Class 5 – Project Evaluation

----------------------------------------------------------------------------------------------------------------------------

Group Evaluation:

Overall, the group dynamic worked quite well. We had 3 people so there was a lot of work to do, but we were all very interested in completing this project, not just for the course but because we actually hope to use it during our future course selection. It also helped that we knew each other beforehand. Sometimes it can be tough working with friends, but at least for this project, all our goals are similarly aligned and we could be more honest with each other. Especially since this semester was remote, it was much easier to communicate and stay in touch with each other and know what we were thinking than it would have been if we were strangers to each other.

Original planning

* Successful original planning
* Well organized, and had in-depth discussions early on about the major goals of our product, and how we would design our system.
* We believe that generally we met our milestones. One could argue that our UI was a little underdeveloped heading into our alpha version, but we made up for it heavily for the debut of our beta version. Otherwise, we hit our milestones and even had extra time to implement a few basic filters, which had initially been a stretch goal.

Experience with design, interfaces, languages, systems, testing, etc.?

* Overall, Python Flask was very easy to use and implement. Within 2 weeks, we had a working 3-tier connected system hosted on Heroku that we could display data on. However, as the project continued to grow, we noticed that although it is easy to write Python code, it is also very easy to write sloppy Python code. Without type systems and with an overabundance of libraries, it is easy to fall into writing methods that return multiple types (ie. in some cases return a list, in other cases a string) or return extra debugging data. We agree with Professor Dondero’s assessment that while Python is fabulous for small/medium scale applications, for large scale applications, more rigid languages are definitely preferable.
* Jinja2 templates are great with Flask, but as the application grows, accessing data through AJAX becomes much easier. We ran into some confusion during development where we would load some data through the template interface and some through AJAX calls. Ultimately, we have mostly switched over to loading data into the user interface through AJAX, but next time, we hope to start by doing this.

What surprises did you encounter on the way?

* When we first implemented SQLAlchemy as an ORM, we had followed the format of the code from the handout 10projecttechhandouts.pdf. However, after setting it up, we noticed that we were creating too many connections to the database (Heroku’s Postgres limits us to 20). The application needs to access the database for multiple purposes: updating user information, retrieving course information, etc, but the handout does not really account for this. We determined that the issue is that after we call session.commit(), we also need to call session.close() to break off our connections. We suggest adding a more thorough example in the handout to explain this.
* In late October, we realized we were about to run out of space on our PostgreSQL database. This was due to how we are storing course crosslistings, which each occupy an entire row in our database. With a 10,000 row limit and about 2,500 stored courses in our database (without crosslistings), we were filling over 70% of our rows with course crosslistings. Ultimately, we decided to simply upgrade our storage rather than rework our database schema, but we learned one of the most valuable lessons throughout our entire project: aim to estimate the scope of our data *before* choosing a software to house our database. This way, we can avoid reaching the upper limits of our storage, and quantify essential metrics such as the number of users our platform can house, and how many semesters of updated information our platform can store.

What choices did you make that worked well or badly?

* Database getter, adder, and updater methods methods were used in tandem to interact with the database
* While the course graph abstraction worked very well to help us understand how our system ranks suggestions, the actual implementation of it in the Python code probably could have been more fleshed out. Accessing data through SQLAlchemy is already extremely simple, and since we were under deadlines, sometimes we chose to ignore the abstraction level and just call database getter methods in main.py. As shown in the Programmer’s Guide, this created a somewhat messy architecture where sometimes we would use the course graph and sometimes we would not. While the system works, from a design standpoint, choosing to ignore the ADT was bad because it makes the system more complex than it needs to be. If we were to do it again, we would make sure to only access data through the coursegraph
* We implemented CAS authentication very early on in the project. We believe this was a poor decision and next time will implement it much later on. Since many browsers cache certain javaScript and CSS files, when we were testing and debugging JS code, we would need to clear browser cookies. However, this also clears the CAS netid, so everytime we made a change, we would need to re-authenticate our CAS, which was a huge pain during development.

Milestones:

---Copied from Project Overview > Milestones ----

**Minimum Viable Product**

* Home page - sleek, nice [Done]
* About page [Done]
* CAS authentication [Done]
* Ask for info:
  + Select classes that you could possibly have taken (calculated with the year we get from your data) [Done] (Ask user for input)
    - Depending on class year, require a minimum of 3 likes, 3 dislikes, and max of 10. [Done] (Changed to min 2 likes, 2 dislikes, max 6 each)
  + Ask personalized questions (makes sure the pulled data is correct): major, class year, [Done] **Stretch**: certificates? [Done], interests
* Calculate a list of ~10 5th courses to show you and display to the user [Done]
  + Can do a re search [Done] (Disliked Suggestions)
  + Can select one to remove and research for another one [Done]
  + **Idea**: can also allow student to pick *any* course [Done] (Search bar)
* We should have a page of classes you say you like and don’t like [Done]
  + You should be able to update these preferences on this page [Done]
* **How we get initial data:**
  + Google sheet: send a google sheet to our friends where the list 5 of their favorite courses, and 5 of their disliked courses. [Done]
    - We want at least 30-40 datapoints [Done (50 datapoints)]
    - Could send to our clubs, or even on listserv if we are feeling extra
  + Text: could also text people the same thing [Done]

