Spectacle

Uplanner

Spring 2018

Overview:

Our main idea was to create an application that UMass Amherst students can use to build their semester schedules. Spire.Umass.edu requires students to enroll in courses before they can see what their weekly schedule will look like. We have students search for courses using our application, and use those results to build their own weekly schedule for each semester all in one window. The schedule has a similar appearance to Spire's weekly schedule, but ours is dynamic so students can build and experiment with different course combinations to see what works.

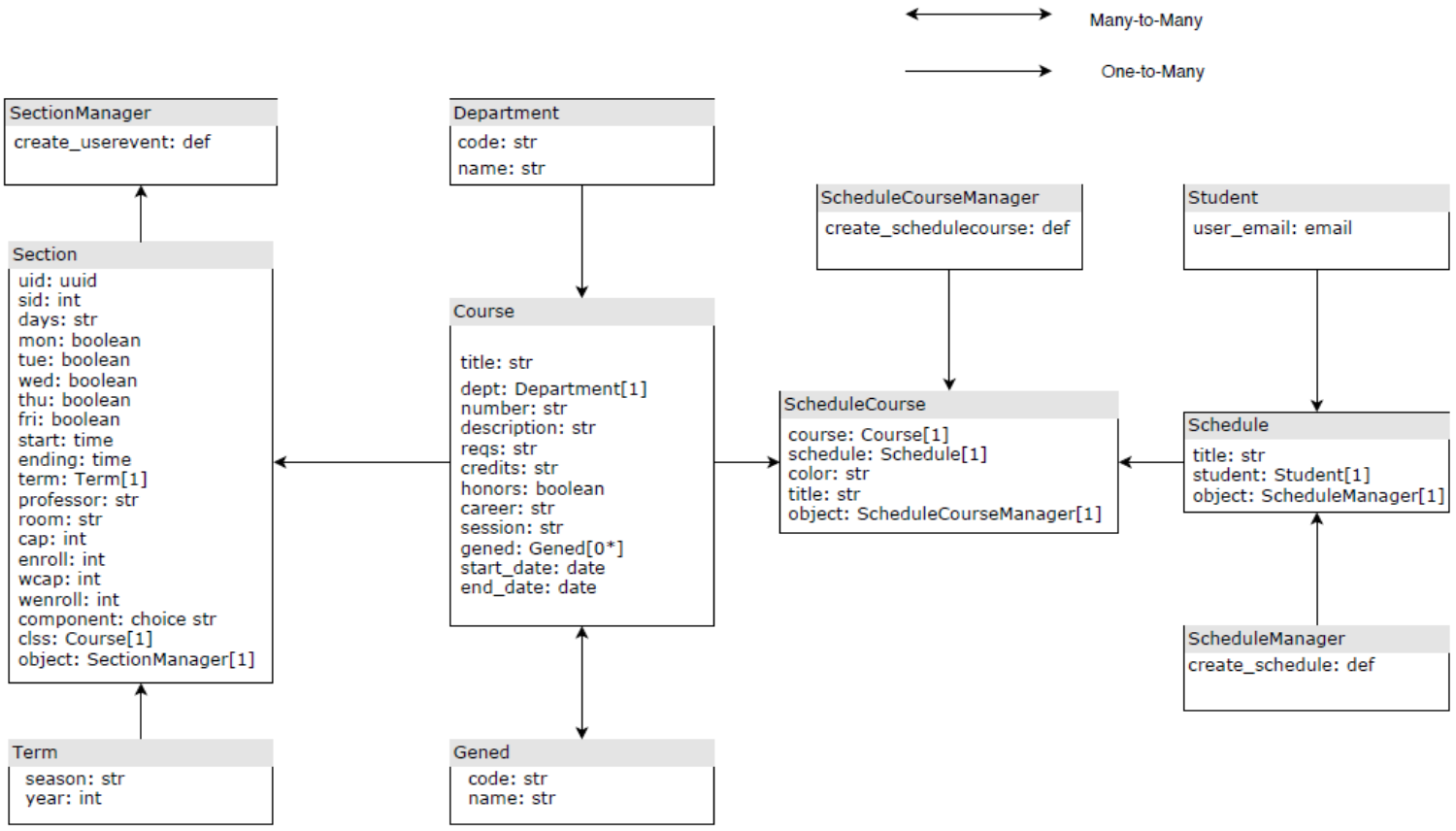
In addition to the schedule builder, we also provide students with a dynamic flowchart of prerequisite courses for each major. Students are able to plan for future semesters by seeing the courses they need to take before they can enroll in particular courses of their choice. From the schedule page, users are able to view full course and section details in a new page by double clicking on a course in the table. Our application also has an admin page for database modifications, and a landing page with information about our application. We have lots of other little features added to make this a great experience for the users, but these are the main parts to our application.

Team Members: Kerry Ngan, Che-Wei Lin, Jason Valladares, Liam Britt, Sihua Chen, and Ben Elliott

Github Repository: https://github.com/frumsy/uplanner/

Data Model:

The diagram below shows the relationship between each of our data model. First we have the model Courses and Sections because a class at UMass Amherst has a course and may have have more than one section for each. There is a one to many relationship from Course to Section. Since we plan scrape new sections from spire every semester, we created a one to many mapping from Term to Section. Each section also has a SectionMangager which contains a method to create a user event. Each course may have more than one Gened if any at all so we created a many-to-many mapping from Course to Gened. A course also contains a department and we created a one to many mapping from department to Courses. All these different models help create a search function based on the models. For example, in our website, students are able to sort classes by Gened or Department. Then we created a model called ScheduleCourse with a one to many mapping from Course to ScheduleCourse. A schedule contains courses with added attributes such as color. ScheduleCourse also contains an object called ScheduleCourseManager which creates the ScheduleCourse in a one to many mapping. There is also a one to many mapping from Schedule to ScheduleCourse. Students are able to make more than one schedule so we created a one to many mapping from Student to Schedule. A schedule also contains a ScheduleManager object which creates the schedule.



