ClassMaps Product Guide

Spring 2018

Team

Adrian Tong (PM/Front-end)
David Todd (Back-end)
Andy Lin (Front-end)
Yang Song (Front-end)
Jerry Wei (Advisor)

User Tutorial

Overview

We have created a new webapp for Princeton with a fundamental feature that had not been done before: mapping courses and other resources, such as printers, laundry, and filtered water, directly to an interactive map.

Login/Setup

ClassMaps is a webapp, so it can be reached directly at http://classmaps.herokuapp.com/ on either a computer or mobile. After duo authenticating through CAS, you will be taken directly to the main page.

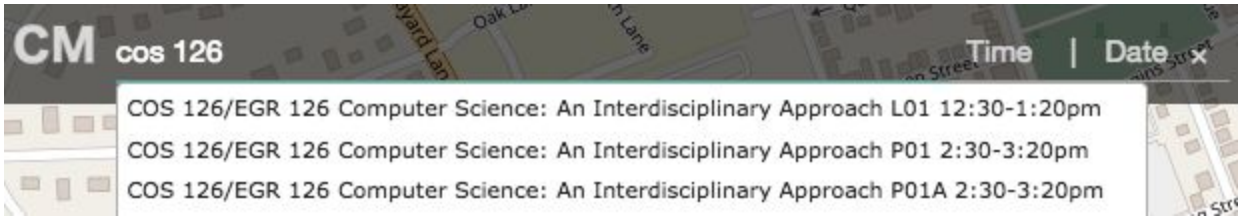
Features

Home Button

Clicking the “CM” logo on the top left will reset all search queries and take you to back to the landing page.

Basic Search

Imagine you’re a freshman who has just signed up for fall classes. Logging into Tigerhub to check out where your COS 126 lecture is located, the word “Friend Center” means nothing to you. Instead, you open up ClassMaps.



ClassMaps offers an intuitive interface, allowing you to look up your classes using the search bar at the top and then select an option to display on the map. Searching “COS 126” and selecting the lecture “L01” using the autocomplete feature maps you exactly to where the lecture meets by zooming to the location on campus and giving a “popup” showing detailed information about the course. This gives a quick and easy solution to finding where your class is located on campus.

More generally speaking, a search query can match the following: department, number, and the building. Our search bar also supports regular expressions, and you can join multiple queries with commas. If there is a marker at your result already, it will be slightly offset so you can see both markers.

Autocomplete and Multiple Results

If you did the action above, you will have noticed the autocomplete feature showing possible results as you entered the search query. If there are multiple results, the results will display horizontally (example search: “cos 340”). You can scroll through them by swiping sideways on trackpad. The first card will tell you details about your search queries and the number of results that it matched with.

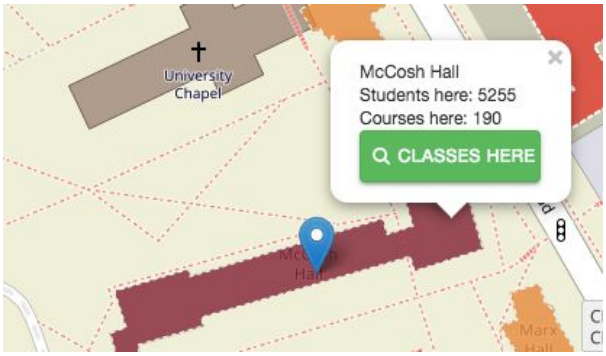
Alias Searching

One of your friends who shares a class with you tells you that the class meets in the Woodrow Wilson Building, which you have no idea is actually called Robertson Hall. You search, and you not only get all the results you wanted, but learned the building’s actual name.

Using Filters

Now, you want to find out the location of your COS 126 precept. Being the incompetent freshman that you are, you’re unsure of the exact precept number. However, like most people, you do remember the time and the day of the precept.

The search bar has two filters to help narrow results: one filter is by time and the other is by date. The time filter is an interactive clock that lets you select hour and minute fields as well as AM/PM before submitting. The day filter is a drop-down checklist of boxes that allows you to check which day(s) you want. Selecting multiple will “OR” the results to show the most results possible. Since you know the time and day of your precept, you successfully filter out other precepts and find the exact one you belong to. Clicking the “x” button at the end of the search bar will clear all search queries.



