SLST Precinct Rota System

This section is pure planning and is not part of the final documentation.

# Project Concept

A precinct rota system, allowing for the precinct monitors of the SLST to manage their responsibly on an easy to use online system.

## Technologies

* Python 3.6 for backend/webserver
* (html, sass, js) skeleton for frontend design
* Jinja2 for templating

## Site Sections

### Administration

Login

User Management

Password Reset

### Precinct

Timetable setup (times, days)

User allocation

Timetable preview

Signing in for precinct

### User

Login

Account Details

Password Reset

Unavailable Days

# Advice™

## Travi

* Relative responsibilities of the personal involved, what do they do?
* Calculate hours of shifts and reliability
* message boards regarding work to be completed
* Bookings for SLST services and assignment of team members

## Class

* allow students to enter a reason for unavailability
* Have teachers be able to delete a student's session if they have evidence that the student faked the login or logged in and left
* rank students by calculated reliability - perhaps notify teachers if students reliability drops too low
* do graphs of stuff - visual presentation of student reliability
* Priority levels to influence prefect selection algorithm
* Punishments for those who are unreliable / send reports to teachers
* calculate percentage chance of not turning up and punish beforehand like in minority report
* Role system
* flexible precinct times, don’t restrict precinct times to just timetabled breaks
* Work hour limits
* integrate with sims to explain lateness/absence and avoid unnecessary sanctions
* update the privacy policy

SLST Precinct Management System

Matt Cowley 4807 – Royal Grammar School High Wycombe 52423

# Analysis

## Objectives

* Allow students to indicate days they cannot complete precinct duty
* Allow students to sign in and out of their current precinct duty and for this to be tracked
* Display an easy to understand timetable of duties
* Allow administrators to define the precinct duty sessions for the week
* Allow administrators to view when students signed in and out
* Display to administrators the attendance levels for individual students on the precinct, the student’s reliability
* Display to administrators the duration each student stays for their precinct duty, percentage of defined precinct duration
* Provide an authentication system for administrators and students
* The system will automatically generate a rota based on the sessions defined by the administrators and the unavailable days as marked by students

## Problem Identification

(a) Describe and justify the features that make the problem solvable by computational methods. Prove that the output that is required, e.g. a prefect rota, a receipt or a game win can be achieved with a list of known inputs and formulas, show that any additional complexity/variables/precision, etc. will not affect the output that much/customer is not bothered about that extra stuff.

The SLST is a team within the school, the Stage Lighting and Sound team, responsible for all services provided on the stage and general technical services around the school site for events. This team is run by a group of students will a staff member lead. As part of being a senior member of the school, senior students are given precinct duties which they must monitor on set days at break or lunch to ensure other students are behaving properly in their precinct area.

Most precincts use a rota of some kind to organise how students run the precinct area but the SLST does not yet have a rota of any kind. Due to this, currently the SLST struggles to manage its precinct. Often multiple students assigned to the precinct are there at once and at other times there is no senior student present which can lead to issues with behaviour in the area. Having no responsible student in the stage area can be very dangerous due to the high voltage power supply we have and the heavy equipment at heights. This lack of organisation and possible risk is most likely due to the lack of an official rota being created.

*(b) Explain why the problem is amenable to a computational approach. Why using your platform the best way of solving this problem, why not do it with pen, paper or a calculator.*

By having the web based solution implemented the precinct rota can be easily managed whenever and wherever allowing for a versatile and portable solution. If hosted on the internal network, all relevant staff and students could easily access it from any device to ensure effective organisation and management of the SLST precinct. Having the rota automatically generated by the system based on days the students are available means the responsible staff do not have to waste time attempting to organise the rota themselves.

Further, having the advanced web based system will allow staff managing the precinct to quickly and efficiently monitor the performance of all students responsible for the area and to view their reliability and attendance when scheduled. The system will allow the staff to view how often a student is late for a duty and additionally how often they fail to attend. This will allow the staff to ensure that the precinct area is being monitored by the students correctly.

## Stakeholders

(a) Identify and describe those who will have an interest in the solution explaining how the solution is appropriate to their needs (this may be named individuals, groups or persona that describes the target end user).

There are multiple parties who will be invested in this project, its completion and hopeful success. These include the students who are responsible for being on duty in the SLST precinct, the staff in charge of the SLST precinct and ensuring it is well looked after and also the students who use the precinct area as their place to chill out.

The staff will benefit from the project as it will provide them with an easy means of understanding their precinct and the students running it. They can view how reliable and useful all the students on the precinct are.

For the students running the precinct it will be useful to them as they will be able to organise their lives within the school better and know exactly when they are expected to be on duty in the precinct area. This will allow them to have more free time at breaks and lunch times as they will not continually be on duty in the precinct area.

