CourseComb
Product Guide

HJ Suh
Ada Nguyen
Matt Yi
DoWon Kim

CourseComb User Guide

GETTING STARTED

CourseComb is a web application that allows Princeton students to automatically create several possible course schedules from a list of user's courses of interest and preferences. It can be found at https://coursecomb.herokuapp.com/. The website is CAS (Central Authentication Service) protected and can be logged in with Princeton University credentials.

SEARCHING FOR COURSES

What if you were trying to find a possible class schedule with 4 courses, but have too many courses you were interested in? Let's walk through an example to find your ideal semester schedule.

On the left column, click on the field labeled "Search Course Here...". You can search using course department (COS, MAT) or number, or both. Then click the desired course to add it to the "Course Queue".

If you hover over a class in the queue, a bar appears with the links to the course information page, evaluations page, and a delete button to remove the course from queue.

Once you have entered all the courses you are interested in taking, press "Create My Schedule!" to proceed.

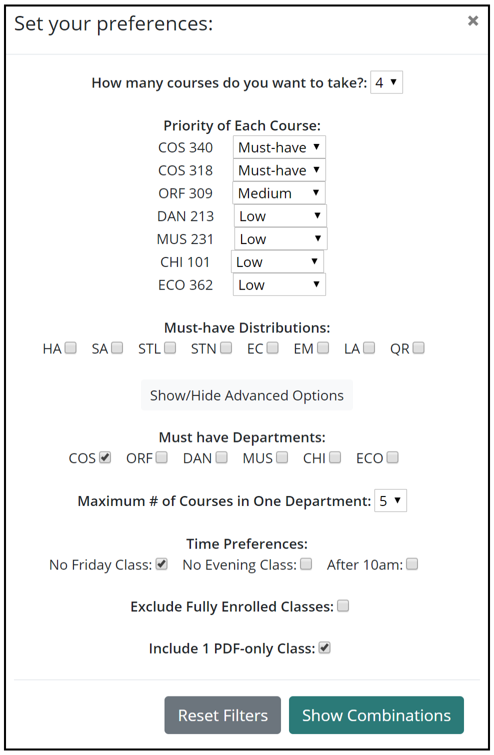
SETTING YOUR PREFERENCES

"Create My Schedule" opens a pop-up window with a list of filters. CourseComb provides several filtering options to find your perfect schedule.

First, you must choose how many courses you are planning to take next semester. You can choose any number from 1 to 9.

Next, you can set the priority of each course as must-have, medium or low. For example, if you absolutely need to take COS 340 and COS 318, you can set them as "Must-have"s . You heard good reviews about ORF 309, but could live with taking it later, so you can mark it as "Medium." The rest are just courses of interest, so you select their priorities as "Low."

Then there are options to select Must-Have **Distributions** and **Departments**. If you select these options,



Set your preferences:

How many courses do you want to take?: 4

Priority of Each Course:

COS 340	Must-have
COS 318	Must-have
ORF 309	Medium
DAN 213	Low
MUS 231	Low
CHI 101	Low
ECO 362	Low

Must-have Distributions: HA ☐ SA ☐ STL ☐ STN ☐ EC ☐ EM ☐ LA ☐ QR ☐

Must have Departments: COS ☑ ORF ☐ DAN ☐ MUS ☐ CHI ☐ ECO ☐

Maximum # of Courses in One Department: 5

Time Preferences: No Friday Class: ☑ No Evening Class: ☐ After 10am: ☐

Exclude Fully Enrolled Classes: ☐

Include 1 PDF-only Class: ☑

course combinations that do not contain the selected distributions or departments will be excluded.

The “Maximum # of Courses in One Department” is a way to limit the number of departmentals in your course schedule. For example, if you have 4 departmentals on your courses of interest, but want to limit the number of departmentals you take to 2, you can set this field as 2.

We also include the option to set **Time Preferences**. For people who like to sleep in, you can restrict the course schedules outputted by our algorithm to exclude any early morning classes before 10am. For those who like extra long weekends, you can set the no friday class filter. No evening class field excludes any classes after 7pm.

