General Architecture:

The application is split into two parts, the frontend (client) and the backend (server) The backend is written entirely in Flask, and it uses SQLite as the database of choice. In order to communicate with the database, the backend uses python's SQLAlchemy library to serve as an ORM. Essentially, we establish a series of API endpoints which are then called by the client and retrieve certain information based on those calls. The frontend is built in Vue3 and requires bootstrap for the styling of elements.

Security

Authentication

In order to be secure, the API calls made by the client must be authenticated by the server. This authentication is established during the login process. On the client side, the user first obtains a token from Yale CAS by following the usual login flow. This token is then sent to the server, which verifies it using the corresponding Yale CAS endpoint. During that verification process, Yale CAS communicates the Net Id of the user that sent the token. The server then creates a randomly generated 128 bit client secret, associates it with the given NetId, and sends both the secret and the NetId back to the client. The client stores both pieces of information in session storage, and when the client makes any API calls (for instance, to retrieve a homework assignment or a grade), it must provide both the NetId and the secret that it has been assigned as HTTP headers. This means that the user is given a new token every time they log in, and also, the server does not need to store usernames or passwords.

API

The API is split into two sections, calls meant to be made only by TAs, and those for students. The API calls usually just function as a CRUD app (Create Read Update Delete) but there are a few somewhat unusual ones. Specifically, API calls that sample a param rule or a grading rule. This involves the user (either the TA or the student) writing a python expression which is then executed on the server. The TA writes rules for grading and generating parameters, and the user submits answers which are then evaluated by the grading script.

Startup:

Installation

I would recommend running the client and backend on a single server. You will need to install the python dependencies, which can be done using the pipfile.lock templates given. It is highly recommended to install this code using pipenv. The vue packages that need to be installed are also located in package-lock.json. Once these packages are installed, the backend server should be started by running app.py, using a python version of at least 3.9 (this project was written and tested using 3.9.10). Once that is running, the client can be started using npm run build and deploying as usual. The client is currently configured to run locally, this will need to be changed if you wish to deploy the project in the future.

Backups

In case there is any issue with the database, the backend is currently configured to make a backup of the entire database every twelve hours, storing copies made at most 4 days prior. This should give whoever is administering the software more than enough time to identify errors and then restore from backup if necessary.

Caveat Emptor:

This code has not been tested using a large number of people. Anybody seeking to use this platform should be warned of this fact. In addition, I will not be developing or supporting this project any further. There are known features that I would have liked to add, but simply did not have time. Existing features might be lacking functionality, and the general flow of the website might need some work. As such, people willing to use this project should be prepared to understand and modify the code if necessary, since it is highly possible that some aspect of the website does not work as intended and you wish to modify it.