Everyone should benefit from the system being implemented as it will provide a way to ensure there is always a responsible senior student monitoring the precinct area when it is in use without there being an excess of senior students on duty.

## Problem Research

(a) Research the problem and solutions to similar problems to identify and justify suitable approaches to a solution. Have you looked at the competition and commented on the good and bad features of each? E.g. 3-4 different versions of Snake. What is in common and what is different between these solutions? What can your program offer that competition can't?

(b) Describe the essential features of a computational solution explaining these choices. Having done your review of existing products, what ideas would you take on board, justify with an example of something more your customer can achieve with this feature.

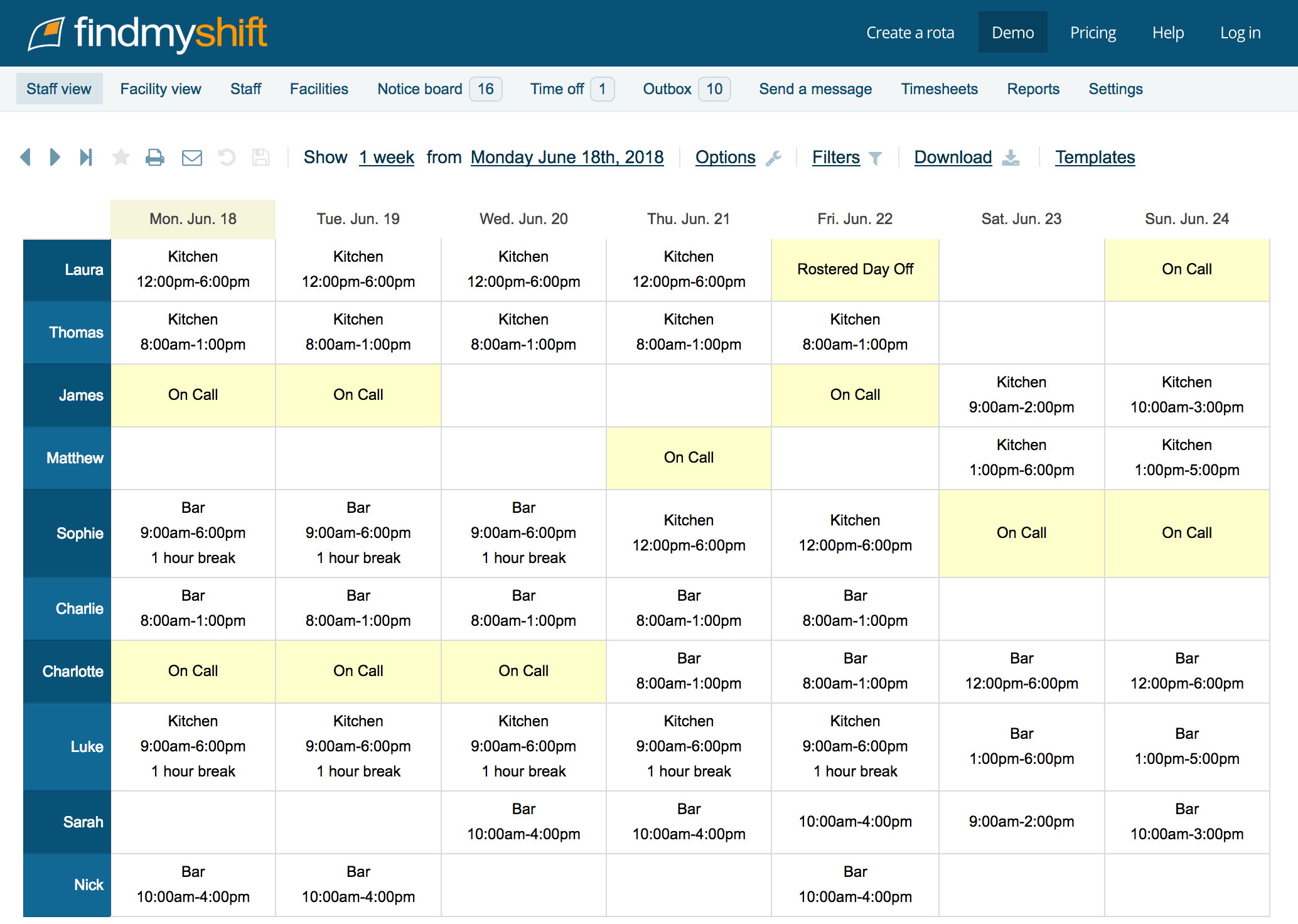
### Excel

One option that was viable instead of this system would be to use Excel (or any other spreadsheet software) and create a simple timetable.

