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

## Jackson

* better write up of Hungarian algo, with step by step example like wiki

SLST Precinct Management System

Matt Cowley 4807 – Royal Grammar School High Wycombe 52423

# Analysis

## Objectives

* Provide an easy to use rota system for students on duty in the precinct
* Provide an easy management system of the rota system for staff
* 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 for the precinct
* Allow staff to define the precinct duty sessions for the week
* Allow staff to view when students signed in and out
* Allow staff to view in-depth analytics regarding student attendance for their duties
* Display to staff the attendance levels for individual students on the precinct, the student’s reliability
* Display to staff the duration each student stays for their precinct duty, percentage of defined precinct duration
* Provide an authentication system for staff and students
* Allow staff to easily manage student accounts and other staff accounts

## 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 within the school 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 a rota, a web based solution, implemented the precinct and its 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.

By using a computer system to manage the precinct rota, the management of the rota can be far more advanced that using a simple paper system. Part of the plan for the rota system is to allow for staff to automatically generate a new rota of assignments based on the students and rota sessions the staff member inputs into the system. This is very open to computational means as algorithms for resource allocation, such as the Hungarian algorithm, can be used to automate the process of creating the rota.

Additionally, with the rota being managed on a computer system, far more data can be easily logged regarding the attendance and performance of students on duty, which can then be processed by the web system to provide powerful analytical data and reports to the staff members. This is far easier to achieve using computational means than with pen and paper as all the calculations can be automated by the system along with the majority of the data logging needed to provide the data for the analytics.

## 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 (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]**
    - Int: User id ***[foreign]***
    - Datetime: Created
  + 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
       1. Find user to assign function
       2. Assign users to sessions function
       3. Front end control for function
       4. Test page
       5. Discuss Hungarian algorithm
    6. Ability to delete sessions (must remove assignments)
20. Test disabled accounts
    1. Fix routes that fail to check for disabled accounts
21. **Attendance report**
    1. Student overview report
       1. Bar graph for attendance per assignment
       2. Scatter graph for in diff and out diff per assignment
    2. Per assignment report for each student
    3. All students overview (home)
22. Staff home page
23. UI improvements
    1. Sort (and provide filters for) staff account list by not disabled, staff only, students only
    2. Make everything easier to understand, general ui tidy

## Solution Description

(a) Explain and justify the structure of the solution. List all algorithms you will use. E.g. searching a 2d list, shortest path algorithm, collision detection for Snake. Use UML diagrams here.

The solution will make use of a web interface to provide all access to the system for the user. This will be handled through the very standard Flask framework for Python. The flask framework handles the boilerplate process of creating the webserver in Python and sending responses correctly. The web interface will have two main sections to it; a staff portal and a student portal.

Within the student portal there will be multiple pages and sections. These will be for the student to view their rota assignments, which will require lining the assignments and sessions database tables together to produce an output for the student. In addition, the student portal will also provide a way for students to sign themselves in and out for the current session, which again will require the linking of multiple database tables, this time the attendance, assignment and session tables. The portal will also give the student a way to mark an unavailability for a session in the rota.

Using a computer system to manage the precinct rota comes into play greatly with the attendance report section that both the students and staff will be able to access. For the student, this report will provide key information on their performance in their rota assignments and for the staff will provide an overview of all students as well as in-depth reports about each student’s attendance to their assignments in the rota. This report is part of the reason why this problem is amenable to a computational solution.

The staff portal will be more complicated than the student portal, providing more pages as part of the portal. It will provide an account control panel, allowing staff to create and delete accounts for both students and other staff members. This will require server side validation and processing of the form data into the database table. In addition to the account control, staff will also have full control over the rota layout through sessions. They will have the ability to create, update and remove sessions. This will require linking between assignments and sessions to ensure assignments are archived if a session is removed for example.

(b) Describe the parts of the solution using algorithms justifying how these algorithms form a complete solution to the problem. Pseudocode/flowchart for all operations. If similar code in different parts of the solution - can link to the same pseudocode. If using OOP, use special OOP pseudocode for at least parts of your solution.

