GradU: gradu.life

Authors: Adrian Gushin, Tiantian Li, Zach Cheng, Chen Xue, Yijun Liu

Section One: Purpose

There are more students applying to undergraduate school than graduate school, and as such, the availability of resources and services aiding students in applying to college are skewed towards this undergraduate demographic. This website aims to alleviate that gap by providing a compiled source of data on past graduate program applications across the country. Particularly, it provides a forum where data typically scattered across Reddit and other such platforms can be organized and presented in a neat, easily accessible manner. Undergraduate users will be able to search amongst profiles of current or past graduate students to see their portfolios and the types of research and activities successful applications include. They will be able to search by both the target university and program. They will also be able to access information such as demographics, statement of purpose, and CV. Users who have already applied to graduate school will be able to use the website to upload their portfolio. They will be able to edit their existing data and also create a profile page with contact details and other general information.

Section Two: Architecture and Organization

The code in this repository consists entirely of frontend and backend files. All the relevant code files can be found in the "src" folder. Upon entering the src folder, note that all of the files that are not stored within a separate sub-folder contain code for the higher-level organization and maintenance of the website. Specifically, the code found here starts the Flask server, (wsgi.py), handles all the URLs and routing for the frontend (App.js), and contains the code for the home page of the website itself (ProfileGalleryMain.js).

All of the sub-folders of src, with the exception of the folder entitled "backend," contain the frontend code that creates and maintains one of the pages or components, such as navigation bar and the footer, used by the website. For example, "ProfileEdit" contains the code for the page that edits an already existing profile, "ProfileForm" contains the code for the page that submits a profile for the first time, and so forth. The main technologies used for these files are Javascript with a ReactJS framework and CSS for styling. While most of these pages are all housed within one file, the login, profile edit, and profile form pages span multiple files. For the login page, the "LogInAll.js" file contains the general framework for the page, while the other files create more specific page elements depending on whether or not the user is signing in or logging in. For the other two files, the code in the file named "header" contains some preliminary code outlining the aesthetic look of the page, whereas the main body of the code is found in ProfileFormFinal.js or ProfileEdit.js respectively.

Below is a diagram of the webpage workflow

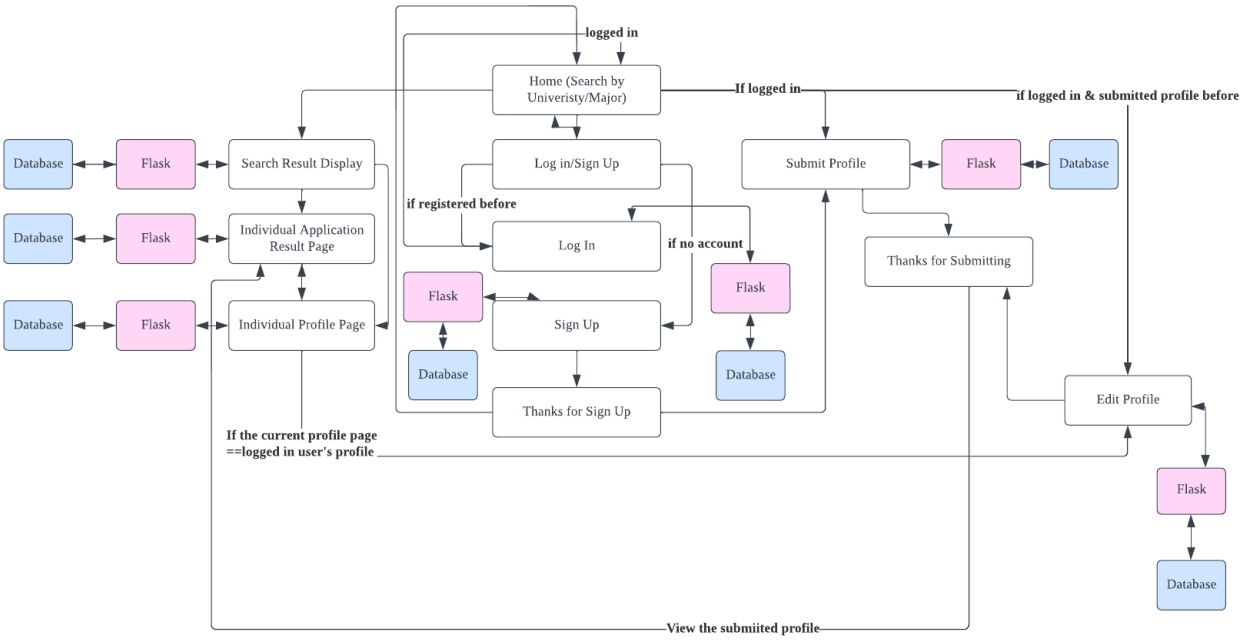

Fig 1. Workflow of the website

With the exception of the wsgi.py file, all of the backend code resides within the "backend" folder. The backend technologies are Flask, Flask-CORS, and SQLAlchemy. Our implementation of the project also used a PostgreSQL database. The file __init__.py sets up the Flask server. The next files, create.py and model.py, interact with the database. Create.py "resets" the database by, upon its execution, dropping all the tables currently in the database and rebuilding them according to the specifications in model.py. Model.py outlines the framework for each of the tables used in the database and their respective columns. Finally, read_reviewer_form.py performs the bulk of the Flask server's operations. It contains all the methods that perform operations like posting profiles to the database, verifying logins, and any other request sent from the frontend.

Section Three: Setup

Setting up the website mostly involves downloading the required dependencies and procuring the technical devices necessary to host the site. The necessary dependencies are ReactJS and NPM for the frontend. The backend requires Flask, Flask-CORS, and SQLAlchemy. These libraries should be installed on a Google Cloud Services (or other such hosting service) instance. Alternatively, the website can be run through localhost for debugging purposes. The final technical requirement is that the user set up their own PostgreSQL database to store the user profiles.

Once these steps have been followed, all that remains is to rewrite the URL-specific method calls inside of the code to match the specifications of the user's system. This requirement

includes rewriting all the fetch calls to the backend to use the URL of the user's new Flask server. All redirect objects returned from the backend should be changed to match the URL of the user's frontend. The user must also, in the backend files, change the URL of the database to access from its current value to whatever the URL of their database is. Finally, the user must create, somewhere on their machine, a directory to store file uploads. They should then change the UPLOAD_FOLDER variable to store a path to this new directory.