This however has drawbacks as it becomes hard to share with all the students and staff who need access to track their duties. Further, it would be very hard to control who has access to what parts of the spreadsheet which could lead to students changing the rota without permission or easily faking their attendance for a duty. This solution would however provide both the staff and students with an easy to view rota for the precinct which would be a key objective of this proposed solution whilst also introducing far more advanced methods for determining the reliability of students as they sing in and out for duties.

This solution may be possible to achieve and use with very complicated VBA embedded into the document but this is not a practical solution. The project aims to make the precinct management process easy and simple unlike this.

### FindMyShift

Additionally, online solutions such as FindMyShift (<https://www.findmyshift.co.uk/>) exist but these are mostly business orientated and are not aimed at simplistic rota systems. With this software, it is designed to be scheduled week by week and not aimed for recurring weekly rota like we would want for stage.

These online systems are almost all closed source and so it cannot be modified to our exact needs and to perform the data analysis on students’ sign in/out times as is proposed in the solution.

This software also has costs associated with more advanced versions of it to unlock all features. Therefor creating our own system will eliminate subscription costs and will mean it can be customised to our aim, a recurring weekly rota with analysis of attendance.

*(c) Explain the limitations of the proposed solution. Given the time you have had and your level of skills (as well as the limitations of the customer), which great ideas/features would you NOT be able to implement? This will connect to evaluation later on, where you will justify why it might be worth adding later.*

Unlike the FindMyShift solution, this solution will not have the ability to add holiday days to the rota in advance, nor will it allow for sick days to be booked in advance on the rota. The rota in this solution will be fixed unless regenerated or edited by a staff member on the precinct. This is easier for both staff and students to manage and understand within the context of the school environment.

Further, this solution will not provide the vast array of settings and options that FindMyShift has as this solution is bespoke to the SLST precinct and will be customised to work best for this as it is written.

## Proposed Solution

*(a) Specify and justify the solution requirements including hardware and software configuration (if appropriate). Provide hardware and software requirements, e.g. administrative access to registry, internet connection. Some of your tests later on will include running your program on the computers of different spec and noting the difference in time it took to do a task, e.g. searching or writing to disk.*

Creating a web based precinct rota system, allowing for the precinct monitors of the SLST to manage their responsibly on an easy to use online system and provide staff of the precinct with an easy to view system for if the precinct monitors are handling their duties correctly.

The solution will allow students to indicate days they are unavailable for the precinct rota which will then be reflected when the system automatically generates a new timetable for the precinct duties. Further, the system will allow the students to sign in and out for each assigned duty period on the timetable and this will be tracked and analysed by the system to provide the staff with insights into the reliability and performance of each student assigned to the precinct.

Additionally, the system will provide a staff interface where they can view the performance of the students as well as having full control of the precinct rota. The staff view will allow assigned staff to edit the rota should they need to do so as well as generate a new automatic one based on the unavailable days as marked by students. The system will also provide detailed insight into each student, showing their attendance record for assigned duty periods as well as their average tardiness and general reliability.

The system will use Python 3.6 as the backend for the web based interface. The Flask python package will be used to operate the base of the web server that the user will interact with on the front end. The templating mark-up Jinja2 will be used as this is the templating language for web pages supported by Python and Flask. On the frontend, SASS will be used for writing all styling which will be automatically compiled to CSS for serving. The pre-made CSS framework Skeleton CSS will be used as the base for all page styling as this provides a lightweight and clean layout system for use on webpages. For interactivity on the frontend, such as calendars, the JavaScript package jQuery will be used as this provides a huge number of features in an easy to use method format.

### Objectives

*(b) Identify and justify measurable success criteria for the proposed solution. A table of 60+ objectives (split into usability, performance, reliability (validation), and maintainability (how easy to change the code later or add new features) and how you intend to prove you will have met these objectives.*

