EECS 399 Final Summary
December 10, 2015
Pranav Mayuram

At the start of this course, Professor Jagadish and I determined that the concrete deliverables of this project would include a functional, Node.js, REST API that could identify past alumni's current addresses with a reasonable amount of error. The deliverables also include thorough documentation, which can be found on the project's Github repository at https://github.com/pranavmayuram/AlumnUS. This document will outline how each of these tasks were completed, as well as the skills and technical knowledge that were gained in the process.

The AlumnUS project had three distinct parts that were needed for its creation. First, an interface by which someone could upload data (CSV or Excel), and it could be processed into a query for an individual. Second, a source (API of some sort) from which personal data could be sourced to find the current address of an alumnus. Finally, it needed a way to choose which result, out of all of the results the people search API returned, was the correct individual, such that the correct address could be selected for the individual the AlumnUS API was searching for.

To tackle the first issue of getting data from a user, the AlumnUS API uses a node module called "Multer" which is capable of storing a file inside a specified folder, with a specific name. In this case, the files uploaded by a user were stored in "uploads" and they were stored under a time-based UUID to avoid any conflicts in filenames when they are being processed. A basic UI was devised to handle this portion, and the user can simply upload a file through a website, and specify basic attributes of the file. After the API receives the file, it uses another package called "xls-to-json" which parses the Excel file and returns a JSON document. This JSON doc is then used for the remainder of the Alumnus API's process, so that the data is easy to manipulate. Creating this process was filled with interesting design decisions, including how to avoid conflicts between files when they are stored, and how to make the process as painless for the end-user as possible. It was helpful that I had used "Multer" before, which meant that setting up the file-upload process was extremely simple. I also tinkered around with quite a few different node modules, until I found the right one to handle the Excel file parsing. From the start, I knew it would be ideal to have all of the data in JSON, and it really made all of the work after that much easier.

For the second issue, finding a people search data source, the path to finding an ideal solution was not simple. The original premise of the AlumnUS project was to use social network APIs to search for the individuals who had lost contact with the university, particularly LinkedIn. After reading much documentation about these social network APIs, and creating my own sample APIs to query the social network APIs, it seemed that there was no simple way to access the information that was needed for AlumnUS. Both LinkedIn and Facebook had closed their APIs off for people searching. I contacted developers at both companies to inquire as to why they had done this, and learned that their APIs had been misused, or proved as security vulnerabilities, which meant less access for the public. Currently, the university is working to become a content partner of LinkedIn, but because this process was on an indefinite timeline, I proceeded to look for other solutions. This led me to an API called Pipl. Pipl searches social media APIs, other third-party and “deep web” sources to find information about individuals given a particular query. I then developed a few sample APIs that used a demo of Pipl as the data source, and proceeded work with that. Pipl is a paid API, but after talking to account managers at Pipl, I found that they provide $300 of free credit to non-profit users, which made it simple to attain a functioning key. Pipl only allows 10 requests per second, per key, so I used a node module called “bottleneck” to limit the rate at which requests were made. The requests were made using the “pipl” node library, which was made by a third-party developer. Recently, I created a way to query Pipl with their “person object” in Node.js, which takes in more details about the user. This is helpful when there are many possible results for one query. This portion of the project was particularly interesting to me because, typically, things like these libraries and data stores are specified for any project in a class. Because the problem statement did not include any technical details, this process of finding the proper resources was particularly fun and different from the challenges faced in other project settings.

Finally, the process of filtering the results returned from Pipl proved to be the most complex challenge, and is still in development. The Pipl API can take in different parameters, so to start, just the name string is passed in. In turn, it sends back information, which is either a “person” object, if it believes that it has found the only possible result for this individual, or a “possible_persons” array of objects, if it sees that there could be many different candidates for the person specified in the query. For the “possible_persons” array, the maximum result set is 49 possible people. In the case that 49 results are returned, there could either be 49 possible results, or more than 49, so the AlumnUS API will make a more detailed request if the API returns 49 potential candidates. Then, yet again, even more detail will be added if the more detailed request returns 49 candidates. The idea is to avoid filtering out too many potential candidates, and leave as much as possible to the internal filtering mechanism of the AlumnUS API. Once the ideal result set if found, the internal filtering mechanism kicks in, and generates a rank for each result, and assigns the highest rank amongst all the candidates, to the person that was sent in the query. The JSON body generated in the original Excel upload is then appended with the associated address from the Pipl result. In the case that the result seems unsatisfactory, an error code is added to the JSON doc in a separate attribute. This updated JSON doc is passed to the “json2xls” node module and converted into an Excel sheet, which is then downloaded to the user. Creating the filtration system was a very different experience from most classes again, and required thorough hand testing to see which results were desirable, and use these examples to improve the filtering algorithm. Because results are not always satisfactory, it was important to add the error codes, which are explained in depth on the AlumnUS Github repo.

Overall, I felt that the process of creating AlumnUS was a great learning experience. I learned to look at problems that were not about getting the clear-cut “right” answer, but also understanding that sometimes you must develop a system that will inevitably fail for some cases, but the point is to minimize and point out those cases. Designing AlumnUS from the ground up was extremely interesting and I hope it turns into something the university can use.