URL Routes/Mappings:

Our top-level urls are “admin/”, “uplanner/”, and the Django default “profile/”. The empty string redirects to “uplanner/”, which routes most of our urls. “profile/” is used for login and logout. The profile urls also include pages for resetting passwords, although in practice these don’t do anything and were never fully implemented. The pages can still be visited, however.

Most of our webpages are under “uplanner/”. The main webpages available are “index”, “schedule”, “profile”, and “prereqs”. There is also a “register” url which lets users make an account. There are also two unused urls and corresponding views, “login” and “logout”, which don’t function and are never linked to on our website.

There are a few other miscellaneous urls. Each Course in our database can be viewed in its own webpage using class-based views. The url for this is “prereqs/course/pk”, and calling reverse() on a Course will return its url. Finally, there are many auxiliary urls used by the schedule page. All urls of the form “schedule/ajax/…” are ajax calls for communication between javascript and the client. These play many roles, but often are needed to update the server based on interactions with the page, or to retrieve data from the server and make changes to the page accordingly. The non-ajax views “make_tab_contents”, “make_current_courses”, and “make_current_course” are all used by javascript to selectively reload only a certain part of the page. Every time a new course tab is opened, the

“make_tab_content” url is used to populate the tab. Similarly, the current_course(s) urls are used to render just one or all of the list elements in the “courses” tab.

Authentication/Authorization:

The project uses default User Authentication template and there were no permissions involved in the project. There were decisions to extend User model to Student model; however, it didn’t go well as none of the implementation tutorials worked for us. We did have a workaround for that where we mapped User email field to Student email field to extract Student’s information rather than re-implementation the whole User model.

Afterward, our team decided that it was best for the user to login in order to see the website content. This is fairly obvious since no one other than UMass students should ever see course information(time, location, instructors, etc). Therefore, we require login for all the pages except for index.html. Each of the page checks for Http request.user to see if the user is authenticated, if not, it redirects to login page.

Team Choice:

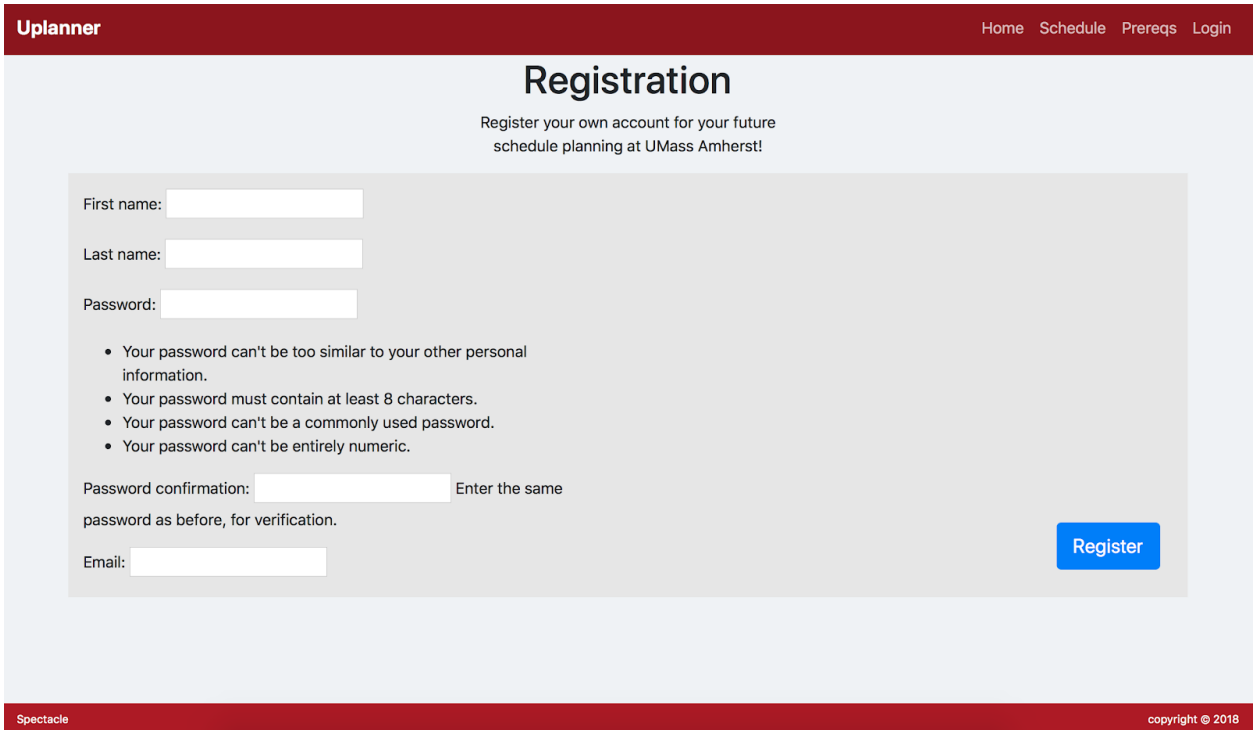
We had a bunch of team choice elements such as using Ajax all over the schedule page of our application, using a third party schedule building tool in our application, and using some Javascript for creating our prerequisite flowchart . We also scraped Spire directly for our data using Selenium and Scrapy.