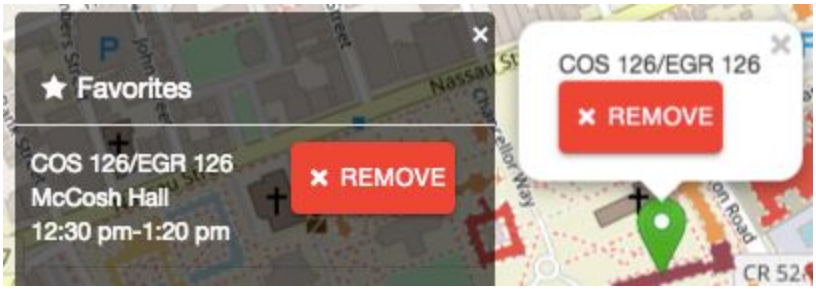
Search by building

There are building outlines around every dorm and building with a course in it. If you hover over they will highlight to help you identify them. Clicking on this outline will open a popup with a button that says “Classes Here.” Click on it to search for classes in that building. If you have applied

time and date filters, it will still work with this search button.

Saving Classes

This is neat and all, but since you're generally a forgetful person, how will you remember which class is in which building is if you close the app and forget everything?

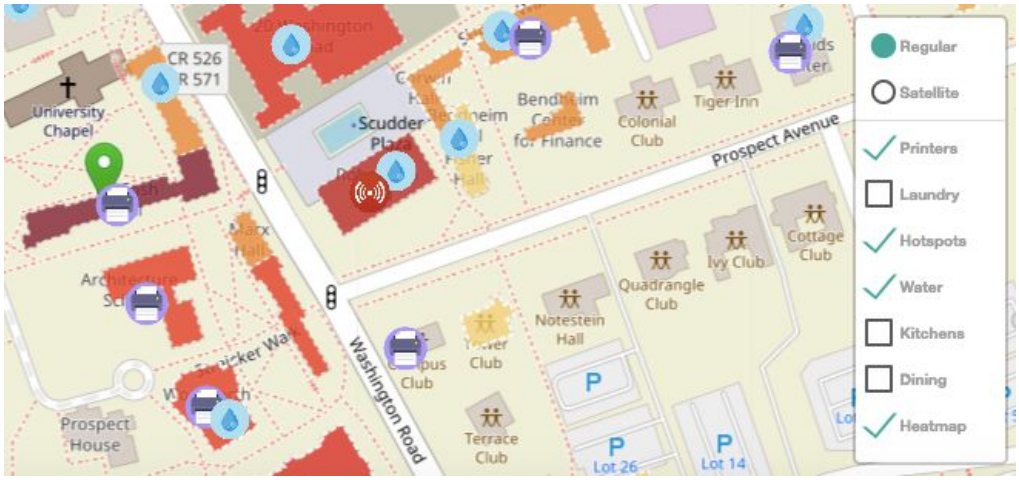


The "popup" after a successful search will give the user the option to "Favorite" a course, which will save it to the favorites list in the bottom left corner. You can now open up ClassMaps on your mobile device and see the location of every class you've saved while out and about. You can also remove favorited classes, which is helpful, considering you just dropped Spanish 101 from your schedule and no longer need to know the location of it. You can also navigate to classes directly from the favorites menu. Selecting a favorites item will automatically set the map zoom to that location. If you save multiple courses in the same location, they will display as a scrollable list in the popup.