### Algorithm Application

The key part of the staff portal however is the assignment control. This approach to solving the initial rota problem is why this is best tackled through a computer system. The staff portal will provide the ability for the system to automatically assign students to sessions in the rota based on whether they are marked as available or not. This will be done through the use of a resource allocation algorithm, most likely one based around the Hungarian Algorithm. Using a resource allocation algorithm ensures that students are best automatically assigned to the sessions in the rota without issues. For the Hungarian Algorithm, a cost is expected for each resource/task combination, this will be handled as a simple cost of 1 or 0 dependant on whether the student is available for each session in the rota.

Due to the nature of how Flask works, the majority of the program will not be OOP but will instead by top-level functions within multiple modules in the application. This is how Flask, the web framework being used, is designed to be run when splitting web routes across multiple files. Flask does not support OOP for its route functions and instead expects a function/procedural programming style with a function for each web route. Flask makes use of blueprints which are separate files in the application where some routes are stored. These can be used to split the route out across multiple files, allowing in this project for the staff and student portals to be in separate locations.

*(c) Describe usability features to be included in the solution. Design all inputs and outputs (including GUI and paper output) of the program, justify your choice of colours, fonts, themes, widgets.*

### Design Overview

The site will make use of a rather standard design, with a header for navigation links at the top of the page, the main content in the middle of the page taking up the majority of the screen and then a footer for copyright and contact information.

A dark colour scheme will be used throughout as this is easier on the eyes than a bright white site and is also more thematic of a stage environment. Similarly, the accent colour will be a blue, which is a colour commonly used for backstage lighting. These colours will both work together to provide a clean, modern and dark colour palette for the site.

All text on the site will be white in colour to have high contrast against the dark background, making it easier to read. Many studies have been done showing that humans find it easier to pick out white text on a black background over black text on a white background, further supporting the argument for using a dark background on websites.



Two dark colours were picked to be used as the backgrounds for the majority of the site with a lighter grey also being picked out for accent elements that won’t be blue.

The blue picked had a pastel hint to it, following the current modern and flat design trend of the internet. The alert colours selected also conformed to the pastel style. These are used for any text or popup alerts that require attention.

As part of the design process for the site, a mock-up of the general base template for the system can be created. My choice of program for creating mock-ups is Photoshop by Adobe. This is a piece of software that I am very familiar with and find it easy and quick to create basic layouts and designs for mockups and other graphics.

In the mock-up, the font Ubuntu was used as it is clean and a font I enjoy using for designs and websites. This may change however to a more common and popular font from the Google Fonts online service such as Open Sans.



In the website, the font service Font Awesome will also be made use of as it provides an intuitive way to use icons in your website using the ‘i’ HTML tag. This icon system can be used, for example, to render the proper copyright symbol in the footer of the site where it is currently represented by ‘(c)’.

(d) Identify key variables / data structures / classes justifying choices and any necessary validation. Tables of variables, lists, GUI widgets SQL tables: names, data types, sample data, any constraints (validation rules)