User Interface:

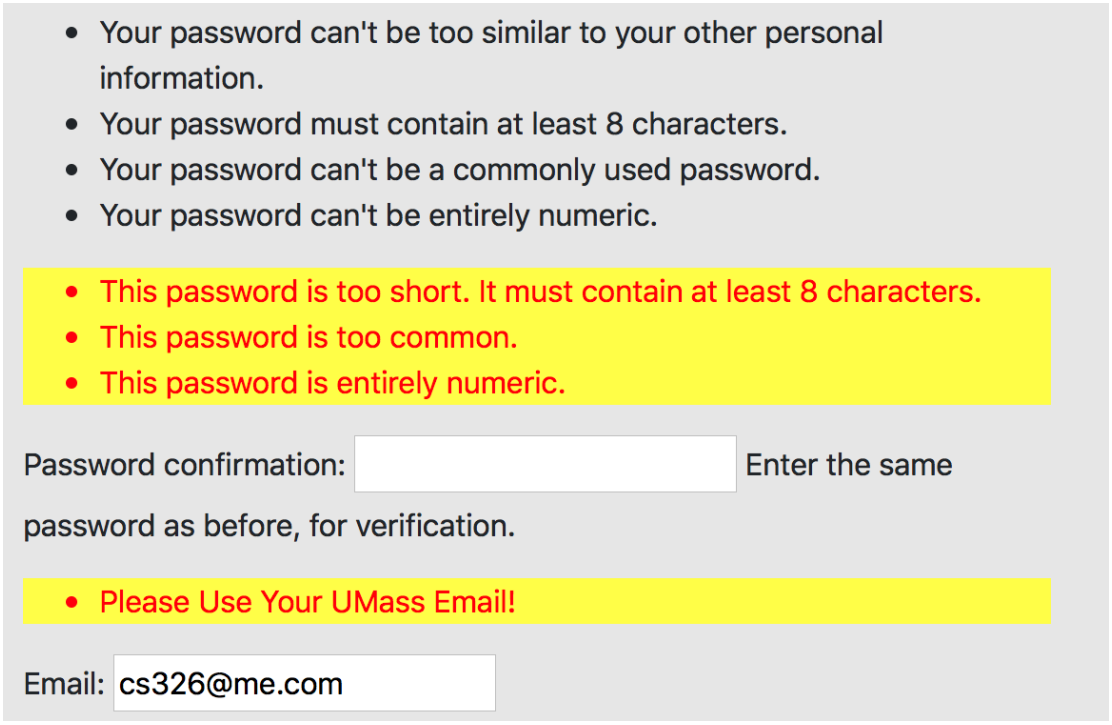
1. Index page (start page of our application)

   a. Button: “Get Started” -- Click this button to nav to Registration page(2).
   b. Nav Bar: “Home”/ “Schedule”/ “Prereqs”/ “Login” -- User has to login to access our web, otherwise all nav bar buttons will nav to Login page(2)

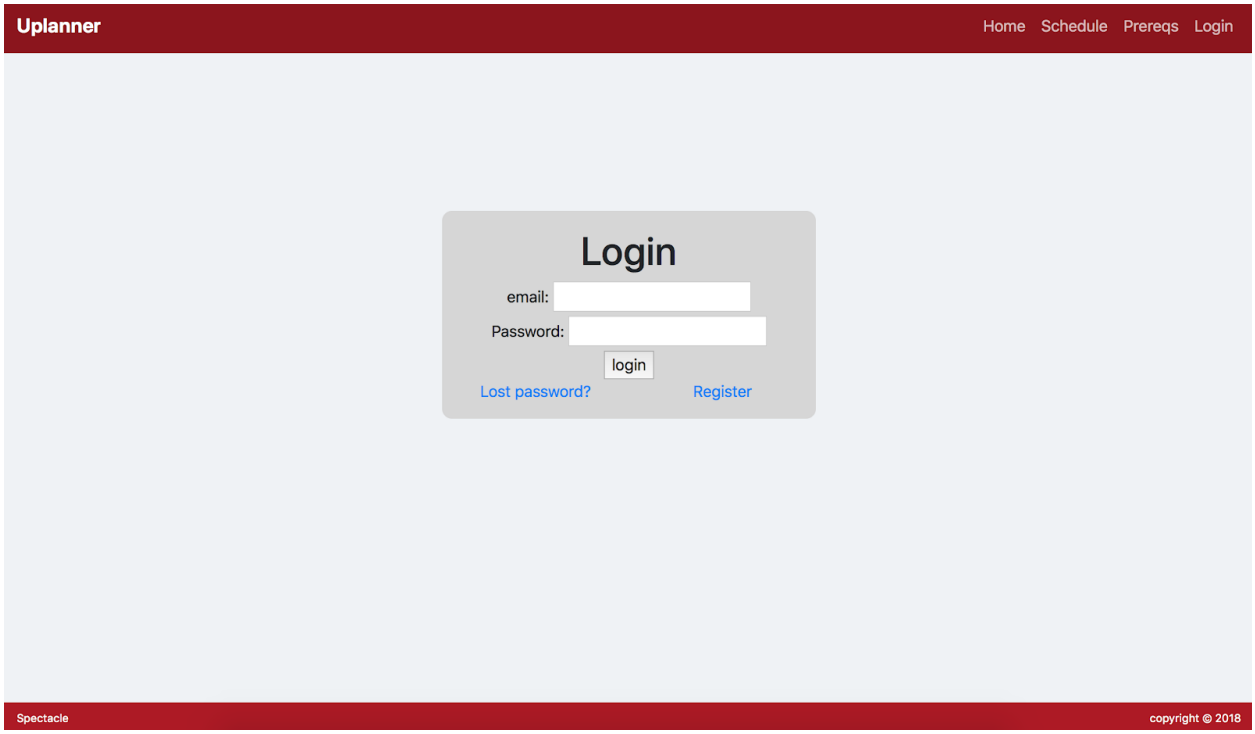
2. Registration page(for user to registrate)



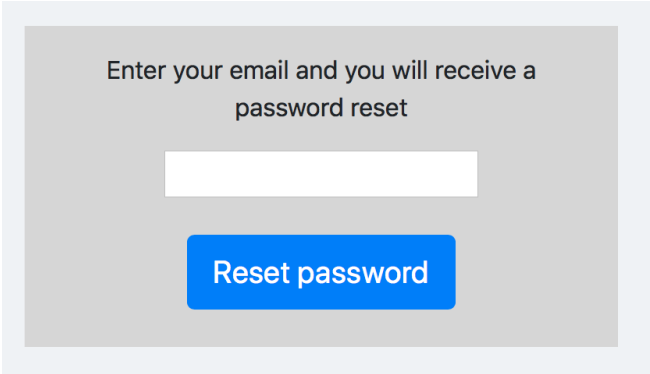
a. Button: “Register” -- This button will register user in and if the password does not meet the requirements, system will return alert.



