Team Five Star COMPSCI 320

Github Link: https://github.com/akyeddala/TeamFiveStar/tree/master

CODE OVERVIEW

- Frontend
  - src contains all the frontend pages and logic
  - Index.tsx
    - Creates the root react element and renders App.tsx which contains all the routes to all the pages
  - Reac-app-env.d.ts
    - Contains global variables for user information
  - App.tsx
    - contains all the routes to all the pages, for example “/Profile” returns the Profile.tsx element or the profile page
  - endpointFunctions.js
    - Defines fetch2 and fetch3 (same function) that calls the built in JS “fetch” API call with the given url, method, and JSON body
  - Navbar
    - Easy access buttons for the main four pages of the application that exist on every page, useNavigate is used to navigate to different pages
      - Home - goes to ‘/Home’
      - Login - goes to ‘/Login’
      - join/create class - goes to “/Joinclass”
      - Profile - goes to “/Profile”
  - Pages
    - Each file name corresponds to the react element displayed on their respective page, for example Profile.tsx is the profile page
    - Each file exports a const function that returns the react component
    - Some pages contain async functions that handle the “fetch” calls of the different information handling as needed
      - Response is used to check whether the call was successful, ret is used for the actual body of the response
    - Each tsx file has the line
      - `if(globalThis.userName === null || globalThis.userName === undefined) {return (<Login2/>);}`
      - This line makes sure that a user will only see the login page if they have not logged in yet, unless they are going to the register page
    - Some information is different whether the user is a teacher or not
      - Teacher
        - Can create a class in join/create class
        - can create office hours for a class (if they created the class) in manage classes
      - Student

            - Can unenroll or delete a class
        - Each page contains the useNavigate function from React that load different webpages based on the intended workflow of the App
        - Both '/Home' and '/' go to the same place as intended
    - Styles
        - All the CSS files of all the styles of their corresponding pages
    - Package.json, package-lock.json, tsconfig.json contain all the dependencies and application definitions as well as the scripts
- Backend
    - backendAPI.py
        - Written in python
        - Contains pretty much the entire backend due to python import issues
        - Section 1: Dependencies, Line 1-21
            - Contains the imports used by the backend and settings for the backend when running as a flask application.
        - Section 2: Psycopg2, Line 23-469
            - Contains the functions used to interact with the database along with helper functions that make them work
            - Names are mostly self-explanatory, but a small summary of the function's purpose, inputs, and outputs are located above each one
            - Additional details on individual functions are located in "Backend to Database Functions.pdf"
            - Reference official Psycopg2 and SQL documentation for modifying SQL commands.
        - Section 3: Algorithm, Line 471-557
            - Contains the algorithm and its associated helper functions
            - Student Schedules are stored as 5x12 arrays as well, with a 0 corresponding to an available time, and a 1 corresponding to a busy time.
            - Algorithm fetches all student schedules for a class, and sums up all available and unavailable times. Stores this in a 5x12 counter array (5 days of the week, 12 time slots a day)
            - Algorithm will compare counter to the teacher's schedules. If no teacher's are available for office hours, it will set the counter to unavailable.
            - Algorithm loops through each day of the week, finding the index of the minimum element in the counter (meaning most available time slot). It will set the office hours accordingly.

        - Section 4: Flask, Line 561-938
            - Contains the endpoints that are used to communicate with the frontend.

            - Endpoints marked with “@login_required” can only be accessed if logged in
            - Sessions work via Flask_session, Flask_login, and userAuth.py
            - Data is passed in and out of the endpoints through attaching a JSON object in the request/response’s body
            - Additional details on individual endpoints are located in “backendAPI.pdf”
            - Reference official Flask documentation for adding/modifying endpoints
    - userAuth.py
        - Written in python
        - Contains most of the authentication functions, including register, login, and logout.

RUNNING THE CODE

- Frontend
    - Dependencies
        - Download npm
        - Run “npm install” to install all dependencies
    - Running the Frontend
        - Run “npm start” for development mode or “npm run build” and afterwards “serve -s build” for production mode
        - Connect to 127.0.0.1:3000 and not localhost. Localhost cannot save cookies and thus the login function and most of the app does not work there.
        - Must run on a web server where CORS is disabled, can do so with chrome in the terminal (mac)
            - open -n -a /Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome --args --user-data-dir="/tmp/chrome_dev_test" --disable-web-security
- Backend
    - Dependencies
      Install using pip
        - Python 3.11+
        - Flask
        - flask_session
        - Flask_cors
        - Flask-login
        - Psycopg2
            - Install psycopg2-binary if the regular install doesn’t work
        - Json
        - Numpty
    - Running the Backend
        - Make sure the database is turned on

            - Database is named “database-1”
            - Hosted on AWS
        - In a terminal, cd into the project, should be the folder that backendAPI.py is in
        - Type “export FLASK_APP=backendAPI.py”
        - Type “flask run”
        - Congratulations, it is now running on port 5000 of the computer

TESTING

- Backend
    - Postman: https://app.getpostman.com/join-team?invite_code=1a5bcb8b3b40e44251e0c1ff5ac57a21&target_code=395386630747b21c0310e53001ff888a
        - NOTE: postman tests use the /backdoor/ endpoint to set up conditions for testing. Make sure to uncomment the branch before testing. The flask app only implements code changes on reboot.
        - Main Tests Collection contains the tests mostly separated by endpoint along with the requests that set up the conditions they use
        - Backdoor Functions contain the original copy of requests used to set up testing conditions, generally all requests that come from this folder have “Backdoor” as the first word in their title
        - Jared_Tests contain tests that Jared used for the algorithm. I don’t know if they are a full test or just used to manually check some elements
        - Uses email “TESTDUMMY@umass.edu” and class ID 65535 and thus will delete any users/classes that use them.
    - buildDatabase.sql
        - SQL file that sets up the tables in the database.
- Frontend
    - Best testing practice would be to run the program and use the application as directed, inspecting the webpage with the console open to look for any failed API calls or unintended workflow (went to the wrong page, wrong page loaded, element is not loading right, etc.)

EXTERNAL CODE SOURCES USED

- Stack Overflow

KNOWN BUGS/LIMITATIONS

- Does not work on localhost, localhost doesn’t save cookies which is where the session is stored, must use 127.0.0.1:3000
- Does not work on a browser where CORS policy is enable, can use the following line on MAC to open a chrome web browser without CORS

    -
      ```
      open -n -a /Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome --args
      --user-data-dir="/tmp/chrome_dev_test" --disable-web-security
      ```

- The backend uses the master account to interact with the backend, leaving no damage control in the event of proper SQL injection