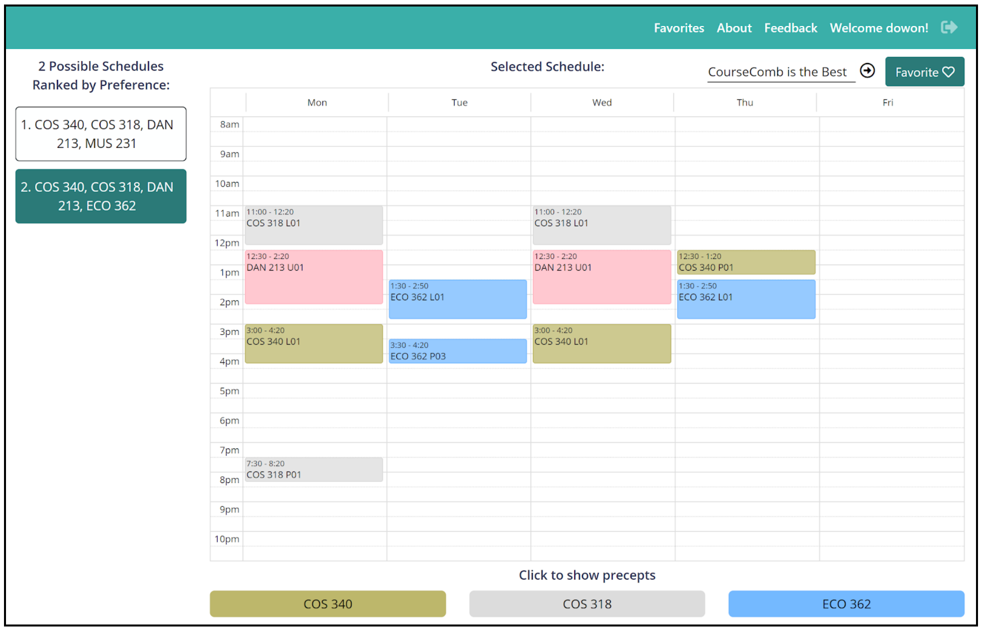
Finally, if you want to make sure you survive the next semester in one piece, you can check the “Include 1 PDF-only Class” option to do so.

SELECTING COMBINATIONS

Once you finalize your filter preferences, press “Show Combinations” to generate all possible course combinations. We *automatically* eliminate all schedules with time conflicts and those that do not satisfy your set preferences.

In the figure below, we have 2 possible combinations, which are ranked according to the “Must-Have,” “Medium,” and “Low” preferences. This way, CourseComb removes the hassle of manually sorting out class time conflicts and gives you exactly what you need!

Selecting schedules on the left will display the schedule on the calendar. Precepts are initially hidden, so as to not clutter the view. You can click each class name on the bottom to display/hide the precept options and select one that you like!



2 Possible Schedules Ranked by Preference:

1. COS 340, COS 318, DAN 213, MUS 231
2. COS 340, COS 318, DAN 213, ECO 362

Selected Schedule: CourseComb is the Best

Mon	Tue	Wed	Thu	Fri
11:00 - 12:20 COS 318 L01	1:30 - 2:50 ECO 362 L01	11:00 - 12:20 COS 318 L01	12:30 - 1:20 COS 340 P01	
12:30 - 2:20 DAN 213 U01	3:30 - 4:20 ECO 362 P03	12:30 - 2:20 DAN 213 U01	1:30 - 2:50 ECO 362 L01	
3:00 - 4:20 COS 340 L01		3:00 - 4:20 COS 340 L01		
7:30 - 8:20 COS 318 P01				

FAVORITING A SCHEDULE

Once you finalize all the precept selections and are happy with the result, you can save this schedule using the “Favorite” button on the top right. After naming the schedule, press the right-arrow to save the schedule. It can be accessed in the **Favorites** tab on the navigation bar.

CourseComb Developer Guide

BACK-END SYSTEMS OVERVIEW

CourseComb uses a PostgreSQL database to store course and user data and Python to process data. We tried to make best use of operations that are made easier using Django, such as basic database querying, updating and saving.

The website is hosted on Heroku. We make use of the Heroku Scheduler to automatically scrape the registrar page every day to update course information. During course selection period, the frequency of scraping can be ramped up to every 10 minutes to constantly update enrollment data for every class.

DATABASE STRUCTURE

Relevant Code: courses/models.py, courses/scrape_import.py

The database can be largely divided into course data and user data. Course data is information on all courses on the registrar page that is needed in our application. User data is information that needs to be stored to remember users’ previous input. This information is displayed to users when they return to the website.

models.py is a file that is required by Django to outline the structure of the database. scrape_import.py is takes the data scraped from the registrar page and saves it to the database.

1. Course Data

There are two tables, ‘Course’ and ‘Meeting’, that store data for each course. Each course on the registrar page has a Course table. The registrar_id field is unique for each course and therefore is used to identify the course when querying the database. Other fields on the Course table are basic information needed in our website, such as the course title, number, distribution area and link to registrar page.

The Meeting table represents a course’s class meetings such as a lecture or a precept. A course can have multiple meetings in a week, but a class meeting is linked to one unique course. Therefore, a Course table has a one-to-many relationship with one or more Meeting tables. The Meeting table stores the day and time of the class, section number, enrollment number, enrollment limit and a is_primary boolean field to indicate whether the meeting is a 'primary meeting'. A primary meeting is defined as a section that is either labelled as L, C, S or U. This field is very important in determining whether two courses have time conflicts, since most primary meetings cannot be moved or switched.