3. Login page

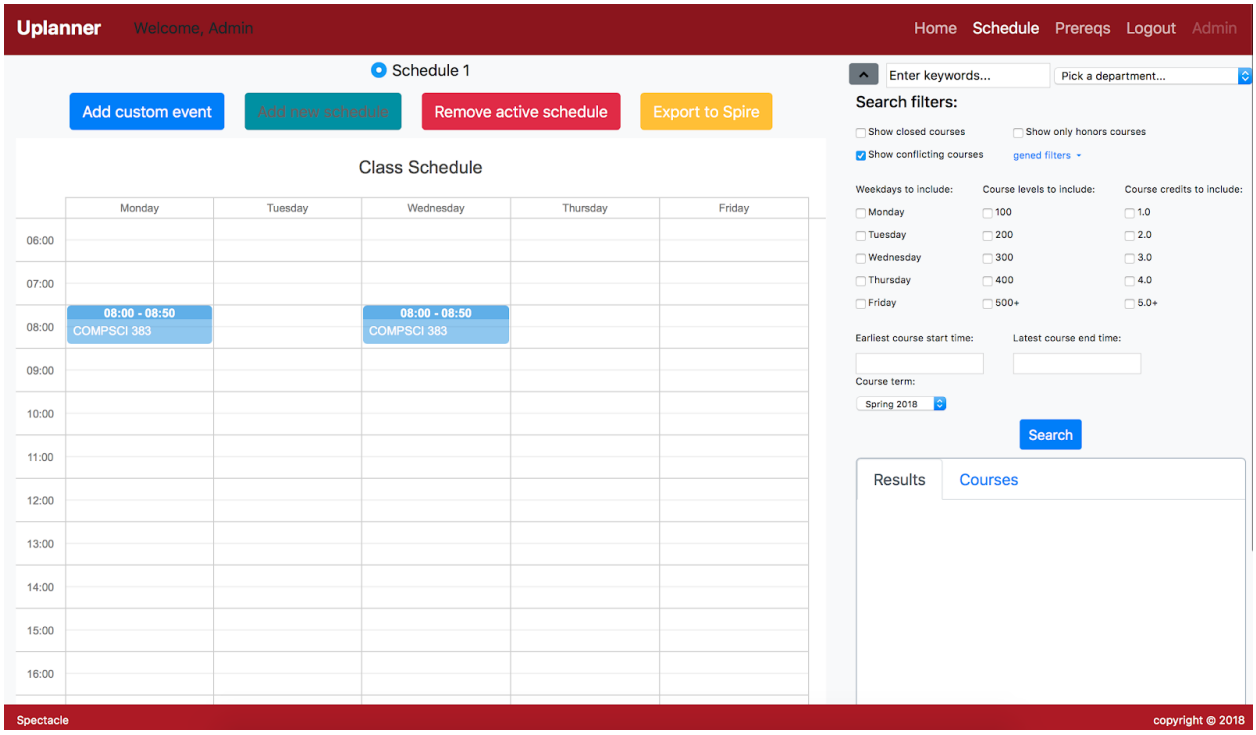


a. Button: “Lost password?” -- It will send reset password link to your email.

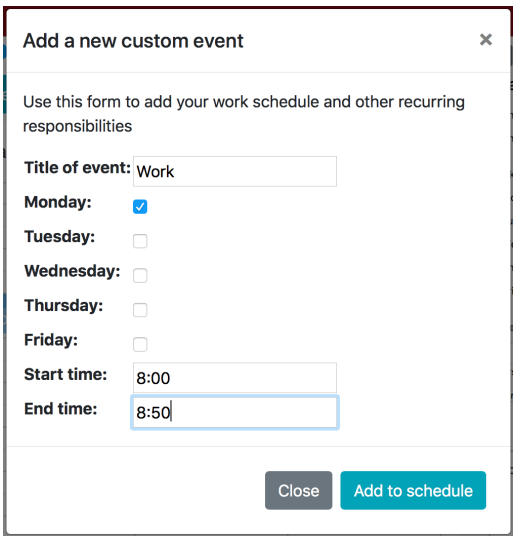


b. Button: “Register” -- Same function as 2.a

4. Schedule page (Our main function page, schedules for user to arrange courses)

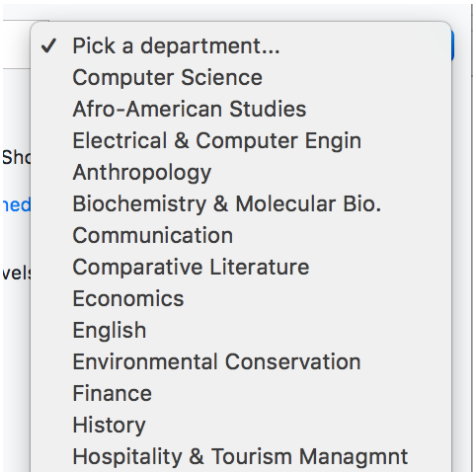


a. Button: “Add custom event” -- Allow user to make their own event for arranging the schedule.



