**UA Enrollment System**

At the start of this project, I had decided to work alone. I knew that I wanted to mimic enrollment at University of Akron (or any college) , and I knew how I wanted to do it. The idea was to make my system as legitimate and effective as the actual, current enrollment system, within reason. However, the whole point of why I chose this undertaking was to have the chance to implement my own feature in addition to what other systems can already do. The idea for the feature originated from the fact that I've always wished that enrollment could be done more easily, and much faster, and I knew how it could be done.

***Technical Documentation***

To fully make use of my enrollment website, one must first create an account (but one may also just do course lookups). Click "Create Account" and choose a minimum 4 character username and password. If the username is taken, the user will be forced to go back to the previous page and try a new one. Once the user is logged in, he/she may begin hunting down classes to enroll in. This can be done in one of two ways. There is the traditional method, which is similar to what Akron and other colleges already use. Using this enroll method, the user may find courses by an exact course ID, course name, department, professor, and time slot. The user need only have an exactly correct input for the course ID, all other components will use an approximate search. For example, the user may type "Computer" for department name, and all Computer Science courses will appear. The user may fill in multiple components to filter down the search. On the next page, the results will be displayed, and the user will have a line to type the course ID and click "Enroll". The course is then added to the user's schedule. The user may continue hunting down classes until his/her schedule is complete. However, there is also the "Build Smart Schedule" option, which is what this entire project was designed for. One line is given for the student to type in all of the courses he/she wants for next semester (comma separated). Every possible schedule is built of these courses, and the user need only click "Enroll" on whichever schedule he/she prefers. He/she is enrolled in all of those classes and the process is complete that easily.

***Implementation***

The initial page is an html, and all following pages are Java Server Pages. This allows me to add user accounts and schedules to the database, as well as return courses from the database. When creating the account, the system will add the user's username and password to a table of accounts in my database. Also at this point, a separate table will be created for the user (titled by his/her username). This table is meant to hold the courses as the user adds them to his/her schedule. When enrolling using the standard method, I am using String.contains() for all except the course ID to compare the user's input against the class components in the database. This is what allows, for example, the user to search "Alg" for course name and the Algorithms classes are returned. The user just types the course ID in and clicks enroll, and the class is inserted into that user's table for his/her schedule in the database. Using the "Build Smart Schedule" option, things are a bit trickier, and quite a bit of time was invested in implementing this feature. Essentially, once the user requests the classes, the database returns all of the classes of all time slots into a ResultSet. Then, every possible unique combination is created of these classes, which is a concept one would learn from statistics. This concept was adapted from code that was taken and altered for my purposes from [here](https://stackoverflow.com/questions/29910312/algorithm-to-get-all-the-combinations-of-size-n-from-an-array-java), from the top answerer. To further explain: each class's components are held in a String[ ]. Each class is held in an ArrayList (ArrayList<String[ ]>) object, which represents an entire schedule. All possible schedules are held in yet another ArrayList (ArrayList<ArrayList<String[ ]>>) object. From here, I needed to get rid of certain schedules based on certain criteria. The stats concept uses what is called "choose N", where the number of classes in a schedule is N. So if I type in just "Internet Systems Programming", which has 3 time slots in my database, theoretically the code could generate every choose 1 combination, choose 2, or choose 3, where choose 1 would be every schedule of 1 ISP class, choose 2 would be every schedule of 2 ISP classes, etc. In my system, in regards to "choose N", N is the number of input courses. So if the user types only "Internet Systems Programming", 3 schedules will appear of choose 1, one for each ISP time slot. However, if the user types "Internet Systems Programming, Spanish 3", the system is now using a choose 2 combination, and some schedules will yield 2 different time slots of Internet Systems Programming, instead of 1 ISP and one Spanish. Thus, I wrote the code to cycle through each schedule in the ArrayList and remove any schedule that contains duplicate class names. Now, the whole point of this feature is to find every schedule without time conflicts, so I needed a technique to do this. You might notice on the website that it requests you to type the courses "in order of priority", so your final entry will be the least important course for you to take. The code will look through each schedule and remove time conflicts, but it will remove your least important classes first. So if the courses I input are of priorities 1,2,3,4,5, in the event of time conflicts, 5 would be removed first if necessary, then 4 if necessary, etc, and it will do this for every schedule. Lastly, if, for the same previous example, the best schedule gets 4 out of 5 courses, all schedules with fewer than 4 courses remaining (meaning that particular schedule had more time conflicts) will be removed.

***Conclusions***

For future work, I would likely try to give the system some styling on every page, and make it look official. Due to having a deadline, I was really only able to implement the core functionality to demonstrate my idea, and not make it feel real. In addition, I would add further functionality, considering that as of right now, the system lacks a very key component: the ability to remove courses from the schedule. One major lesson I learned is how powerful Java can be at times, especially when coming from a primarily C++ background. The ArrayList is a powerful tool, considering it is every bit as functional as the C++ vector, but can add and remove elements of any type. The biggest lesson I learned, in my opinion, is that the biggest different between any regular coder and a true computer scientist is that a computer scientist should be able to see a grand picture of what he/she wants, use any tools they can find from any area of life, and find a way to reflect it in code. I find that many people will just try to code and debug their way out of a situation, using numerous edge cases, and hope for the best. This is what I was doing at the beginning of implementing my new feature, but I had to think like a true computer scientist and look for the most appropriate solution, which came from the field of statistics.