* Database
  + **[primary key] *[foreign key]*** *{default value}*
  + Table to house accounts
    - Int: Id **[primary]**
    - String: Username
    - String: Password
    - Datetime: Date created
    - Int: Auth level (student/staff)
    - Bool: Disabled (disables login ability) *{false}*
  + Table to house password resets
    - Varchar(255): Reset token **[primary]**
    - Datetime: Expires
    - Int: User id ***[foreign]***
  + Table to house the rota layout
    - Int: Session id **[primary]**
    - Int: Day
    - Int: Start time
    - Int: End time
    - Bool: Archived (used for attendance to track old rota layouts) *{false}*
  + Table to house student unavailable days
    - Int: User id ***[foreign]***
    - Int: Session id ***[foreign]***
    - String: Reason
  + Table to house students assigned to rota
    - Int: User id ***[foreign]***
    - Int: Session id ***[foreign]***
  + Table to house student attendance
    - Int: Attendance id **[primary]**
    - Int: Session id ***[foreign]***
    - Int: User id ***[foreign]***
    - Datetime: In time
    - Datetime: Out time *{null}*
* Student Portal
  + Auth
    - Login
      * Success (redirect to rota view)
      * Failure
    - Logout (redirect to login)
    - Password Reset
      * Set password (requires token in get)
        + Invalid token error
        + Success (redirect to login)
  + Rota view
    - *Nb: Will display the current configured rota with markers for when students are assigned. (Highlight sessions for current student).*
    - View current configured rota
    - Mark unavailable days
  + Sign in for duty
    - *Nb: Warn if student is late.*
  + Sign out of duty
    - *Nb: Automatically sign out at end of duty time if student forgets.*
* Staff Portal
  + Auth
    - Login
      * Success (redirect to student view)
      * Failure
    - Logout (redirect to login)
    - Password Reset
      * Set password (requires token in get)
        + Invalid token error
        + Success (redirect to login)
  + Rota View
    - *Nb: will display the same rota preview as in student portal without any markers.*
    - Edit rota sessions
    - Edit students assigned to sessions
      * *Nb: Warn if student is marked as unavailable on manually assigned session*
    - Generate automatic rota configuration
  + Student View
    - *Nb: List of students displayed with average reliability displayed next to them. Students act as links to dedicate pages.*
    - Individual View
      * *Nb: Shows students unavailable days, reliability and performance summary.*
      * Attendance details
        + *Nb: Paginated!! Shows list of all attended sessions. Indicates if they are sessions on the current rota.*
        + *Nb: Shows time session starts and ends as well as time student signed in/out. Shows how many minutes late they arrived and minutes early they left if applicable.*
      * Account details
        + Edit account
        + Delete/disable

*Nb: Sets disabled flag on account.*

*Nb: Removes all assigned sessions from student.*

* + - Create new student account
  + Staff View
    - *Nb: List of all staff accounts on system.*
    - Edit account
    - Delete
      * *Nb: This can just delete; nothing is using staff id as foreign key.*
    - Create new staff account

# Design

## Solution Decomposition

*(a) Break down the problem into smaller parts suitable for computational solutions justifying any decisions made. Create a schedule of which parts of the program will be done in what order, e.g. database setup first, text interface second, login/out, all the searching, deletion, sorting, GUI , validation, etc.*

1. Create the base Flask application
2. Setup basic templating for the project
3. Create authentication module (staff + student access)
4. Create basic base templates and styling for the site
5. Create required database models for student rota (sessions, assignments)
   1. ERD
6. Create student rota view mock-up
7. Student rota routes and generation
8. Create db models for unavailability
   1. ERD
9. Create views & post routes for unavailability
10. Create db model for attendance
    1. ERD
11. Create routes for signing in and out on attendance
12. Create view and functions for attendance report on student portal
13. Design and create attendance overview page for student
14. Create password reset functionality for student
15. Begin staff controller
16. Staff account management
    1. Individual account edit
    2. Account list
17. Create new account
    1. Test inputs
18. Delete accounts
19. Rota
    1. Current full rota view
    2. Edit button for each session & new session button
       1. Test inputs
    3. Edit page allows users to be assigned or unassigned
    4. Show student unavailabilities on edit page --
    5. Auto generate button on rota view
20. Test disabled accounts
    1. Fix routes that fail to check for disabled accounts
21. Attendance report
    1. Overview per session
    2. Overview per student
       1. Per student per session
22. UI improvements
    1. Filter staff account list by not disabled, staff only, students only
    2. Make everything easier to understand, general ui tidy

## Development Log

### Creating the base Flask app

To begin the process of making the SLST rota system, a base Flask app had to be created.

This started off with creating the basic folder structure that will be used by flask, including the templates folder for all the webpages and the static folder where assets used by the webpages will be kept. Additionally, there is also a folder created called assets where all the un-compiled styling will be kept and modules where the Flask page modules will be kept.

### Issue with Jinja2 templating

During the setup of the jinja2 templating, testing a basic index page at the root of the Flask app site, the jinja2 engine which generates all the webpages from template files, was unable to detect the default templates directory correctly and was raising an error as such as it was unable to therefor locate the template set to be used for the index route.

This issue was resolved by implementing a custom jinja2 loader for the Flask app which was set to provide an absolute path to the templates folder being used for this project. This absolute path was created using the os module to ensure it worked cross platform and would base itself off of the location of the current file to ensure portability.

### Setting up authentication

To get started with making the Flask app functional, I began by writing a basic authentication module that will allow all users of the system to login. This was written in a folder called auth in the modules part of the app.

