Actual Writeup

Overview:

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 scraper 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 of our scraping changes, we will need hosting for both the server and the scrapper. Additionally, since there is no unified format for prereqs, we need to parse them from text descriptions. Since this is likely to be very hard and prone to error, we hope to include a method for users to indicate and correct errors. It would also be nice for users to give feedback in general.

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 user login/logout functionality is currently very simple. We support users logging in and registering for new accounts using a fairly standard approach. One point of complication is in tying our own Student model and django's User model together. We do this by specifying a unique email per account. Then whenever a user registers, we create both a User and a Student for them, each using the same email. The email can then be used to look up either User or Student data.The schedule page was also edited to only retrieve and update data from the currently logged in user, and it is now impossible to see the schedule without logging in.

Problems/Successes:

Scheduler: The scheduler is working in the way we want it to as of now. However, there were many problems to overcome getting there, and there are still some issues with maintainability. For instance, there are a few places where we directly inject html from javascript, especially when adding to a list. This javascript html must exactly match the appropriate html in the template, with all classes and other attributes. This means that for some parts of the page, it must be changed in two places, and it is not always clear when that is the case. This could be a problem in the long run.

Flowchart:I found out that is not possible to have a link or a sub-button inside a big button even I changed button to div. And it is really hard to style things differently inside one button since every style will make things an individual part. When I assigned a function to button, every individual parts inside button would be signed the same function. Instead, I use on-click and double-click to make different functions working inside one button. I also found that it was really hard to connect courses with their pre-req courses in Javascript. Originally, I wanted to return more detailed pre-req info but end up with "simple" relationship.

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.

Team Choice:

We have spire scraping, an interactive page that uses ajax as well as jquery, and a third party schedule builder integrated into our web application. The schedule builder app and page interactivity are highly linked, as both run on javascript/jquery and use ajax to communicate with the server.

Sihua's work **16%**

Restyled flowchart page for multi functions and showed more info for courses for selecting. I added on-click function to "course button" so that we can select one course and system will automatically select all pre-req courses. Also if we want to unselect one pre-req courses the system will return an error unless we unselect the courses needed the pre-req before. I made the page to show the total credits for all selected courses.

Ben's work **16%**

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. The application now supports multiple separate schedules and keeps track of added classes. I have customized the front end further since the last project, and I've made sure we have everything prepared for these submission deadlines including essays.

Tim's work **16%**

Later on for this part of the project, I look into user authentication related documentations and implemented them into the website. The problem I run into was that our Student model isn't able to extends Django User model for whatever reason. I tried all the possible ways in the Django tutorial documentation sites and none works for our models. I solved the issue by linking User model with Student model using email fields. After user authentication, I look into the account registration related feature. Since User model and Student model are linked by email, so I implemented two separate forms, an registration_form.html page, and new function in views.py for registration. On top of that, I re-made some of the UI for user authentication/registration related html page for better looks. I also have integrated and fixed some of the UI issue for profile.html so it shows user profile appropriately. I also changed and fixed some of the issues with url redirect after user authentications.

We also ran into problems with migrations where we violate some of the column constraints but the column never existed. I look into each of the migrations files and delete the unnecessary one to clean out the migrations so everyone that need migration and start to migrate properly again.

Jason's work **20%**

I wrote the ORM document and models.py for the database, and refactored the old html to use django's conventions. I also set up the initial views.py and urls.py files. 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.

After setting up the basic schedule.html page, I focused on integrating the schedule.html javascript and the database. The schedule page is very interactive. The user can create and delete new schedules, switch between them, search for classes, expand search options and course details, open up course details in a new in-app tab (not browser tab), close the tab, and add and remove course sections from their schedule. I implemented all of these features and used ajax to update the database and session as necessary. I also continue using selective reloading of the page when it seemed appropriate. In sum, I wrote 10 new views, 7 of which are ajax views and the other 3 are selective rerendering. I also wrote or significantly edited 9 javascript functions and 4 template files. (although my team members edited the template files afterwards to make them pretty). I also altered the schedule page to use sessions, and only show/update data from the currently logged in user.

<u>Liam's work</u> **16%**

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. Worked on parsing the strings that hold the requirements for each course using regular expressions that way we can use them for the flowchart.

<u>Kerry Ngan's Work</u> **16%**

I worked on adding information from spire directly into the database. I have scraped a list of all the classes in the computer science department so far. Adding the information requires to first parse the information that is scraped so that it is in the same format as the django models that we defined. That has taken the bulk of my my time but since all the courses are in the same format, it will be easy to scrape all the other majors as well. Currently I am working on add the section for each class to the database. Also, I am working on a solution to add a many to many relation attribute to the database.