CS 316 Final Report

David Kohler, Michael Nicholson, Karen Ou, Siyi Xu, Jingchao Zhou

Application Description:

Every semester, registration and book-bagging are a recurring source of frustration and stress for many students. Every major and minor has required prerequisites, core classes, and electives that make course scheduling a time consuming and stressful experience. This is especially true for students with multiple majors and minors.

At the same time, upperclassmen and graduates have a wealth of untapped knowledge and experience with planning courses and also have completed course paths that are valuable to many underclassmen trying to figure out what classes to take and when. We intend to bridge the gap between these two groups. This application allows upperclassmen to share their course paths with underclassmen in order to reduce the time and effort required to choose classes each semester. This also gives undeclared students a way to see the range of potential paths and program combinations that are possible at Duke, all with the reassuring knowledge that every path has been completed by previous students.

Our application is comprised of three main components hosted by Heroku: a PostgreSQL database, a Django based backend, and a React frontend. Together, these underpin a web-based application that allows underclassmen to query programs and course paths and upperclassmen to add their completed course schedules. We also provide the recommended course path provided by various departments to supplement the schedules entered by upperclassmen students.

The PostgresSQL database houses all the data we query for our application. The structure and tables of this database are described in-depth in later sections. Since our data is highly structured, we decided a SQL database would provide the best framework and data structure for our application. We choose Postrgres for its useability and since it was the SQL management system we were the most familiar with.

We chose to use Heroku to manage our web application and integrate with Heroku PostgreSQL because it allows us to collaborate on the project and has its own version control system. Heroku also provides a hosting service with simple Github integration, so we can easily deploy our application to Heroku to test its functionality. As for the tech stack, we chose to use Python with the Django framework for the backend because every member on the team is familiar with Python and Django has abundant built-in features and support. For the frontend, we decided to use React because it is more modular and flexible than plain HTML.

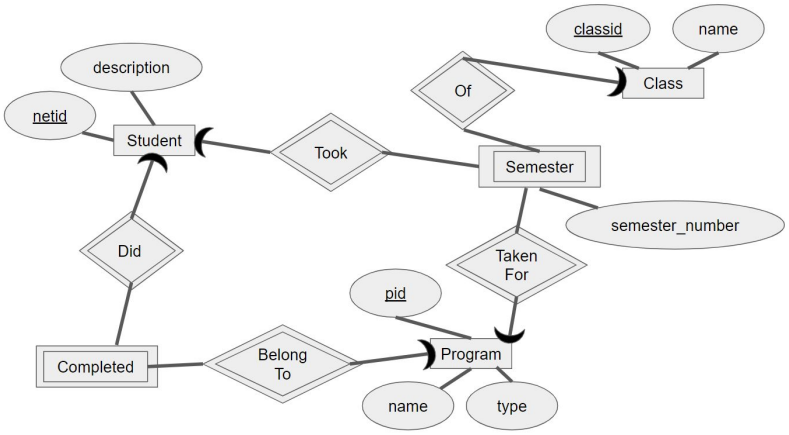
Our web application itself has a number of features. On the homepage, a user is immediately given the option to query one or more programs from a dropdown window. At this point, they can select “Search Academic Plans” and will then be presented with potential corse paths. On the sign-up page, users can input their netid and desired password to create an account. Then have the ability to sign in using that information on the login page. Once users have created

an account, they have the ability to add their information and course plan on the ‘Account’ page. They can here input a short description about themselves, their completed programs and their completed course schedule. This information can then be added to the database and queried by other users.

Data Sources:

By nature of our application, the majority of our data is produced by the users themselves. However, as a starting point, the group members uploaded their respective course plans and those of a few friends. As a supplement, we also initialized the database with the recommended course plans for a number of the most popular majors at Duke. In total, we have plans for 28 programs as well as functionality for new users to add their own plans. We additionally scraped the courses currently offered at Duke for both Pratt and Trinity from a public open website(https://m.siss.duke.edu/app/catalog/listSubjects) and the programs currently offered in Pratt and Trinity from each department so that we can limit the required user inputs for the class and program tables. The student, completed and semester tables will be further populated as new users join the platform.