The auth folder contains three key python files; A controllers file which contains the Flask blueprint and routes for the auth module which are loaded into the main Flask routing controller for the website. A forms file which simply contains the python definitions for the form being used to login which is written using the Flask-WTF and wtforms libraries. There is also the models file which contains the database models used in this module, in this case the user table.

### Beginning styling

Once the basic authentication was setup, I could then begin to work on styling the website and testing it out on the new login page. This started with creating the basic folder structure for the scss styling which would later be compiled to a singular css file for use on the site.

The base of the site’s styling was the normalize and skeleton css frameworks which are available and free to use on the internet. Normalize is a large css file that aims to ensure that all standard html elements are styled the same basic way across all browsers so that your custom styling on top looks the same everywhere. This is a great framework to use and is a requirement for using skeleton css to ensure it works properly. Skeleton CSS is a basic framework for web design that provides a simple grid layout and some other really basic styles for stock html elements to provide the designer with a clean base to work from and a grid system to make use of. These two frameworks were put in the framework sub folder of the scss.

Once the frameworks were set up, I began writing the basic global styling that will be used throughout the site. This comprised of a few component files; base contained styles that will provide the base of the site on all pages, footer provides the styling that will create the footer used in the base template of the site, similarly with navbar for the navbar styling. There are also global and overwrites files which provide global variables used in the style files and overwrites which are css snippets loaded in last to give them the most selector power in the DOM of the site.



An issue with the styling that I soon encountered and hadn’t thought about was that as the site was using a dark background, the text was white. This was fine for the majority of the site but on the login forms the inputs were given their stock white backgrounds which made reading any input almost impossible. This was quickly fixed by setting a different text colour for the inputs but was something I forgot to deal with when initially setting up the styling of the site.

### Student rota view

The key part of the site is the rota view provided to both the students and staff. To begin this process a frontend only mock-up of what the rota view would look like was created. This has no link to any database tables or backend logic and was simply to test layout and design.

The concept of the design here is that each day is shown and within that each session is shown with a simple start and end time along with the student assigned. There is a highlight class created which is shown in two rows here that will act as a highlight when the student is looking at a view of the rota that shows their sessions and others to ensure their own sessions are distinct from the rest.

With that sorted, the Session database model was created which holds all the sessions that will be in the rota. This was put in the student/models.py file similar to how auth was set out. Once the session model was defined, the assignment model was created which links a user to a session in the database. This was formed of two foreign keys also acting as a composite primary key.

These models can be shown in a simple entity relationship diagram. The assignments table acts as a simple linking table between the other two.



Once the rota view was setup, a basic student portal homepage was created which at this time simply displays the next assigned session for the student currently logged in.

### Student unavailable days

Now that the basic rota setup has been done, the next thing in the student portal to work on is how the students will mark the sessions they are unavailable for. A simple display of the full rota is shown on the unavailability page but instead of showing who is assigned to it, it simply states if the current logged in student has marked themselves as unavailable for the session and offers a button to update their unavailability.

The edit page then offers a simple checkbox for whether the student is unavailable and a required reason input if the student does mark themselves as unavailable which staff will be able to see when creating a rota layout with assignments.

Now that this is done, the entity relationship diagram from earlier can be updated to also include the new unavailabilities table that had been created.

### Issue with navbar styling

When the new Unavailability link was added to the site’s navigation bar to be displayed when a student user is authenticated and logged in, there was an issue with the current design on the navbar that was discovered. The design of the dropdown menus caused the next item to be pushed along the full width of the dropdown and not just the dropdown title. After some changes in the css, removing some flex elements and using standard block ones, the dropdown behaved correctly and I could move on.



# **Before**

# **After**

### Issue with updating unavailability

An issue that was overlooked whilst I implemented the ability for a student to mark their unavailability was that they shouldn’t be able to mark themselves as unavailable on sessions where they have an assignment to that session. Doing this could lead to confusion with both students and staff when they find students assigned to sessions they are marked as unavailable for.



Student assigned to session:

Student able to update unavailability:

The fix for this was to implement a simple check when displaying the unavailability table data as such:

Further, to ensure there was no way to get around this UI only block, a check was also added to the edit route directly to ensure that a student would not be able to access the unavailability for an assigned session.

### Archived sessions displayed throughout student portal

Due to an oversight and a bit of forgetfulness, no checks were implemented throughout the student portal backend to hide archived sessions from view. Removed sessions from the layout by staff are marked as archived and not actually deleted from the database to ensure integrity in tracking old attendance of students if the session layout is changed.

However, as no archived sessions had been created up to this point, I had completely forgotten they were a thing. When creating an archived session in the database (Sunday 0:00 – 5:00) it was clear to see no checks had been written as it was displayed on the full rota view.



Again, however, this was simply fixed by implementing a filter for all Session queries and similar checks wherever sessions were referenced but not directly selected.



