Proposal for a Course Scheduling Visualizing Software
Object-Oriented Programming Class (ENGR-UH 2510-LAB) - Term Project

Nishant Aswani, *nsa325@nyu.edu*, Barkin Simsek, *bs3528@nyu.edu*

I. PROBLEM DEFINITION

A. Problem Identification

Students currently have to navigate through NYU Albert or the NYUAD Student Portal to view classes and the times at which they are offered; it becomes quite difficult to view classes in the context of others and see how they would fit ones schedule. Students also do not have access to any planning software to help them puzzle together their basic degree requirements with other courses. Thus, our solution is an intuitive graphical solution allowing students to input class preferences and receive a recommended, visual class schedule.

B. Problem Analysis

In order to select classes, students have to navigate through a sluggish and maze-like text-based system embedded into NYU Albert to view the timings for a given course (see Figure 2). The desired courses then have to be added to the course shopping cart, after which courses can be enrolled into, assuming the system confirms that there are no scheduling clashes. This implies that students have to either note down or remember the timings of a previously selected course before adding a new one to avoid scheduling clashes later on. This is problematic because the course times are provided in a convoluted and difficult to read format (see Figure 2.

Furthermore, NYU Albert also does not indicate whether a certain course is a major requirement, nor does it mention pre-requisites or co-requisites until after courses undergo verification. Any verification issues result in the student having to go navigate through the course list again.

Essentially, there is no intuitive and graphic tool for students to:

- Clearly view and compare section timings for a course to quickly build a schedule with no incompatibilities.
- Compare their major requirements with desired electives to, once again, avoid scheduling conflicts.
- Enter certain preferences, such as time or course type, to obtain possible course choices to fill gaps or credit requirements.

