-stretch goal filters, e.g. PDF, No prereqs, reading/week, grading criteria), and finishing up the basic UI (while potentially considering linking to Course Offerings, as well as ReCal).

Tasks:

- Interface the scraper with the database once all testing is completed
- Implement the non-stretch goal filters
- Continue to build the UI to make it look pretty, as well as implement the filters as they continue to be added

### April 9 - April 15

Goal: Project prototype finished (due 4/14). If time permits, begin "stretch goals" - this involves: major/certificate requirements, course conflict detection, course evaluation ratings.

Tasks:

- Write the scraping code for Course Evaluations
- Implement the major/certificate requirements filter
- Continue creating the UI to match these filters

### **April 16 - April 22**

Goal: Continue/begin “stretch goals” - this involves: major/certificate requirements, course conflict detection, course evaluation ratings.

Tasks:

- Implement the course conflict detection and/or course evaluations
- Creating “user” objects and implementing CASS Authentication
- Building the UI to match these filters (e.g. user schedule, user information page/section, etc.)

Overhead:

- Test the “user” objects by creating/using our own accounts

### **April 23 - April 29**

Goal: Alpha test on 4/28.

Tasks:

- Finish up any loose ends in the code, thoroughly test the application and debug with corner cases

### **April 30 - May 6**

Goal: Beta testing begins.

Tasks:

- Continue to test the application and debug
- Get feedback from the general Princeton population and use the feedback

## **6. Risks and Outcomes**

- *Learning Curve for ReactJS.* Professor Kernighan has cautioned us that React has a higher learning curve than alternate option such as Bootstrap or Vue. ReactJS also uses JSX rather than plain JavaScript. Our current plan is to devote the week of spring break to becoming better acquainted with ReactJS and JSX. If the learning curve proves to be too great or there are compatibility issues between React and Django, we plan to switch to either Vue or Bootstrap.
- *Data Acquisition via Scraping.* It would be great to test this on the new course offerings when they come out, but it would be difficult if the format of the course listing changes and the scraper doesn’t work. If the format doesn’t change too much, we could probably quickly write up a new scraper. But if the format changes drastically, we could still test the rest of our code using old course offerings and when we have time, scrape the new course offerings and put them in our database. Nothing else should need to change much if we properly separate our code into standalone pieces.
- *Changing Course Details.* Course details, such as room assignments, precept times, etc. are never set in stone and often change throughout the semester. Most notably, precepts are usually posted as P99 initially and then created at the start of the semester. Some classes and precepts are also cancelled based on student interest. If our database is out of sync with the registrar’s course offerings, this could lead to misunderstandings and user frustration. A solution could be periodically re-running the scraper.