Additional Features:

Resource Overlays

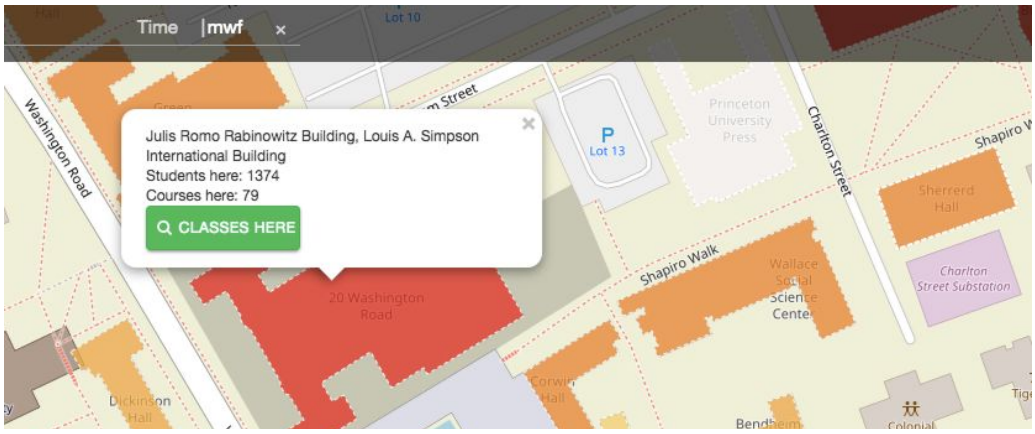
At this point, you're wondering, "What else can this map do?" We're glad you asked.



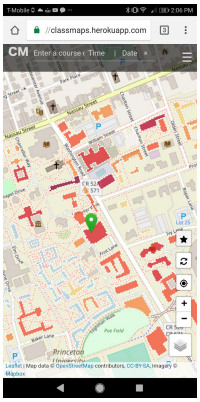
The arrangement of buttons on the bottom right corner of the map allow for auxiliary features, such as finding the buildings where laundry machines, printers, and hotspots are located; changing the map to satellite view; closing the Favorites tab, zooming; and finding your current location. If there are multiple resources in the same location, it will list the results in the same popup for convenience.

Satellite View
There is a toggle for satellite view at the top of the overlays box. You can still put all the same overlays, heatmap, etc. on top the satellite view also.

HeatMap
We know you're probably interested in what this heatmap feature is as well. By default, the colors you see on the map correspond to the number of students in each building based on enrollment numbers in every course summed over the entire week. The light blue color represents the dormitories. When you hover or select on a building, it will display the number of courses and number of students hosted in the building. Applying the date and time filters will dynamically update the map and recalculate the number of number of students and courses in each building.



Mobile-Friendly
We anticipate that users will use ClassMaps on the go, so we made it [mobile compatible](). You will see that the favorites bar is hidden by default, and the overlay controls are collapsed into an icon. You can toggle the favorites with the "star" button on the right. The mobile version retains all of the same functionality as the web version. You can add ClassMaps to your home screen on mobile through the browser and our app icon will appear.



Menubar
The menu icon at the top right corner will open up the menu bar from the right side. You will see multiple options, including the ability to give feedback, and to learn more about our team and project. The bottom of the menu bar has instructions for a user to add the app to their home screen on mobile.

Developer Guide

Frontend (classes/templates/classes/*)

Leaflet

Leaflet is an open-source Javascript library for mobile-friendly interactive maps. We decided to use Leaflet because it was powerful yet simple and very customizable. Leaflet also allowed us to mark GeoJSON objects, have different vector layers and image overlays as well as interactive markers and popups. Other past projects, such as the TigerApps Rooms website, also used Leaflet because of its functionality.

Bootstrap (classes/static/classes/dist/)

We used bootstrap package to create the clockpicker. We changed it to be a 12 hour clock, and it applies "AM" by default if AM/PM is not selected, since that is when the majority of classes are. There are dependencies for it in our static folder. We also used a bootstrap template to create our about page.

HTML

All of the elements you see on top of the map are created as overlays, since the map is full screen. The search bar at the top manages the queries and filters through a form that is submitted to the Django framework. The results are displayed as card divs, with one for each result.