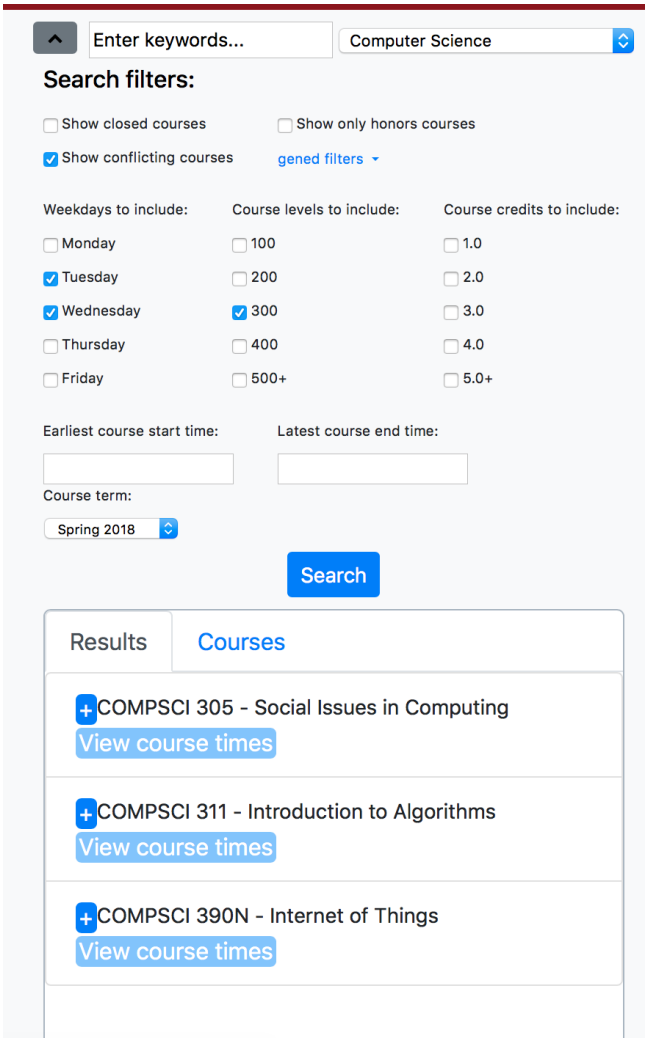
b. Button: “Add new schedule” -- User can add multi schedules to compare and back-up plans.
c. Button: “Remove active schedule” -- User can delete schedule from the page and database.
d. Button: “Export to Spire” -- Feature Requirement: User can export the schedule to Spire for enrollment.
e. Input text: “Enter the keywords” -- User can search by keywords

f. Selection: “Pick a department” -- User can pick a department for filtering courses.

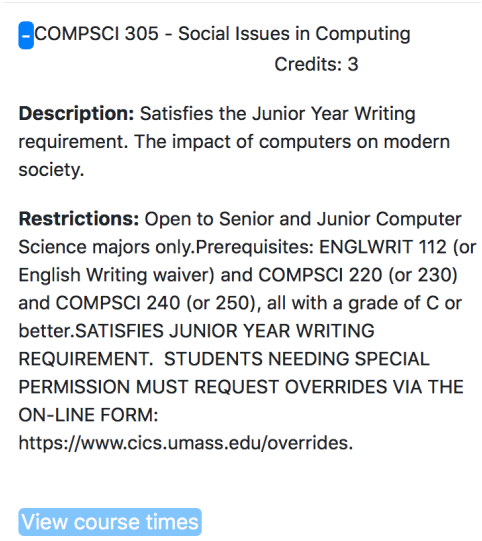


g. Search Div (Checkboxes) -- User can search and filter courses according to time, gened, course level, course credit and course term. All search functions are implemented.

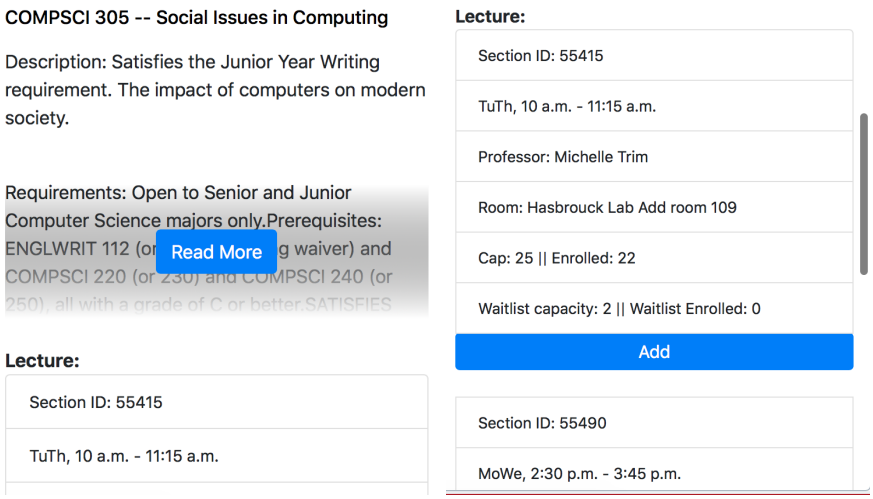
h. Button: “Search” -- Click to search courses according to selected keys in UI 4.e, 4.f and 4.g.



i. Tab: “Results” -- Click to show search results.
j. Button: “+” -- Click to show course detail.

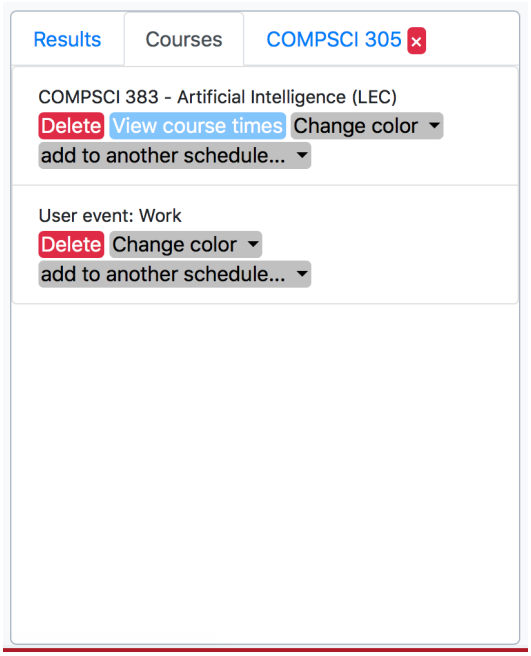


k. Button: “View course times” -- Click to add a new table to show user the course’ sections.

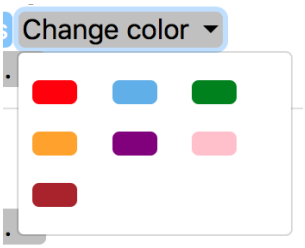


l. Button: “Add” -- Click to add this section to schedule.