Once these checks were implemented the rota views and other views throughout the student portal dispalyed only the active, current rota sessions and not the archived ones.

### Beginning attendance tracking on the student portal

To begin the attendance section of the student portal, the attendance model was created in the models.py file. This contains a unique primary key id for the attendance so that it will always persist and is not reliant on anything else. The model also holds the user and session ids to track who and what the attendance record is related to. Alongside those, there is the in and out times. The in time is required and defaults to the current utc time stamp when a new instance is created and out time can be null until updated.

With this new model created the entity relationship diagram for the project can be updated.

Once the model was created, the route for signing in could be created alongside the route for signing out. These will be shown as buttons on the student portal index page.



### Issues with attendance

Due to an oversight in the development on the attendance sign in route, there was not a check in place to see if the user had already signed in for the current assigned session.

This led to the ability to end up with multiple attendance records on the same day for the same session as the student user could sign in multiple times.

This was fixed by making use of the quick utility function that had been created for the out route and for the student portal index, which returned Booleans for if the student had signed in and/or out for the current assigned session. Once this was added to the in route, the student user could only sign in once per assigned session.

A further issue that I soon realised whilst playing around with the attendance system is that a student could forget to sign out which lead to an infinite attendance when calculated upon. To counter this the time that the assignment would end is stored in the attendance model so a simple or comparison can be used when fetching the out time from the model. As the original out time from the session will be stored, it makes sense to also store the original session start time so that if the session gets updated, the attendance can still be tracked to the session times when the attendance occurred.

Alongside this, I also realised that a student could simply not sign in which would result in no attendance record being created at all. To counter this, I decided to add a created date to the assignment from which the backend could track forward each session where the student should attend. A date attribute was also added to the attendance record to make selecting it easier.

### Attendance report

With the basic attendance signing in and out completed, a general attendance page on the student portal can be created. This will act as an attendance report that the student user can view.

The page will give the student an overview of their general attendance and punctuality as well as a breakdown per assignment on their user. On the left of the page a quick summary of overall attendance and punctuality will be displayed with a red/green bar to visually represent attendance. On the right of the page there will be a large breakdown table that will show basic stats for each assigned session in the student’s rota.

### Password reset

With almost all of the student portal created, the final part was account management for the student and specifically the password reset. However, as the site does not store the email for the users, providing a secure password reset was not possible.

Therefore, if a student forgets their password, staff will be able to reset the password from their management panel. However, when a student is logged in, they may change their password through their own account management page.

The verification for the password update page is provided by HTML5 with the required attribute on all three fields. In the backend, there is then further verification, checking that the new password field equals the new password confirmation field and that the old password field equals the current password for the user. If these verification checks pass, the password will be updated.

### Beginning the staff portal

To begin the staff portal, the basic file and folder structure was setup similar to the student module. The authentication check from student was copied to the staff controller file and was updated to check if staff are authenticated. The blueprint was created with the /staff url prefix and imported into the app.

### Staff account management

The first basic staff route to be created would be the account route, identical to the account route found in student. To make this route more global, it is moved to the auth module.

To make this route more powerdul for staff, a check is added for if the current user is staff. If the current user is staff, additional fields are shown in the form. If the staff member is editing themselves, only username is shown is extra. If another account is being edited then a field for auth level is also shown. Additonally, if the user is staff and editing another account, the option to disable the account will be shown.

To complete this the form validators have to be removed and all checks will be done in the backend. This is as if validators were in place and then some parts of the form were not dispalyed, the form would never validate and submit.

#### Issue with auth level input

The auth level input selector has a value of either 1 or 2. However, due to how html inputs work, flask was seeing these as strings when selected leading to the form not validating and throwing an error.

The fix for this was in the WTForms SelectField definition for the auth level input to set the coerce attribute to int which will cause the field on validation to attempt to convert the value it gets to an integer before validating it which allows the input to work as expected.



### Account list

With the account management setup for staff to be able to edit any account, the next step is to create a basic view in the staff portal with a list of all accounts.

This will use the same render table method that was created in the student portal but with different heading and data. The table will display the username of the account, the auth level with its label and integer value. Additionally, whether the account is disabled or not will be shown and an edit button will be shown for the account.

### Create new account

The first step of creating a new user from the staff panel, the form has to be defined in forms.py. As all the components of the form will always be displayed, unlike with the edit account form, we can use validators within the form definition this time.



However, standard backend checks will still be made as the validators cannot provide all the checking functionality we need, such as ensuring the username is unique to this account. Whilst the username has no constraint in the database to be unique, having two accounts with the same username would be confusing and so a manual constraint in editing and creating accounts is added to ensure the usernames are unique. This is done by simply selecting all results from the User model with the username, if there are no reults then the username is not in use and can be used now.