2. User Data

When a user logs in using CAS, a user in Django is automatically created. A 'Profile' table that is created for each user has a one-to-one relationship with the Django user created after CAS login. The Profile table also stores all the courses the users add to their course queue as a String of registrar ids.

The last filter setting which the user inputs is saved to the database. So, the profile of a user has a one-to-one relationship with the user's filter in the database. The 'Filter' table has a field for each part of the filter on the website, e.g. number of courses in a course combination, must have courses, must have departments.

The 'Combination' table stores course combinations that have been calculated by our algorithm. A user normally has more than one course combination that is generated, and so there is a one-to-many relationship between the user profile and combinations. There is a field that stores a String of the registrar ids of the courses, which is used to query the database when information about the courses is needed. There is also a field that stores a String of the titles of the courses and this String is used to display the course combination to the user. Lastly, there is a boolean field 'filtered' which indicates if the course combination was filtered out based on the user's set preferences. If this field is true, the course combination is not displayed to the user.

The 'Favorites' table stores the user's favorited schedules and similarly has a one-to-many relationships with the user profile. The table has 3 fields: name, courses, and favorite_fields. The name field is for the user to name their favorited schedule, a UX decision made to help the user distinguish between their different favorites. The courses field is a text field that contains the courses' names (in the format of department + number) to be displayed on the Favorites tab. The favorite_fields field contains the JSON object of FullCalendar data of the favorited schedule.

SCRAPING AND DATA COLLECTION

Relevant Code: courses/scraper.py, courses/scrape_all.py, courses/scrape_evals.py

We only had to minorly edit ReCourse's scraper code to collect all the course data that we needed. Most of the editing process involved deleting lines of code that were needed for features in ReCourse but not needed in CourseComb.

scrape_all.py calls methods in scraper.py and scraper_import.py to scrape and store data. scraper.py uses Beautiful Soup, a python library that is often used to parse data from web pages. It collects data from the registrar page and returns all the information in JSON format. scrape_evals.py scrapes the course evaluation page to obtain the URL to the evaluation page of a certain course.

GENERATING COMBINATIONS & FILTERING

Relevant Code: courses/combination.py, courses/courses_filter.py

The process of generating course combinations is separated into two steps: 1. generate all combinations of given courses excluding any with time conflicts, 2. Filter out any of the combinations that do not fit the user's preferences.

combination.py performs the first step of this process. A method in the code takes an array of all the courses in the user's course queue and the number of courses the user wants to take. It returns all possible combinations of the given courses excluding any with time conflicts.

courses_filter.py performs the filtering of course combinations. It implements a brute force method of going through each combination and checking if the combination satisfies each field of the user's filters. This appears to be the performance bottleneck of our application.

FRONT-END SYSTEMS

Relevant Code: templates folder, static folder

The front-end design is primarily created through the use of Bootstrap, FullCalendar UI, and jQuery UI (along with some simple JavaScript and CSS tools for styling and responsive UI).

1. **Bootstrap 4 (getbootstrap.com)**

   Similar to other Princeton course scheduling apps in the past, CourseComb was designed for target users to access our application on their desktop or laptops. Using Bootstrap's grid system, we were able produce a responsive layout for users in cases where they view our website fullscreen or smaller, if they needed to multitask with their schedule on hand.

2. **FullCalendar API (fullcalendar.io)**

   As our app was first and foremost a course scheduling app, we decided to use an existing and well-used API to display our combination results. FullCalendar API offered a clear API and a comprehensive set of tools - from saving events in a state to dynamic changes in visuals resulting from user input -for displaying our combination schedules dynamically.

3. **jQuery (jquery.com)**

   jQuery was very prevalent in our website - event selectors and responsive animation always involved using jQuery. Also, we used the jQuery Autocomplete UI for implementing our course search, which allowed for direct communication with our regularly updated database.

FRONT-END BACK-END INTEGRATION

Relevant Code: courses/views.py, static/js/script.js, static/js/script_favorites.js

We used AJAX to send information from the front-end to the back-end in four events: modifying the coursequeue, creating course combinations from user's coursequeue and filters, displaying events on calendar when user clicks on a course combination, and favoriting a schedule.

1. Modifying the coursequeue: when the user adds or deletes a course from the coursequeue, the selected course's registrar_id is sent through a POST request to modify the user's coursequeue in the database.
2. Creating course combinations: when the user sets the filter and clicks on "Show Combinations," the filter data and a JSON object of registrar_id's of courses in the coursequeue is sent through a POST request to the back-end. The filtering algorithm does the work and sends back either a JSON object of course combinations or an error message if no combinations can be created.
3. Displaying events on calendar: when the user clicks on a course combination in the second column, a JSON object of the selected course combination is sent through a GET request to the back-end, which returns the meetings data of the combination.
4. Favoriting a schedule: after the user names their favorited schedule, a JSON object of the events of the selected schedule is sent through a POST request and saved in the database so that it could be displayed in the Favorites tab.