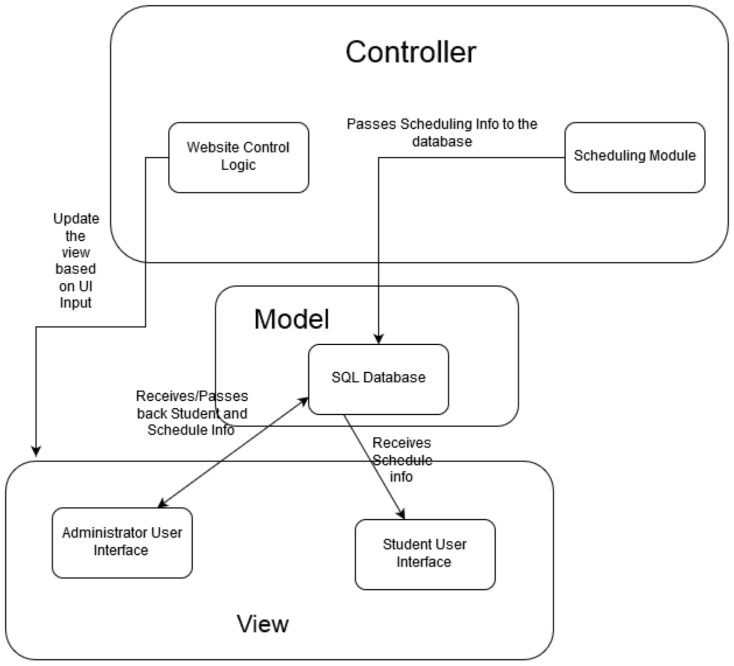
Design Document

Architecture

Our design follows the ModelViewController paradigm. Our model consists of a backend database done in SQL Alchemy. This database will send data to both the front view and the controller in the form of JSON objects. Our View will consist of several different user interface pages including a calendar view done using FullCalender and form pages done using Flask. Our controller will consist of a scheduling engine as well as some control logic in order to control the website logic. Additionally, our design will also follow the Client-Server architecture where the Client is the front end application which will be fairly minimal and the server will perform most of the work of the application. The server will perform the database storage, management, scheduling, and the control logic for the application.


Controller
Website Control Logic
Passes Scheduling Info to the database
Scheduling Module
Update the view based on UI Input
Model
SQL Database
Receives/Passes back Student and Schedule Info
Receives Schedule info
Administrator User Interface
Student User Interface
View

Decomposition: Modules

User Interface Module

- Student User Interface
    - The students should be able to log into the site via their ONYEN and password.
    - The students should be able to access the master schedule with the hours that they are scheduled for.
    - The students should be able to set their availability via a calendar. They should be able to select Prefer to Work, Can Work, and Cannot Work.
    - Once the availability has been completed then the data should be sent to the database module of our application.
- Administrator User Interface

    - The administrator should be able to log into the site via their ONYEN and password.
    - The administrator should be able to view and edit the master schedule and the changes that they make should be sent to the backend database module that will deal with storing the changes that are made.
    - The administrator should be able to see information pertaining to each individual student including ONYEN, email, PID, and amount of hours and money allocated. This data should be populated via the database module. They should also be able to edit their employees pay and hours.
    - The administrator should be able to edit the scheduling constraints and run the scheduling algorithm in order to generate a master schedule via the scheduling module. The generated schedule should be viewable via this page.

Database Module:

- Algorithmic Information
    - The master schedule should be stored in the database and should be editable via the scheduling module and the administrator's view.
    - This data should be passed to the FullCalender when it needs to be displayed and changed.
- Student Information
    - Each student's information will be stored in the database. Exact database structure will be determined when we get to the creation of the database.

Scheduling Module:

- User Class
    - A user will be able have a name, PID, email, and ONYEN, as well as a type code which will specify what type of user they are including whether they are a returning student or not or if they are an administrator or not.
- Employee Subclass
    - An employee will have a copy of their individual schedule and they will have an array signifying their availability at all times during the schedule.
    - An employee should be able to schedule themselves meaning that they take add themselves to their personal schedule and they remove their availability for times that they are scheduled at.
- Location
    - A location will have a 3D array that it will populate incrementally in order to generate the master schedule. In addition, it will have a list of employees that it can add into the schedule, an array of timeslots that represent how many employees are needed at each time represented by a positive number or zero, and it will have a need array that will have the relative urgency with which each timeslot will need to be scheduled.
    - A location should be able to add an employee to its potential candidates list.

    - A location should be able to calculate the need of each individual timeslot and populate its need arrays given an array of potential employees.
    - A location should be able to find the greatest need in the need array and return where in the array that timeslot is at.
    - A location should be able to then take the greatest need and add that location to the master schedule.
- Schedule Manger Class
    - A schedule should maintain information on the number of shifts in a day, the shift length in minutes, and the time that shifts start. It will also maintain a list of all the locations at which they need to schedule people for.
    - A schedule should be able to add a location to the list of locations and verify that adding the location will not cause conflicts with other locations.
    - A schedule should be able to go through and schedule students for each of the open timeslots across all locations.
    - A schedule should be able to compute a rating for how optimal the schedule that was generated and return this value.

Code for scheduling module available at: https://github.com/WritingCenterScheduler/Engine

Design Decisions

- Using FullCalender to display the calendar instead of an HTML table due to the fact that this provides an easily editable calendar and a more aesthetically pleasing UI.
- Using Flask to provide an architecture for out MVC website due to its lightweight architecture, which is optimal for a small application like this.