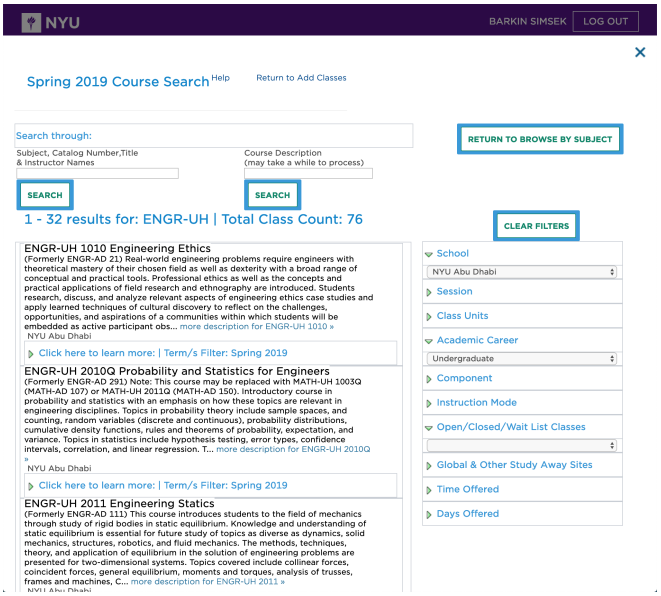

Fig. 1: Course times are hidden under the Read More tab

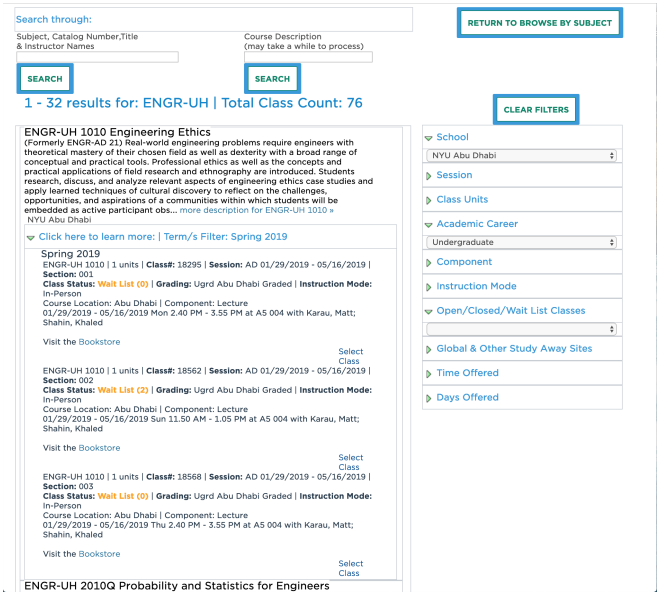

Fig. 2: Course times are not emphasized nor easy to read

C. Summary of Solution

An intuitive, graphical system which allows students to input their major, completed courses, and time preferences to receive a list of possible courses, a block scheduling interface, and recommended courses. This will allow students to visualize their calendar as its built to avoid conflicts.

II. CONSTRAINTS

A. Data Constraints

As the course scheduling software is heavily dependent on time and term data, one of the largest constraints stems from the source of data. For initial testing, we looked towards the NYUAD Student Portal to obtain all the data in XML format (http://services.nyuad.nyu.edu/academics/courses). The XML data was uploaded to a MySQL database, which was then accessed using a SQL database connector library in Python. Data between various tables was correlated using primary and foreign keys already provided in the XML file (see Figure 3).

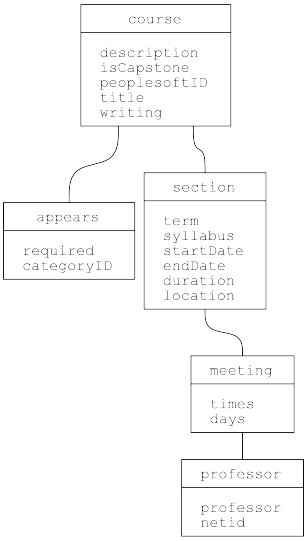

Fig. 3: Visualization of the relational database

The data constraint lies in the fact that the Student Portal's database of course information is not properly formatted; the class times are especially unorganized, with some times, for example, written in 24 hour format and others in 12 hour format. Looking towards Coursicle, a generalized scheduling app, and NYU Albert's public page, we sought to find a better source of course information; however, it seems that both services grab data from an inaccessible database. Thus, it seems that the project will be utilizing the XML file obtained from NYUAD's student portal.

B. Architectural Constraints

As the software is meant to be part of a larger future web project, we are constrained in our architecture of the course scheduling software. Since we are avoiding the use of the PHP programming language, we are required to use a framework in which to deploy our Python script. We are considering a microframework such as Flask, which is actively maintained, allowing us a lot more freedom to write our Python script over a solution like the Django framework.

Thus, as of now, we are constrained to separating our graphical user interface (GUI) Python script (wrapped in Flask) from our back-end database interaction and information storage script (see Figure 2). The idea is that the GUI script will gather presentable data from the back-end Python script and also return user selected data back to the back-end script to carry out operations.

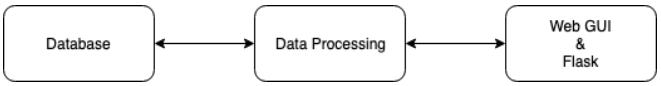

Fig. 4: High-level model of the course planner application

III. SYSTEM ARCHITECTURE

A. High-level Architecture

Figure 7 shows the high-level architecture of the course scheduling app. In summary, the user will request to create a schedule, select their (intended) major from a list of available majors and select their year, select completed requirements, and select course and time preferences. This will land the user on a scheduling page providing them with a list of recommended classes, a reminder of classes they have already taken, and also a reminder of classes that are recommended by the bulletin (based on the current semester)