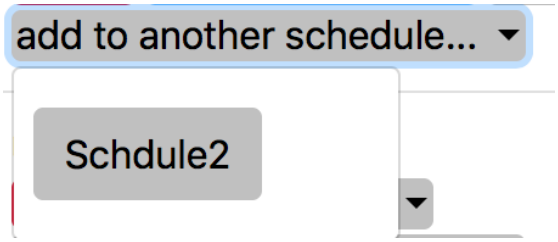
m. Tabe: “Courses” -- Show the courses info in schedule. Mouse over the courses will highlight the courses in schedule



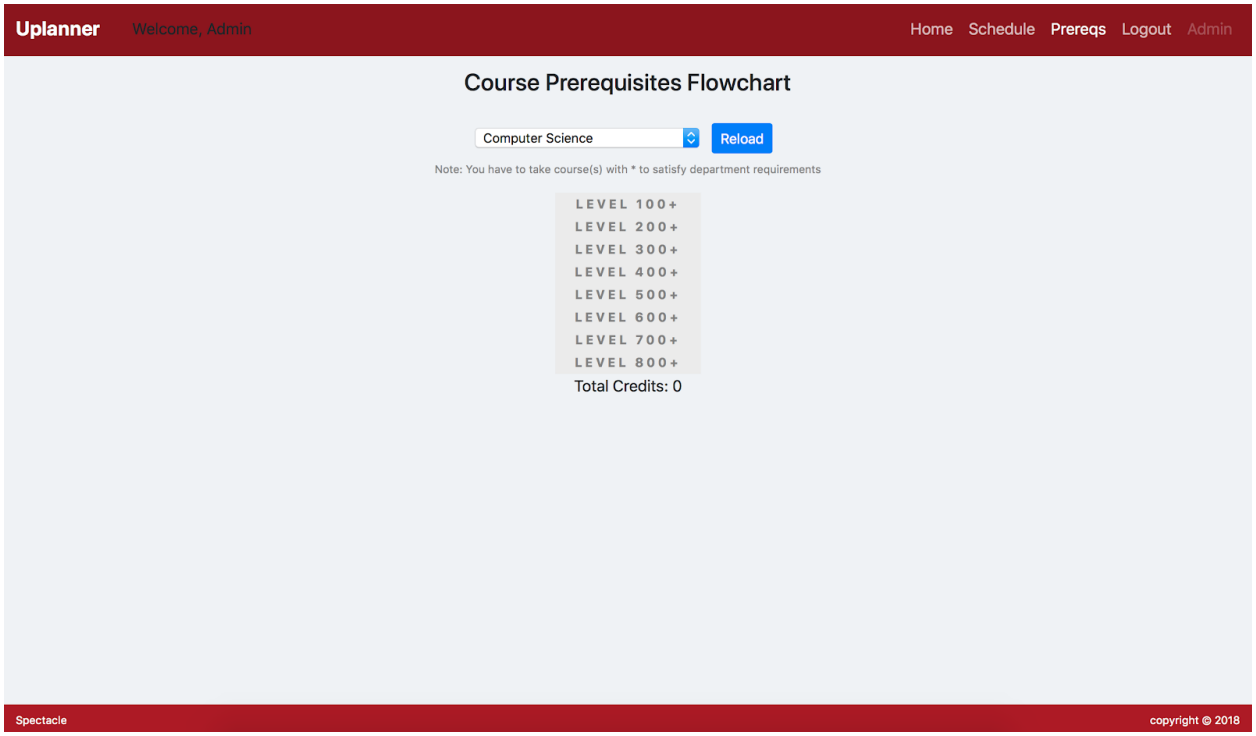
n. Button: “Change color” -- Change the block color of course in schedule.



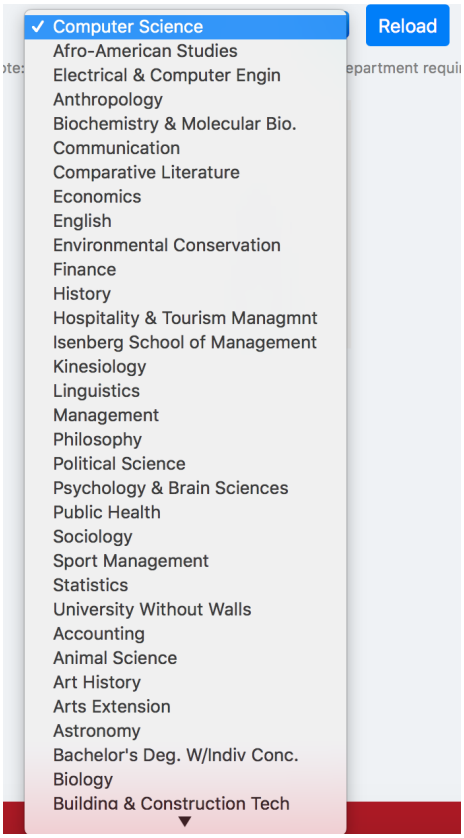
o. Button: “add to another schedule” -- Add this course to another shedule.



