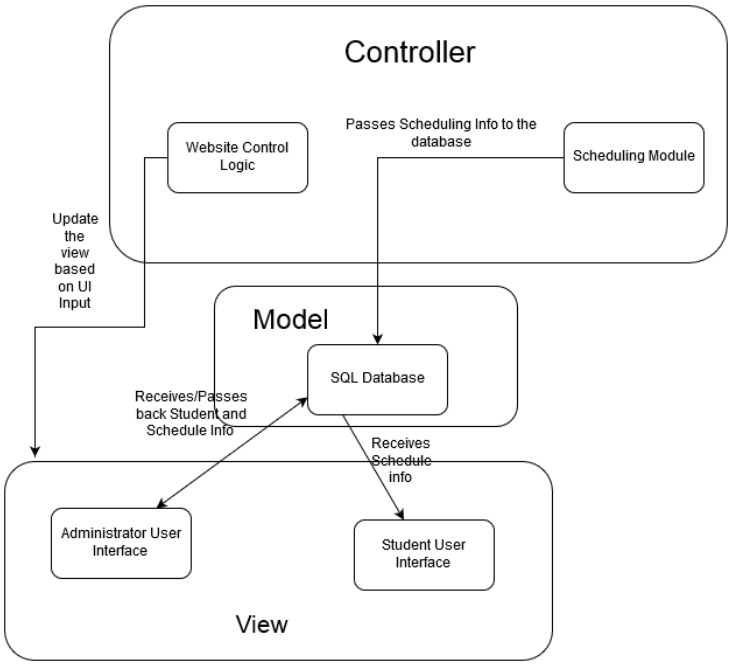
Design Document

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.



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. Additionally, they should be able to add administrators if they should need to in the future. This will be done through the use of PIDs.
- 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.
- User Information Table
  - This table will store information pertaining to the different users of the database including their student information and the
  - This will have seven fields:
    1. Last_name—This will be a required field that stores the last name of the student as a string
    2. First_name—This will be a required field that stores the first name of the student as a string
    3. Pid – This will be an unique field that be stored as an integer.
    4. Email—this will be stored as an email field and will contains the students email
    5. Typecode – This will be a string field that will store a type code that lets the application know what type of user the current user is defined as either non-user, user, or administrator.
    6. Availability – this is a dictionary field that stores a dictionary containing the availability of each student at each time slot a zero would refer to unavailable, a one would mean that the student is available to work, and two means that it is a good time for the student to work.
    7. Resolution_minutes – this is a integer field that stores the amount of time that a timeslot spans in the availability array (usually 30 minutes in our case)
- Location Table

    - This table will store all of the information as it pertains to different locations in the database including their names, hours and the number of people needed at each time slot
    - This table will have 6 fields:
        1. Name—this will have the name of the location stored as string for example Writing Center Downstairs.
        2. Code—this will be a integer that serves a unique identifier for each location that will be used to distinguish between different locations
        3. Open_at – a required string that will identify the time that the location opens at for example 8AM.
        4. Close_at – this will be a required field that identifies the time that the location closes for example 5PM
        5. Requirements – This will be a dictionary field that will store an integer for each timeslot that serves to identify the amount of people that need to be scheduled for each location at each timeslot.
        6. Resolution_minutes – this is a integer field that stores the amount of time that a timeslot spans in the requirement array (usually 30 minutes in our case)
- Schedule Table
    - This will be used to represent a final or partially complete schedule for each location
    - This table will store information pertaining to the different schedules each of which will be stored in the database.
    - This will have three fields:
        1. Sid—this will be a randomly generated 4 digit code which will be used to uniquely identify the schedule and allow for it to directly referred to. This field should be both unique and required.
        2. Data—this will store the actual schedule itself. It will be list containing all of the timeslots in the schedule and what users are scheduled for each individual timeslot.
        3. Created_on—this will store the time that the scheduled was generated.

Code for Database Module is at:
https://github.com/WritingCenterScheduler/application/blob/development/app/models.py

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.