GPA Web Application – Team 4

Pitch

We intend to create an ECE based GPA disparity page that provides additional information about technical electives in a convenient, understandable manner.

Functionality

1. Users can view the GPA disparity of engineering ECE technical groups
   a. Users can view ECE belonging in different technical groups
   b. Users can view the GPA of different ECE classes belonging in different technical groups
   c. Users can view the GPA *by section* of different ECE classes belonging in different technical groups
2. User can view course details
   a. Users can view the credit for their searched course
   b. Users can view the size of their searched course
   c. Users can view the prerequisites for a searched course
   d. Users can view the different instructors that teach their searched course
   e. Users can view courses that are similar in course content to their related course

Components

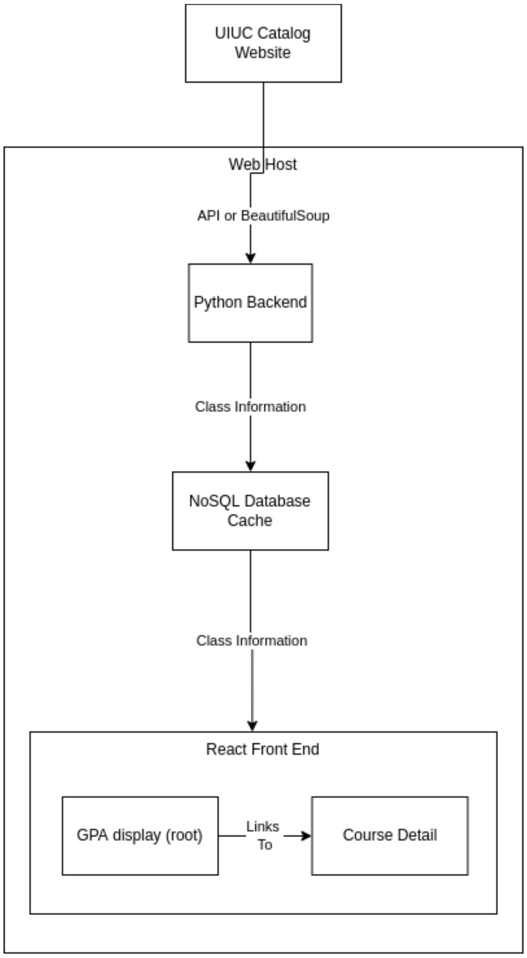
1. Python Backend
   - Functionality – Data collection and processing
   - Programming Language(s): Python because three of our team members are comfortable with python and it is easy to code in python.
   - Major Libraries Used: JSON, Flask, MongoDB, BeautifulSoup (if needed), and data processing libraries like Pandas, Numpy
   - Testing Methodology: Pytest unit tests because unit tests are good at testing backend code. We chose Pytest because the syntax is simple to read.
   - Interactions With Other Components: Flask to create an API for the React front end to query
2. React Front End
   a. GPA display:
      - Functionality: Displaying the GPA disparity of classes under different technical groups. This is our main component and incorporates our biggest user-facing feature.
      - The GPA display component will be the front-facing component. Under each class's GPA the user can click into course detail (our second feature)

   b. Course Detail:
      - Functionality: Course detail will include useful information that students will need when exploring a class: credit hour, size, pre-reqs (which is often time misleading on course explorer), instructors, and similar courses. This feature can

be developed independently from our main feature because it does not have functional dependency on GPA display. It will be a free-standing component.

    - This component will be reached from our GPA display component.

- Language: Typescript react because two of our team members are familiar with Typescript and it provides better type safety than JavaScript.
- Major Library: MaterialUI for GUI consistency, KendoReact Charts or ApexCharts to display scatter plot of GPAs
- Frontend testing will be manually done, a series of requirements will be written prior to implementation to check that the frontend is following its specification.



Weekly Planning / Schedule

Goals for each week:

Week:	Tasks
1	• MongoDB cache • Data collection (UIUC's API / BeautifulSoup Scraper) • API Schema
2	• Determine what data will be needed for GPA graph page and upload to *graph_requirements.txt* • Create frontend class page with mock data
3	• Data processing to meet needs of *graph_requirements.txt* • Flask API • Host API online
4	• Create frontend graph page with mock data • Host Webpage online
5	• Update Flask API and data processing as needed for frontend • Expand to collect for more majors within the college of Engineering
6	• Network Requests from API & React Data Layer • Navigation between Class Page & Graph Page • Document 5+ things to test in next week's manual testing
7	• Test usability and navigation of the Class Page and Graph Page (ECE) • Update and fix any problems found from manual testing

We originally had an extra week for planning at the beginning, however, we felt it would be better to start coding earlier in case we got stuck on anything.