E/R Design:



Data Assumptions:

For the Student table, we assume all ‘Student’ entries are Duke students that have completed a program or have a full schedule to complete their intended programs (minors and majors). We use their netid as a unique primary key for each entry. We also allow for an optional description field to allow students to write a short summary of their interests and careers. This table is populated as users sign-up for the application.

For the Class table, we populated the table with all courses currently offered by Duke. For each course, we use a unique class ID that is comprised of an uppercase code corresponding

to the department that offers the class and an integer corresponding to the class within the department, e.g. “COMPSCI316”. We also include a name column corresponding to the full name of the course, e.g. “Introduction to Databases”.

For the Program table, we initialized the table with scraped programs from Pratt and Trinity. We also allow users to add concentrations that are not already in the database. We assign every program an integer unique key that is generated when a program is added to the database. We also include a name column corresponding to the department of the program and a type to differentiate between B.A., B.S., minors and concentrations.

In the Completed table, we hold every completed program for the students in the Student table. We enforce that every student and program in the table corresponds to a matching entry in the Student and Program tables, thus netid and pid are foreign keys in the table. We assume that all the entries have corresponding course plans in the Semester database, i.e if a program is in the completed table then we have a full course path for it.

For the Semester database, we assume that the entries represent each class a student took in pursuit of their respective programs. We assume each student, class and program in the Semester table corresponds to an entry in the Student, Program and Class tables, thus netid, classid and pid are foreign keys in the table. The table also includes a semester_number column corresponding to when a student completed the course, e.g. a course taken in Freshman Fall would have a semester_number of 1 and one taken Senior Spring would have a value of 8.

Datatables:

1. Student (netid VARCHAR(8) NOT NULL PRIMARY KEY, description VARCHAR(512))
2. Class (classid VARCHAR(32) NOT NULL PRIMARY KEY, name VARCHAR(256) NOT NULL)
3. Program (pid integer NOT NULL, name VARCHAR(256) NOT NULL, type VARCHAR(256) NOT NULL, PRIMARY KEY(pid))
4. Completed (netid VARCHAR(8) NOT NULL, pid integer NOT NULL, FOREIGN KEY (netid) REFERENCES Student(netid), FOREIGN KEY (pid) REFERENCES Program(pid))
5. Semester (netid VARCHAR(8) NOT NULL, classid VARCHAR(32) NOT NULL, pid integer NOT NULL, semester_number integer NOT NULL, FOREIGN KEY (netid) REFERENCES Student (netid), FOREIGN KEY (classid) REFERENCES Class(classid), FOREIGN KEY (pid) REFERENCES Program(pid))

Conclusions:

Ultimately, we feel the application provides the missing bridge between upper and lower classmen we set out to provide. It is straightforward for underclassmen students to query our database, and we have a robust framework for adding new users and course paths. That being said, until the application gains popularity, the number of completed programs will remain a bit sparse. We currently have 28 programs with completed paths which covers the most popular programs at Duke but do not have the full set of pairwise combinations of programs that our

users may wish to query. As users are added to the application, though, this constraint largely corrects itself. We also lack robust planning for courses taken abroad. Since these courses vary widely from school to school, and even semester to semester at these schools, we did not feel it was satisfactory to create courses taken abroad in the same manner as those taken here. Future iterations would take these differences into account.

In terms of implementation, we are satisfied with the overall structure of our database, but this came after many iterations of design. We were forced on two occasions to change the schema of the tables which required significant backend work to repopulate the data because of the numerous foreign keys in our schema. We faced a similar issue with connecting our database to the backend, Django has precise ID requirements under their models framework, and again had to repopulate data into a clean database initialized in Django. While the database ultimately took on a useable form, we could have saved ourselves significant time and effort with more through planning for the database schema from the onset and a deeper understanding of working with databases in Django.

Looking forward, there are a number of features that could be added to the application. We could integrate course or professor reviews to help students choose what professors are the best to take each course with. In many cases, the professor teaching a course is more influential on a student's performance and enjoyment of the class than the material itself. In addition, knowing the difficulty of different courses before the semester could help students smooth out the overall difficulty of different semesters, for example pushing a difficult class to the next semester if you already know you are taking 3 other difficult classes in the current semester. We are satisfied with the end result of this semester's work on the application, but we believe these future steps and directions would further the usability of the application and make the lives of students a bit less stressful during course selection.