TAPS Source Code Documentation

Components:

- **Frontend**
    - This project is done entirely in React Js front-end for the security of the data that is being handled

Implementation:

As stated, TAPS is done entirely in ReactJS. Tailwind and bootstrap are combined to help with the design of the front end application.

Specific Details of Each Part of Code:

- **App.js:**
    - States for each component are created to keep track of specific data:
        - all_TAs: Meant to hold information about all TAs schedule, current classes, former classes and grade that was achieved in all
        - classList: Meant to hold information about each class that is being offered for the upcoming semester
        - Predet: True/False variable that determines if user has entered all necessary csv files
        - Run_alg: True/False variable that determines when stable marriage finishes
        - Predeterminations: meant to hold information about which TAs have been predetermined to TA which class
    - Return statement renders the input elements. It consists of a parent div element and two child div elements, which contain other child components.
    - The first child component is FileInputs. It receives props that correspond to the all_TAs, setAllTas, classList, setClassList, and setpredet states.
    - The second child div contains conditional rendering.
        - If predet is true, it renders the Predeterminations component, which receives props corresponding to courses, tas, setpredet, setRunAlg, and setPredeterminations.
        - If run_alg is true, it renders the StableMarriage component, which receives props corresponding to setProgress, courses, tas, and predeterminations
- **AllClassesCSV.js**
    - This component was made to handle the user uploading a CSV that contains information about every class that is being offered for the upcoming semester
    - Only csv files are accepted. A list of required headers is created. The function then checks to make sure that each of those required headers are present: if they are not, the user is notified and the file is not accepted.

    - Once an acceptable file is uploaded, the code goes through each row of the csv and extracts the necessary data for each class that is being offered, including the time it is offered and the number of students that are enrolled in the class
    - The extracted data is then stored in a dictionary, and that individual dictionary is stored in a larger dictionary that holds every single class
    - The user is then allowed to upload the next file
- **InClassNeccCSV.js**
    - This component handles the user uploading a CSV that contains information about each class that has a specified value for if the TA needs to be present in the class or not - not all classes have a value for this so a default value of "No Data" is used for those without one
    - This component follows the same input validation methods as AllClassesCSV.js
    - If a class has a specified value for if the TA needs to be present, that value is appended to the list that contains all the classes that are being offered for the semester
    - Once this file is uploaded, the user is allowed to upload the next file
- **TAListCSV.js**
    - This component handles the user uploading a CSV that contains information about each TA that is being considered for the upcoming semester
    - The same input validation is used
    - When a valid CSV is uploaded, the code goes through each row of the CSV and creates a dictionary that contains information about that TA, including the amount of hours that they need for the upcoming semester
    - The user is allowed to upload the last file after this one is added
- **AllTAsCSV.js**
    - This component handles the user uploading a CSV that contains information about all people that are eligible to TA classes at the University
    - Input validation is the same
    - Each TA is stored as a dictionary, that holds information about every single class they have taken or are taking currently
    - Once this file is uploaded, the user is allowed to enter predeterminations
- **CreateEligList.js**
    - This component takes in the list of the TAs that are being considered for the semester and the list of classes that are being offered for the semester and determines which TAs are able to TA for which classes
    - First, a list of classes that the TA could have taken in place of the actual class itself is initialized
    - The function convertStandardTimes converts standard time to military time because the class times are listed in standard form on the class list but they are in military time on the TA list

    - The function handleEligibilityList starts by iterating through each class that is being offered for the semester
    - Next, each TA that is being considered is looped through
    - Then, a third loop is run that goes through each class that the TA has taken in the past
    - If the TA has taken the class that is being evaluated, or taken a class that qualifies them to teach that class, a variable is set to true
    - If the class requires the TA to be present in class, and the TA is available to do that based on their schedule, another variable is set to true
    - If both variables are set to true, then the TA is set as an eligible TA for that class
    - A dictionary of every class that is being offered is created and it contains a dictionary that holds a list of every TA that is available to TA that class
- StableMarriage.js
    - This component takes in the courses (as classList), tas (as all_TAs), and predeterminations const state variables and runs the Stable Marriage Algorithm to output working TA Assignment Schedules.
    - Our knowledge and implementation for the Stable Marriage Algorithm comes mostly from these two videos Stable Marriage Problem - Numberphile The Stable Matching Algorithm - Examples and Implementation So refer to them if you have any questions about the algorithm specifically, however since we are dealing with a many to many problem and Stable Marriage is not a many to many algorithm, originally some modifications needed to be made.
    - The algorithm Starts by giving each TA a preferred hierarchy of Courses and each Course a preferred hierarchy of TAs for the Stable Marriage Algorithm all of which was handled in the createCourseTas Function. We used some hierarchical sorting algorithms that are in the utils/PreprocessingFunctions.js file to make these lists, but for Courses we found the best results in randomizing their preferred TAs. The createCourseTas function also sets the remaining hours for each course and ta based on the files inputted.
    - Everything in the Stable Marriage component goes through the run_all function. The function works by running the StableMarriage algorithm x amount of times set by the ITERATIONS variable with each iteration checking to see if the new schedule is better than the old and if it's not trying again. The function tries to keep the working matches of the current best schedule and optimizes it from there, however if after x amount of time set by the AUDIBLE_COUNT variable it doesn’t make any improvements then it restarts from scratch. Before restarting however the function stores the current best version of the schedule with its complications and remaining tahours dictionary as a list in the history variable.
    - Schedules are determined to be better or worse than one another by the validate_schedule function which keeps a list of all working and not working

matches as well as if any course has more than 3 TAs assigned which is not preferable. The complications are stored in the complications variable and if the schedule is considered the best then it is stored in the conflicts variable.

Algorithm Comments:
It is worth noting that the runtime of this algorithm is determined by the number of iterations that the algorithm runs through. Increasing the number of iterations increases the runtime but also increases the effective accuracy of the eventual output. We have used a base of 10,000 iterations that has resulted in a runtime of one to two minutes. Letting the algorithm run for longer would conceptually create a high probability of a positive output.

Frontend Comments:
We have two repositories– the second one can be found here. Our initial plan was to merge this repository with the current one, but we didn't have enough time to do so. This repository contains a lot of standardized, scalable frontend work.