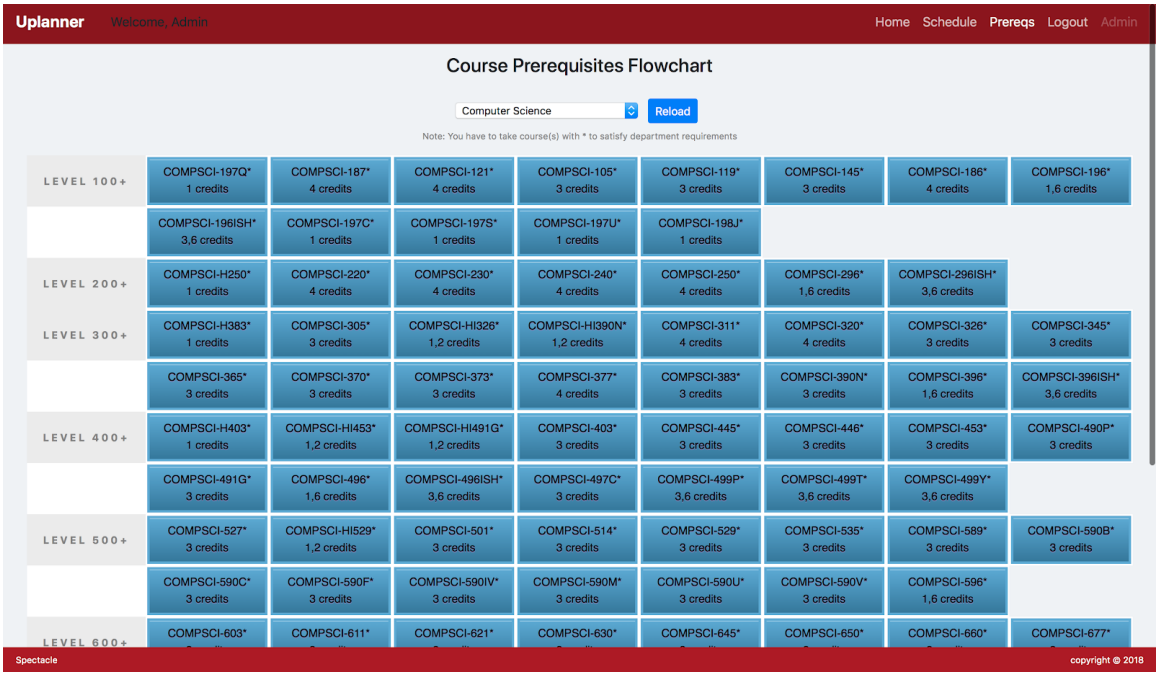
5. Prereqs page (User can check all prereqs in our flowchart to arrange schedule better)



a. Scrolling down: “Computer Science” -- Default as Computer Science but user can scroll down for other majors.



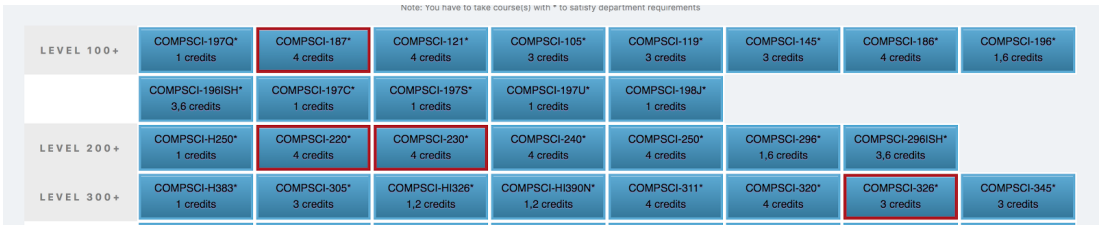
b. Button: “Reload” -- After selecting major, click to generate the flow chart.



c. Button: “Courses” -- Every course is a button.

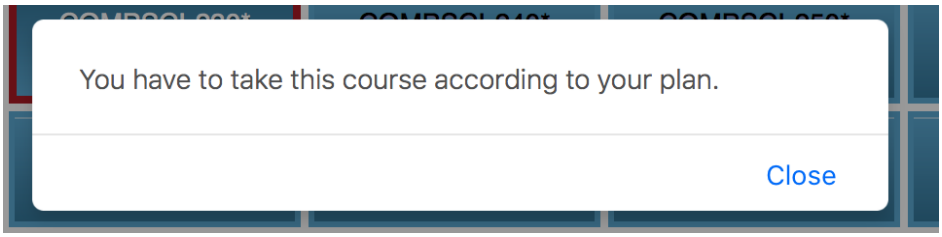
i. The button will show the course number and credits.

ii. Havor: If mouse in one button, system will highlight all prereqs courses. (Havor CompSci 326 and will highlight CompSci 220, 230 and 187)



iii. Double Click: If user wants more info for one courses, double click will nav to Course Detail Page (6).

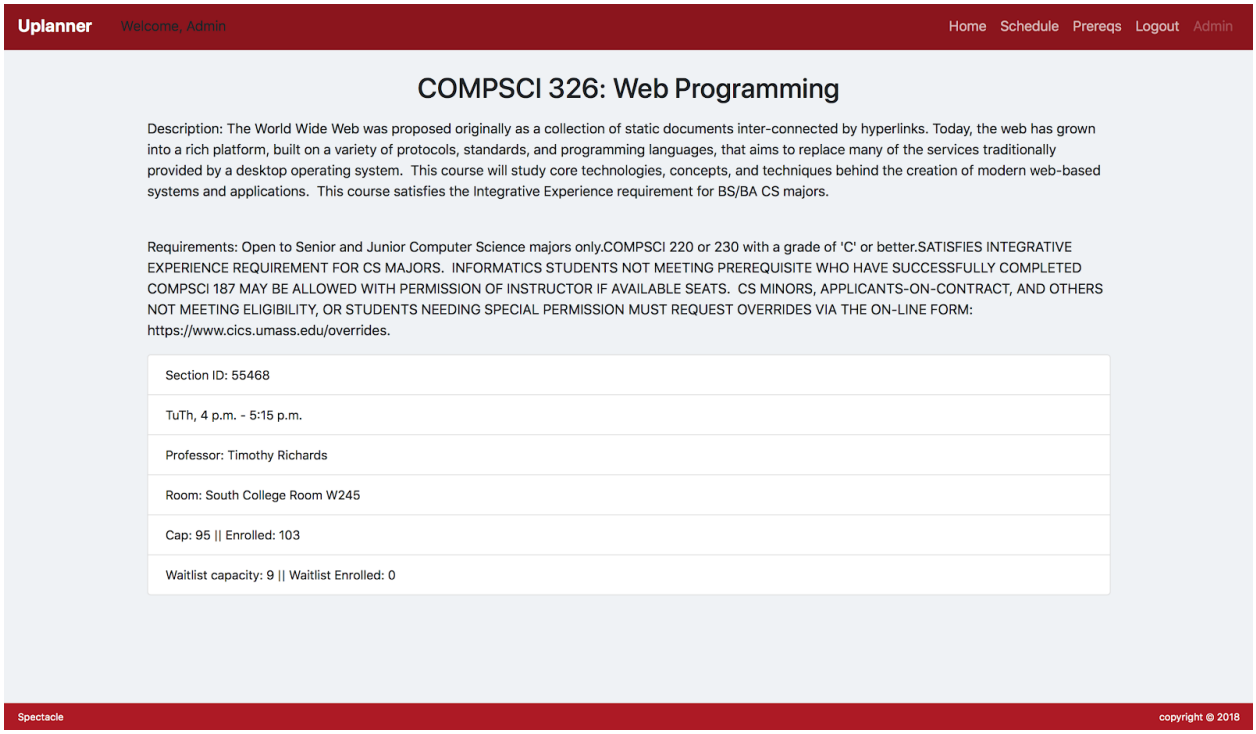
iv. Single Click: Click to select course and system will select all prereqs courses for user. Click selected course to unselect that course. If user wants to unselect a prereq course that are still bounded, system will return alert.