**Stretch Goals**

1. Being able to change your preferences for liked/disliked classes as well as reactions for the Class5 suggestions [Done]
2. Resizing for tablet and phone [Done]
3. Filters [Started] (Completed Most Liked, Top Fifth Classes, Most Disliked)
   1. Class based filters
   2. By major, or by class year, or by certificate
4. Displaying information about the courses [Done]
   1. Numerical reviews (ratings)
      1. Our own metrics? I.e. sentiment analysis
   2. Written reviews
   3. Registrar link
5. Sharing the suggested classes via email or text
6. Semester suggestions

---------------------------------------------------------------

Looking at what we initially set out for, we accomplished all the milestones in our original MVP and are making good headway into some of our stretch goals like creating a mobile web app, allowing updating course preferences, and adding filters. Overall, we did a good job planning our project because we feel like we have a good, working product, but there is still so much more we can do. We have already met with the head of TigerApps who thinks our app has a lot of potential, and we plan to work in USG Labs to hopefully get a fully realized version out before Fall 2021 course selection.

What We Learned and Final Thoughts

----------------------------------------------------------------------------------------------------------------------------

Arin

This project truly taught me the full-stack experience. I had no previous experience in web-development, including server side and presentation side. In this project, I initially wanted to just take care of the backend logic because that is what I am comfortable with. However, I found myself working more with data and on the frontend side - two aspects I had never previously worked on. In fact, I had never worked with Flask, PostgreSQL (or any database for that matter), SQL Alchemy, JavaScript, CSS, etc. I only had knowledge of data structures and algorithms and Python.

With this inexperience came a massive learning curve. Of course, when learning a new skill, you never know the best practices. So our front-end code is extremely messy, especially the code that was written in the first few weeks. We wanted our system to be very dynamic and very visually appealing, so as someone with no experience, of course the code would be messy. But I learned the importance of documentation and communication with my teammates. As said before, there were some blips along the way with AJAX and Jinja2, and these things are to be expected.

Another massive aspect I learned about was quality control. Once we had a big enough system, any small change in code could trigger a massive chain of events that lead to a small little error somewhere in the application. Experiencing this first hand with our small codebase made me realize and respect the importance of testing. I wish we had done more testing in general, whether that be setting up toy examples to make sure certain logic works or to actually stress test our system.

Additionally, aside from the code, I really enjoyed taking the lead on the design/branding aspect of our project. Somehow a mishmash of colors and fonts made quite a nice looking website :) In all seriousness, it showed me a side of who I am and what my passions are, which very well might help me in the future.

Finally, while I had worked on a team before during an internship, my experience was drastically different in this team. I had an incredible trust for my teammates, and we communicated so well that we pretty much knew what needed to be done and what everyone was up to at all times. I am so grateful that we shared this journey together, and we are attempting to become a TigerApp, so I can’t wait for what the future holds!

----------------------------------------------------------------------------------------------------------------------------

Ethan

I focused mainly on the overall project and implementing pieces where needed. I worked on the autocomplete and many of the AJAX methods and helped outline the course graph data structure.

I think the biggest piece of advice I learned from this project is that it is important to address test code as soon as you write it and at as low a level as possible. There were times when I would write code and our system seemed to be working so I would move on, and then weeks later when testing it, I would encounter a bug. Debugging code you wrote 3 weeks ago is a lot harder than debugging code you wrote an hour ago. Even though it takes a little extra effort, I think it is really important to make sure to test your code at the earliest opportunity. I also think my confidence as a software developer greatly improved with this project. I feel like I really understand how MVC architectures work and how to think about requirements and use cases ahead of time.

I also think it is worth mentioning how much I learned working together as a team with Chaz and Arin. Their energy and enthusiasm for the project really helped keep me interested and productive even while I had a lot of other classes to think about. I really see the value of having a great team and hope to look for that in industry.

----------------------------------------------------------------------------------------------------------------------------

Chaz

Although I focused more on the back-end of the application, I definitely feel much more confident working in all areas of the stack, and learned a *ton* through this whole process. From the early stages of the project, as my teammates and I discussed the core of our algorithm and our Graph schema, I realized the importance of discussing these high-level ideas in depth. Looking back on the first six weeks of the semester (before we even started writing the application), I can more fully appreciate how our early organization and planning paid off in the long run, and made building the application much more streamlined than it would have been otherwise. Moving forward with future projects, I’ll be sure to communicate frequently with my teammates, so we’re more likely to have a solid idea about how the product should look and feel.

Having had little experience with front-end or AJAX going into this project, I’ve also become much more comfortable with JavaScript and jQuery and using these tools to responsively manipulate web pages. One of the trickiest problems I solved while working on this application was fixing the “refresh suggestions” feature, which – while confusing and arduous to figure out – was a fun debugging experience. I was then able to more fully understand the logic behind requests/responses are handled, as well as how to manually manipulate browser event signals.

Class 5 was also my first fairly large-scale project working in a team of 3, and has shown me the importance of being reliable, as well as responsible with how I write and disseminate my code. I’ve built great habits such as using branches in version control, testing code in small chunks, debugging thoroughly, and being proactive with error handling. I cannot overstate how much I learned from Arin and Ethan, and I’m so thankful to have gotten the chance to work with them. This class has exceeded my expectations, and this project has made me even more excited to continue my COS career at Princeton!

----------------------------------------------------------------------------------------------------------------------------