Actual Writeup

Overview:

Since the first overview in project one, we decided to take out the wishlist because we wanted to focus our work on other more essential features of the application. On the course schedule page we have added the feature to view a course description tab under the search fields. One major change is the frequency we need to scrape at. At the start we thought that we would only need to run the scrapper once a semester or so but we were neglecting to take into account the fact that student count in each respective class changes frequently around registration time. We originally thought the structure of the flow chart would be based around a javascript canvas or some framework we found online but we are implementing it with an html table instead. We have also expanded on our stretch goals, because the scrapper will need to be run more frequently than we expected we will need hosting for the server and the scrapper.

Team Members:

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

Github:

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

Design Overview:

The main project urls.py redirects to the uplanner app by default, and also includes a path to the admin page. The uplanner app routes to all of the links in the navbar verbatim, with a redirect to index by default. It also has a reverse lookup for individual course pages, and finally a url for loading just a subset of the schedule page, the contents of a dynamically added course tab.

The views are also simple, except for the schedule page. This has two views, schedule() and make_tab_content(). These render the overall page and tab contents, respectively. This means a dynamically added tab does not reload the entire page, but only the newly added section. The search results per course are written as a dict to make lookup of data per course easier in the template.

The data model is directly based on the ORM with minor modifications. The most important model is a Course; generic course information. Each class has many Sections, which could be lectures, labs, etc, and contains information particular to a specific section of a course. There is also a model for a User containing miscellaneous information about them. Each User has many Schedules. Each Schedule has many ScheduleCourses. A ScheduleCourse is the rendering of a Section in a Schedule. It contains a link to the Section it represents, and any formatting information specific to how the user wishes it displayed. Currently, it only indicates which schedule the course should be rendered in.

Problems/Successes:

Scheduler: The dhtmlx has constraints on its available functionality, but the code can be molded to what we want to use it for. The base idea for our scheduler has been implemented, but it is difficult to support multiple schedule views. However, this is a key part of what we want.

Flowchart: It is possible to use flowchart template to hardcode a specific flowchart for each major. The only hard point for drawing the flowchart inside HTML is the rangement. However, I decided to use table instead of flowchart and I used the highlight to show relative courses and the linkage. And I coded table generator inside javascript so that we do not have to hardcode complex flowchart for each course.

Backend: We had problems getting django forms to send data directly to javascript. We realized we can use ajax to do this, but still have to implement it fully in some places.

Scraping: Scraped Spire using 2 different frameworks, Scrapy and Selenium. We have successfully scraped data from Spire and are working to put the data into the database.

Sihua’s work **10%**

I arranged and styled profile page and flowchart page based on Ben’s Navbar and footers. I mainly focused on flow chart page and did researches to find out the best way to implement flow chart page. I used javascript to auto-generate flowchart (turned to be table there) and made the linkage between courses and each course’ pre-requirements.

Ben’s work **25%**

I contributed to the front end of the project. I initially coded a lot of the mock UI for the homepage, the schedule page, the prereqs page, and some of the admin page. I spent time watching videos and learning the do’s and don'ts of Bootstrap for this project; and used those techniques to get the columns working and scaling well for the UI. Most of the front end work went into the Schedule page’s search criteria and results portions. The rest was less difficult to do after doing that page. I also figured out and added the javascript to make our buttons on the Schedule page change UI properties and show certain parts of the application. I then did the write up on the mock UI, since I had a good idea of the direction of our project at the time from working with the UI, and shared it with everyone to edit if they wanted.

My other contribution has mainly been working on the actual dynamic schedule builder. We used a third party application called dhtmlx scheduler and it has taken time to figure out what code to use and how to even use their code. I molded a lot of the application to suit our needs like showing weekdays only, not showing actual dates, only showing reasonable times, adding/removing events with buttons, coloring events, and even just putting it next to our own html on the same page. There is still more to do like supporting multiple separate schedules, keeping track of added classes, customizable coloring of classes, and etc.

Tim’s work **10%**

I started off by helping frontend part of the project by making Mock UI html page for user and prereqs. Later I moved on to backend part of the project but was assigned to do user authentication related task which is later part of the project. So in the meantime, I organized our Github a little bit while making our first couple wiki pages for the project.

Jason’s work **30%**

In the beginning, I only did a little frontend by helping rework the display of search results and the search bar/options. More recently, I have done a lot of work on the backend. I wrote the ORM document for the database, and then wrote it more formally in models.py. I have updated as necessary, as well. I also wrote a base generic template based on the original html written by Ben. I then translated the original html to extend that html and use the django template language with for loops, if statements, etc. I did this for every page. I also planned all the url mappings and wrote them in the two urls.py files of our project. Finally, I wrote the views in views.py for all of our pages. The schedule view took an especially long time. The others are trivial.

After setting up the initial backend, I began integrating the javascript in schedule.html with our backend. We have many elements on our schedule page that require dynamic behavior, so the javascript is fairly extensive. I rewrote some of the functions so that they were no longer hardcoded based on element id. I also added a boostrap tabbed view to the search results to display detail on specific courses. This is done by rerendering only the most recently added tab with jquery's .load() function. I add the tab through javascript, and populate it with django and html from jquery load().

Liam's work **10%**

Started the scraping with cheerio, requests module in node js. After a while I realized cheerio and requests were not powerful enough for what we needed so I learned phantomjs and wrote code to go to spire and eventually got it to log in and go to the course page. We ended up switching over to python and Kerry took over scrapping. Later, I did work on forms and integrating the flowchart(prereqs page) with the database.

Kerry Ngan's Work **15%**

To start building a spire scraping bot, I first learned Scrapy and python. Scrapy was well documented. I soon realized that Scrapy cannot interact with javascript and decided to use Selenium to navigate Spire. There were a lot of problems along the way that took a while to fix but were eventually solved. The scraper currently outputs the course title and course information as an XML file. I am now working on cleaning the data and putting it into the database.