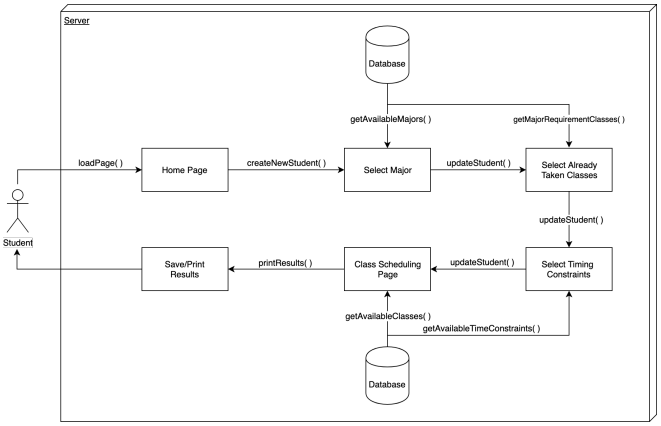

Fig. 5: High-level model of the course planner application

B. Implemented Classes

As the Python script for the GUI is wrapped in Flask, the two back-end classes will be imported into the main script and objects from both classes will be instantiated as needed. Figure 6 provides a summary of the two classes and their functionalities. Essentially, the Database class exists to interact with the mySQL database and return arrays of information. On the other hand, the Student class exists to store the information of the user and build their profile as they move through the program, all the while interacting with the Database class to obtainn information. The front-end script calls functions from both classes as needed.

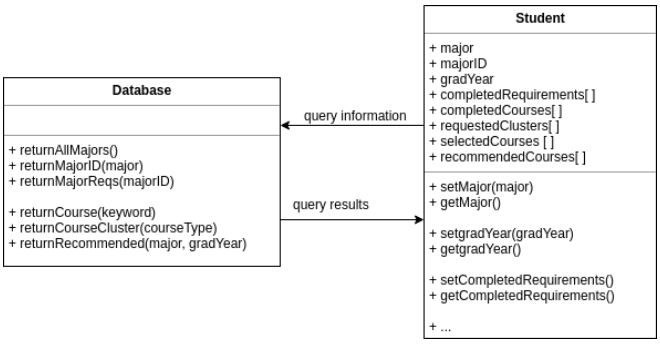

Fig. 6: Back-end class implementations

C. Technologies

The system will be written in Python and Flask will be used as the web framework for deploying the application to the web server. The course information obtained from the student portal is stored in a relational MySQL database on the Linux web server. HTTP requests coming from users are handled by the Apache web server and automatically transferred to the Python App container created on the Linux web server. Later, Flask web framework takes over the requests and lets us process requests by using Python. Classes implemented in Python are used like libraries to abstract database transactions. Data that need to be transferred are being passed to classes and later classes deliver data to required destinations at a lower level.

Utilizing Flask and the Linux web server gives us the ability to deliver content to students whenever they want and wherever they are, as long as they have internet connection.

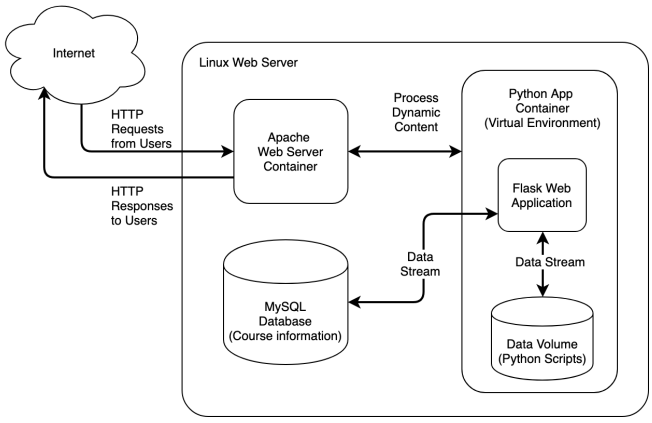

Fig. 7: A broad summary of the technologies used

IV. FUTURE PLANS

A. Milestone

Our goal for the upcoming milestone is being able to through the background data acquisition part. We plan to successfully show the classes that a user can take. We hope to finish the visualization part but our priority is completing rough functionality. So, users will be able to enter their majors, select previously taken classes, enter various class scheduling constraints and view the list of classes that satisfy the entered criteria when the milestone is achieved.

B. Future Goals

The course scheduling software is meant to be part of a larger project called nyuad.app, which is meant to act as a more focused and accessible student portal. Our goal is to provide the course scheduling software as a service that is part of the nyuad.app project. Thus, we would like to improve on the CSS aspect of the application, making it compatible across all browsers. Another goal is to eventually conduct student feedback surveys to understand what functionality is missing or what would improve the aesthetics of the application.