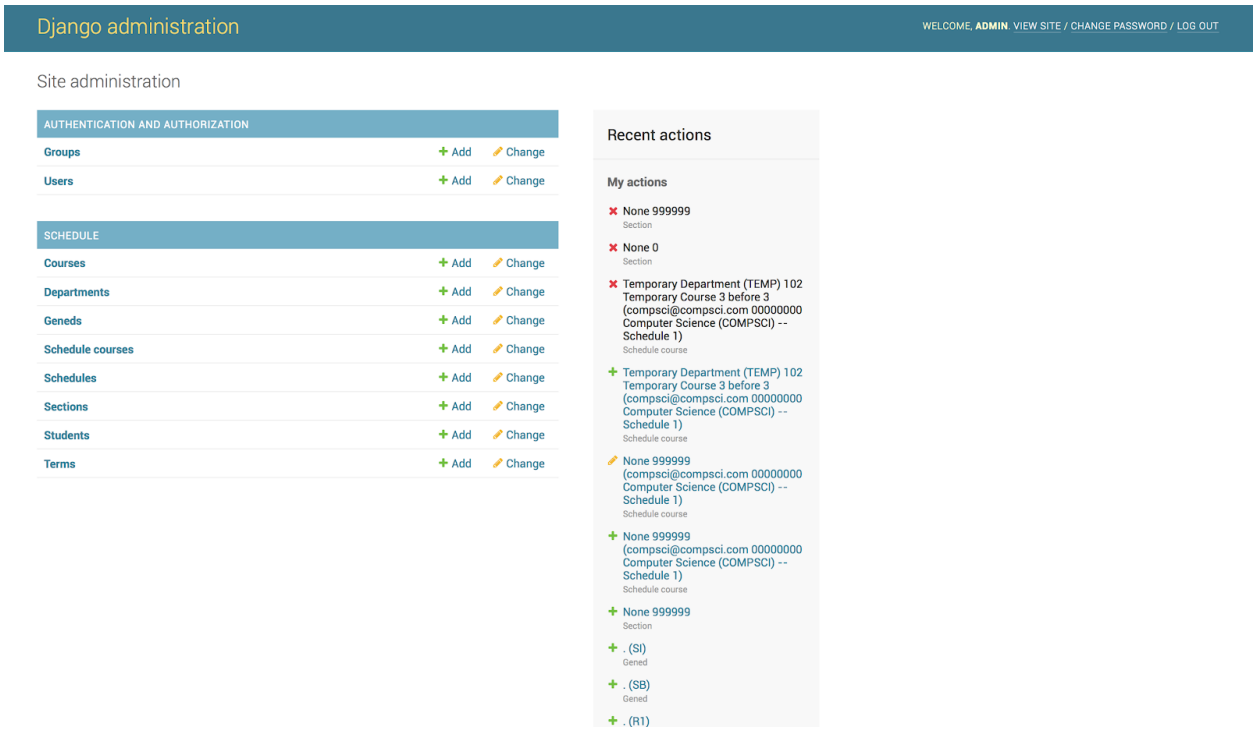
Unselect root course will unbound the prereq course.

d. Div: “Total credit” -- Feature Requirement: It will return the total credits of chosen courses. Since some courses are like 1-6 courses and weird Javascript data structure, we failed to implement it.

6. Course Detail page (Refer to 5.c.iii)



7. Admin Page (Django Admin Page)



Conclusion:

This project was a great experience for our team. Everyone was willing to work and take on large tasks for our project. Because everything was divided up so nicely, we rarely had any conflicts which allowed for consistent progress towards our goals with hardly any hiccups. When team members needed to cooperate on certain tasks, everyone was available on our group chat to discuss what needed to be

done. We learned a lot from the in class tutorials about Django, but we also did outside research to work with Scrapey, Selenium, Dhtmlx Scheduler, Javascript, Ajax, Jquery, and Bootstrap. In addition to just using these tools, we had to learn how to mold them for our purposes which did pose some challenges. We also learned a lot from the design process, specifically how to design an ideal model for our database. Our initial models needed changes to function the right way, but by the end our models made sense and worked as intended.

We had some difficulties throughout the project, but many were small and could be discounted. One issue we ran into was accounting for Django's User model when working on the user authentication portion of our project. We tried lots of different solutions before we finally got it all working about a week later. It was also hard working with so many different tools. One member of the team would know how to use a tool very well, but if another member wanted to make certain changes, they might also need all the background information about the tool to do so. This specialized a lot of our team members, but thankfully we were all willing to put in the effort to solve any of our own minor problems that arose.

If our team had knowledge of all the tools we would need and their uses from the start, it would have cut down on a lot of time. It would have also been nice to have more knowledge of Git before starting this project; but we all learned it and worked with it pretty easily. All in all this project taught us just enough about the tools out there to make something really cool, and hopefully we will be able to deploy our application in the future!