With the backend complete the frontend for the form could be created. Similar to the backend, the frontend could be copied from the edit account template and modified slightly to support the updated form attributes.

#### Testing the inputs

Once the design was completed it could be tested to ensure it was working correctly.

Having a blank username field and pressing create relies on the HTML5 validation and renders a popup telling the user to input a username. This is the expected result.

The same occurs if no password is present.

This is all the frontend validation that is present. On the backend further validation is done, such as ensuring the username is unqiue. If a taken username is entered and create is pressed, an error is displayed in a formatted form a expected.

No further validaton is present or needed on the form. Validation testing to the same effect was carried out on the edit form at this stage with identical results as the front and backend code is very similar for both.

### Deleting an account

To ensure data consistency there is no way in the system to delete a user account. Instead the disabled attribute in the user model will stop a user being able to log in to their account, making the account essentially deleted whilst still being linked for relevant other data entries.



This may be confusing for a new user to the system and so on the main staff account management page a note was added to the frontend design explaining this, below the newly added create new account button in the sidebar next to the main account table.

This provides more content for the page to balance it out as with just the create new account button the sidebar was very unbalanced with the table.

### Rota control

With the account control done for the staff portal, the other major component is the rota control. To begin this, a rota view similar to what the students see on the full view will be generated with an additional edit button provided to the staff. The edit button will take the staff member directly to an edit page for that rota session where they can edit which students are assigned.

With this plan to edit button would not allow the staff member to edit the session, only who is assigned to it. This would mean a secondary page would need to be created for editing the session itself. To make the system less confusing, it was decided that there would be two edit buttons provided on the rota view, one for updating the session and one for controlling the assignments on the session.

Missing from this layout was the ability to create a new session easily. To remedy this, a button on the top right of the page was added which allows staff to create a new session on the rota. Additionally, to make the rota easier to understand, it will automatically highlight sessions with no students currently assigned to it.

### Edit session

A key part of the rota system is the ability for staff to easily manipulate the sessions in the rota layout. For this a new form has to be created which will allow the editing of existing sessions and the creation of new ones.

The form has three inputs, a standard select for the day of the week, using list comprehension in the form class to quickly generate all the possible values.

Along with the day value, start and end times are also inputs required from the user. These are done with the HTML5 time input field, which falls back to a standard text field on browsers that do not support it. WTForms handles parsing the input from this into a standard datetime.time instance.

### Create session

Creating a session is very similar to editing one however instead of updating an entry in the database, a new one has to be created. Therefor the method for editing a session can be copied and edited to create a new entry in the database.

#### Testing

With the method written, testing can be carried out on the edit and create session pages.

Failing to input a value in the start time or end time results in a popup showing the field is required before submission. This is a frontend check based on the required attribute in the input fields. In addition to this there is also a backend check that also ensures a value is present.

Inputting a value into start time or end time that is in an incorrect format, an error is shown at the top of the page (e.g. 1pm). The correct format is displayed to the user in the label of each field on the form (e.g. 13:00).



Additionally, if a start time is inputted that is after the inputted end time, the backend checks will detect this and return an error to the staff member creating or editing the session.

### Updating assignments

The ability for staff to update assignments on sessions in the rota is a key feature of the system. Looking at all the HTML5 input options available to me, nothing struck me as an easy to use input option for selecting multiple people from one list to have them in another list. Based on this, I decided it would be best to create my own custom input that relies on JavaScript on the client side of the site to send the correct data back to the web server.

For the input, a single hidden field was used in WTForms which will transmit that data to the server. This had no checks implemented in it directly as this input would not be controlled by the user. On the server side, at the time the page is loaded, the template renderer will be sent a list of students currently assigned to the session and all those who are not assigned. This will be the basis for creating the custom input in the web page.

With the data passed to the template, creating the custom input was the next thing to do. This involved creating two lists displayed to the staff member which had buttons on each item to swap it to the other list.

#### Button design

The first stage was to mock up a basic button design that would offer the ability for left and right buttons with the student text in the middle. In operation, only one of these buttons would ever be shown at the same time but for designing the button, both will be displayed.



#### Button generation

With the button design created, next up was to create a bit of JavaScript to create this element on the fly when it was needed. This involved creating each element that is in the button design and the joining them correctly using appendChild or insertBefore to get the a element in the correct positon relative to the span that gets appended first.



The script created worked as intended and returned the correct HTML, matching the design but with only one button as will be needed for the custom input on the page.

With this done, the on load JavaScript can be setup that will generate all the initial buttons based on the data passed by the template.

This requires a slight modification to the script created which appends the elements directly to the document.

With the intial genration script written, the page rendered at load with the correct student buttons in place ready for the staff member to manipulate as they see fit.