Backend Systems

Scraping (scraping/*)

We modified the script written by Alex Ogier '13 (scraping/scraper.py) and maintained by Professor Kernighan to scrape the course listings provided by the registrar. In addition to getting departments, course numbers, meeting times, and the usual fields needed to identify courses, we also found the unique building id number associated with the class's location. We then scraped the longitude, latitude, canonical name, and aliases for each building on Princeton's campus (organized by the id number discussed above) from OIT's WebFeeds (getbuildids.py). From here, we linked the two datasets by comparing the building id numbers or approximately matching by name in the few cases id numbers were not provided by the registrar (mainly for newer buildings) and serialized them into forms that could be added to our database (merge.py).

We also scraped locations for resources on the OIT's WebFeeds page. We did this through parsing XML data into GeoJSON objects, which were then mapped to various layers on the Leaflet map. For each of these we stored the type of resource (e.g. printer, kitchen, laundry room) along with its coordinates, exact location within the building (e.g. Entry 1 basement Room 005) and the number of features present.

Lastly, we used OpenStreetMap's API to obtain map data for Princeton's campus. After obtaining the raw data, we used QGIS3 to extract the building outlines as GeoJSON objects. We then filtered the outlines to include only buildings that were tagged as dormitories or had at least one class to remove unnecessary clutter from the map.

Database Structure (classes/models.py)

To store our data, we defined 3 tables. A Section holds all the information that defines a class meeting including the listing on the registrar, the time, and the location. A Building holds the longitude, latitude, and names of every building on campus. Sections and Buildings had a many-to-one relationship (using the ForeignKey functionality in Django) so that several sections could all have the same building. Each class, though, still separately stores its building name to speed up the calculations needed to compute the heatmap. Looking up the name after looking up the building for each class just took too long. Lastly, we maintained a table for users that tracks their saved classes and saved buildings. While we could have defined a many-to-many relationship between Users and Sections (as well as Users and Buildings) we felt this was unnecessarily complicated and might become difficult to maintain when we have to rescrape and update course enrollments throughout the year.

Searching Algorithm (classes/views.py)

Our search function defined in views.py takes in any query entered by the user, let's say, "COS 333". The query is separated by spaces and each term (COS and 333) is used as a filter through all of the classes in the database. We used Django's built in filter() method with "icontains" and "iregex" to see if the query term matches the start of a word in a course's listing or building ignoring case. This matching strategy ensures queries like "COS" don't match "McCosh Hall" like it would if we used the "icontains" method alone. Since all of the classes are filtered across all terms (first "COS", then "333"), the resulting QuerySet that is returned will contain only classes whose course names matched with "COS" and "333", which would be only COS 333. We also required that each term matched the end of a word, except for the last query term, which the user have left incomplete. This means the query "2 dickinso" matches only the co-op building the user intended rather than "HIS 267", which happens to be in Dickinson Hall with a course number that begins with 2. Because some outlines encompass multiple buildings on Princeton's campus and we want to find classes that are in any of the buildings when searching by outline, we also added the functionality where queries separated by commas are split apart and the resulting QuerySets OR'ed together. The search bar also does not recognize the title of the class since we assume people memorize only the department and course number.

As for day/time filters, our search method also takes in queries that are passed by the time clock picker and day checkboxes, say "11:30AM" and "Th". These queries are used to filter out the resulting QuerySet, which only contains COS 333 in it, using day__icontains="Th" and "gte"/"lte" to get all the classes that meet on Thursday and have a

time with a range that contains 11:30AM. Since COS 333 meets on TTh 11:00AM-12:20PM, it passes through both filters and the resulting QuerySet still contains COS 333.

Bridging the Front and Back Ends (index.html/views.py)

Django

Users searching for courses and buildings, whether by typing into the search bar, selecting the time/day, or clicking on a button in a building outline popup, were processed as a GET request and translated into appropriate database calls using Django and the searching algorithm described above. A synchronous response was then returned as either a single marker on the map or a list of matching courses.

jQuery/AJAX

User text typed into the search bar was retrieved via AJAX and used to query the database with the same mechanisms as general searching. The asynchronous response was received as a JSON object and used to create the autocomplete functionality. The heatmap was handled similarly, with changes to the time and day selectors triggering calls. Updating user saved classes and buildings were handled as POST requests with AJAX when the user pressed a save or remove button as well.

• • •