Potential Risks

1. Long-term maintainability – the reason this project was created was to address aspects of the old projects that were no longer maintained. We plan to address this by obtaining data programmatically rather than manual entry of data. If the formatting of the data changes, it will likely take a day or two to adapt to the new format. Coding the project in a manner that is compatible with open-source licenses will others to fork and maintain in the future.
2. Legal issues – This project is for non-profit and academic purposes, so it falls under fair use. The data used for this project is available to the public. We will cache data locally and source data

contentiously to prevent it from being blocked from the source data or receiving C&D orders. The time impact of this risk is likely about 3 days. If our backend is blocked, we can use client-side on-demand polling so that all polling traffic isn't coming from a single IP address. Otherwise, we can rely on a different backend method and do less aggressive polling if it becomes a problem.

3. The data is dynamic which may lead to accuracy issues. For example, a different professor may be assigned to a class at the last minute before a class begins. Ultimately, the data can only be as accurate as the source data. We will try to balance continuous polling of data with recency of data to provide up to date information to the best of our ability. The time impact of this risk is likely minimal. Ideally, we should implement database caching in our project and just tune cache recency settings as needed.

At present, our 5-week schedule allows plenty of time if we encounter any of the above-listed issues. For issue 1, because we have a local database cache to load from for if the data format changes, the front end team will not be impeded due to the API masking any changes to the backend. Issue 2 would likely the project back a week – the frontend team would request the data and forward it to the backend. Issue 3 should not impact the schedule of our project.

Teamwork

At present, we do not expect there to be significant issues with code working on one person's computer, but not another person's computer. If it becomes an issue, we will investigate solutions such as EWS labs, docker, pyenv, and Windows Subsystem for Linux if necessary.

Expected sources of friction seem to be lack of availability of team members and waiting until near the deadline to begin a project. We have tried to mitigate this by filling out when2meet schedule. As for being slow to start projects, we believe that as we familiarize ourselves with our semester workloads we will be able to better balance our time.

Given the transition to online with the pandemic, friction of collaborating has already been reduced. It is easier to set up meetings between individual team members if someone runs into coding issues. Online eliminates unnecessary travel time and screenshare makes it easier to see what the other person is having troubles with while still allowing the person with the coding issue to be involved. Zoom, provided by UIUC (University of Illinois at Urbana Champaign), allows high resolution screen share.

Team management - We will use team management software like Trello in order to keep track of what tasks need to be done to implement the project. While we originally considered breaking into teams of front end and back end, we were concerned about workload balance and having issues with one team waiting for the other team to implement the related parts required to start work. Instead, pairs will work on groups of related tasks. At present, we suspect that Joe will be working on the frontend as he is more familiar with Typescript, whereas Allen will be working with python, where he is more comfortable. Jason and Kalika did not express preference, so they will help where needed. Task roles should become more apparent once we set up a Tello / Jira.

Continuous Integration

Library for testing: Pytest for python (note- supported on specific versions), Jest and React Testing Library

Style guide: For React, the airbnb style guide - https://github.com/airbnb/javascript/tree/master/react

For python, the pep8 style guide - https://peps.python.org/pep-0008/

Computing testing coverage: Coverage.py for python (popular and can work with Pytest), React Testing Library has a built-in tool for React coverage

Pull request workflow: As noted by the requirements, we will have each PR is reviewed by one other team member, and that each team member reviews at least one PR each week and provides any necessary feedback. We will open PRs within 2 days of being initially noted, we will select a reviewer based upon who is closest to that specific work. Because we will be working in pairs, a person's partner can be an alternate reviewer if they fail to review your PR. Each team member will have their own branch to commit to. We will try to avoid merge conflicts by commenting our own name above functions and communicating early week what classes, functions, and functionality need to be implemented. If a person needs a function changed that a different person owns, they will need to contact the other person to change the code. If there are merge conflicts, it will provide a good opportunity for our group members to learn about git rebase.

Additional Notses

Our project is similar to uiucmcs.org however, rather than being geared toward the needs of CS students, our project will be targeted to the needs of ECE majors. We plan to include basic visuals akin to https://waf.cs.illinois.edu/discovery/every_gen_ed_at_uiuc_by_gpa/ however, the novelty of our project is the integration of the two services. We believe that by integrating GPA graphs into a website for sub-disciplines students be able to get a bigger picture than using either service in isolation and saving time rather than having to hop between resources.