#### Button click events

To allow the staff to control the items, on click events need to be added to these custom buttons. To begin this process a general click handler is written that will be called for each on click event. The click handler pulls data from the element clicked using getAttribute and then uses this to generate a new button in the opposite list.

With the click handler created in its basic form, it can then be attached to the elements when they are created. This involves once again editing the ‘create\_button’ script to add the onclick attribute to the ‘a’ element created. The onclick attribute is given a basic anonymous function that prevents any default actions for the click event and then calls our custom click handler with the instance of the ‘a’ element passed through.



#### JSON input updating

With the UI completed for the inputs on this custom page, the key part now was to get the hidden input updating when the UI is updated by the staff member so that the update button will post the correct data back to the server.

The hidden field being used in this form will use JSON to hold the data that needs to be posted back to the server. JSON allows objects such as arrays in JavaScript to be displayed in a text form that can be read back into a language as a variable.

To update the JSON value stored in the hidden input, the click event for the buttons needs to be modified to edit the JSON data. This involves reading in the current value from the input, adding or removing an item, then exporting it back to JSON and setting the input to this value.

#### Updating the database

With the JSON value posted back to the web server, the data now needs to be placed into the database or removed from it. The first stage is to parse the JSON data from the form and to create lists of student ids that need to be removed from the database and additionally a list of ids to add.

To remove assignments, a new attribute needs to be added to the assignment model, a removed value that will act as a flag and a timestamp for when the assignment was removed. This allows it to be ‘removed’ whilst still tracking for attendance.

This change to the model required large changes throughout the web server to ensure only active assignments were used where needed. Once this was completed and the remove/add database changes were made the update assignments section was completed and ready to be used.

#### Showing unavailabilities

A key part of the system is that students can set themselves as unavailable for some sessions on the rota. This should show up for staff when they are assigning students so that hopefully the unavailable students won’t be assigned to that session.

To allow this to work, the method for passing assigned and unassigned data through to the page template will be updated to include a new item which is the student unavailability.



With the backend updated to support this, the template now needs to be updated to process the new item passed through and render it for the user viewing the page.



The relied on a simple for loop in the template to render each unavailability passed through, with a fall back if there are no student unavailable.

As part of this, the reason displayed in the template was put through Flask’s internal escape funciton which escpaes all HTML entities to stop XSS occuring, which is a real danger in wwebsites that can lead to them being comprimised if not correctly dealt with.

With the unavailabilities displayed in their own column, this information is proivded to the staff member whilst they are updating assignments. However, the studnets that have marked themselves as unavailable are still shown in the selection and a staff member could easily still assign them whilst missing the unavailablity. To counter this, it was decided that unavailibe students would also be marked in the selection part of the page.

To achieve this change, additional data will be sent within the assigned and unassigned lists sent to the template. This additional data will be a simple boolean marking if the user is unavailable or not for that session.

Using this boolean an additional class can be applied to the buttons for those students who are marked as unavailable, muting the buttons to make it more obvious they shouldn’t really be used. They are however not disabled in the event a staff member decides that an unavailable student needs to be assigned.



### Automating Style Generation

Whilst working on the above development, some additional margin overwrites styles were created. These required a lot of manual work to generate all of them in the style files.

To reduce the workload, a custom function was written in SASS, the CSS pre-processor being used for this project. Using a SASS array and then an each loop built into SASS, all the required margin overwrites styles could quickly be generated.

These functions written in the SASS files are then compiled during the development cycle by the SASS pre-processor into the static CSS file used by the site and the browser to style the HTML on the pages.

### Generating Rota Assignments Automatically

The ability for staff on the system to press a button and have the rota assignments automatically generated is important to the end product and ease of use with the system.

With a large rota, this functionality will drastically reduce the level of work the staff member will need to put in to get the rota ready to use by students.

The problem of algorithmically creating the assignments originally looked like a simple resource allocation problem that could be solved by using an algorithm designed to handle these situations. However, after looking into these algorithms such as the Hungarian algorithm, they mostly require some cost factor to assign the most efficient solution.

With the rota system, there is no cost factor that can be used for the assignment generation, just a list of students and a list of sessions on the rota. Therefore, it was decided that a bespoke algorithmic solution would be implemented to best generate the assignments.

#### Fetching the next user to assign

The first step to tackling this allocation problem is a relatively simple function to fetch the next user to be assigned to the session.

This is achieved by providing the function a list of all users and the session. From this, the function can then iterate over the list of users and find the first that is available for the session. In the event that no users are found, the function accepts a force flag that will ignore unavailabilities and select the next user in the list for this session.

If a user is found, it can then be moved to the back of the users list which is then returned to be used again if needed. Alongside the returned list of users, the specific user selected, if any, is also returned.