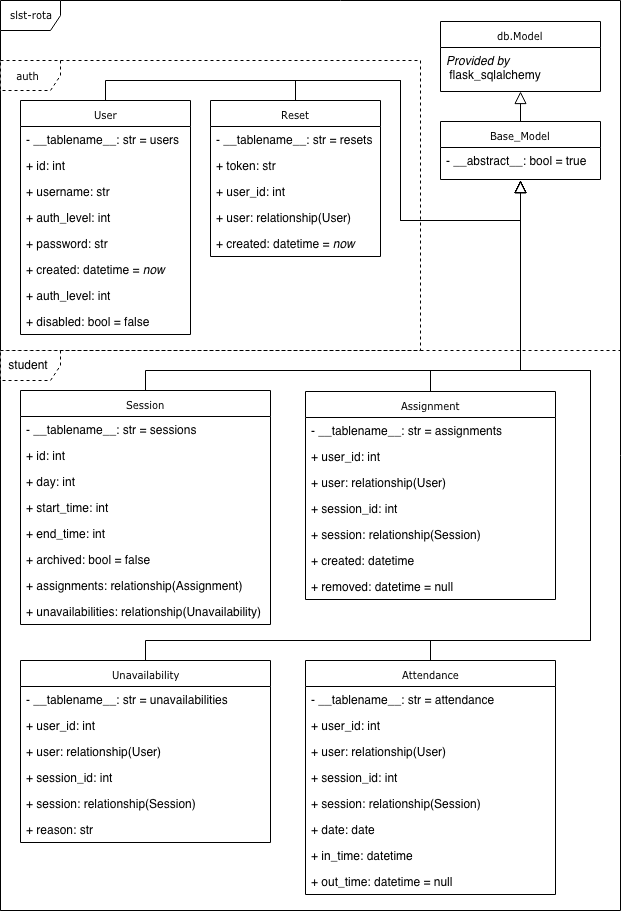
UML diagram of classes

### SQL – Database

However, whilst OOP won’t be used for the main Flask routes, it will be used in the database models that the program will use continually for all database access, reading and writing.

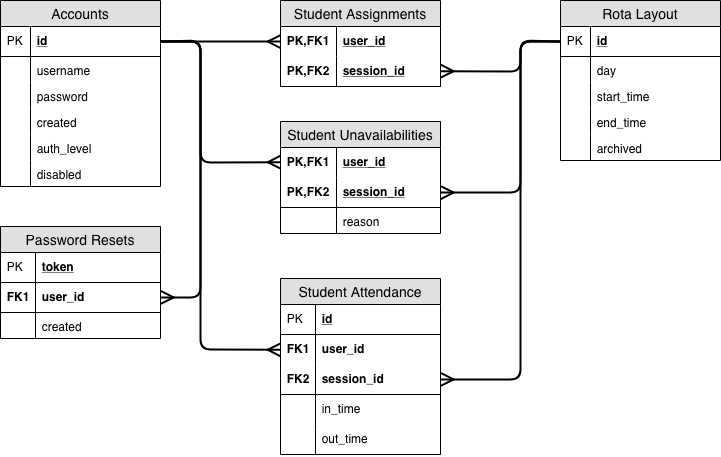
Based on the database layout specific in the objects above, a UML diagram of the OOP database model classes used in Python can be created. An ERD can also be created of the database itself to provide a proper insight into the structure and relationships.

#### Database Models UML Diagram



In the diagram, the planned database model classes are displayed. They all inherit from the base model class as part of the app which will inherit the database model class that the database engine I am using provides. Relationships between the classes are displayed by an attribute type of relationship as this was the easiest and cleanest way to display all the relationships. Frames have also been used to represent which module in the app the classes will be stored.

#### Database ERD



Creating the entity relationship diagram allows me to more clearly represent the relationships between each of the tables in the database and specifically which fields are responsible for the relationship.

Additionally, I can display how the keys for each table will be set out.

## Testing Plan

*Identify the test data to be used during the iterative development and post development phases and justify the choice of this test data. Test plan: what will be tested when.*

Throughout the development of the project testing will be carried out as is needed for each section written. During the development testing, data that is considered normal will be used as the first test to ensure that the code written is functional. Once it is determined that the code works, it can be tested with extreme and bad data, ensuring it works as expected in all situations and can produce a clean error for the user.

Much of the testing will be carried out on user inputs where the “content” is provided by the system and is simply manipulated by the user. Whilst this should not allow the user to modify the data itself, this will still be tested to ensure that if a user bypasses the front-end checks the system will still error correctly.

At each stage of the development where testing occurs it will be documented with the data used and the results. If an unexpected error occurs during any testing it will be rectified immediately if it breaks the system or may be put on a list for later fixes if it is not an urgent error.

The most important things to test are anywhere where the user inputs content. This will be on the login page, the password reset page, the create and modify accounts pages and the session create and modify pages. The assignment edit page will most likely be user manipulated by not have data directly inputted by the user, whilst this still needs testing it is not as likely to get bad data submitted like direct user inputs.

Any testing carried out should ensure both front-end and server-side checks are being carried out where needed. Any user input should use validation on the client-side to present a clean error to the user prior to data submission. However, validation should also be carried out on the server in the event that the front-end validation fails or is bypassed by a user.

Once the development is completed, further testing must be carried out to ensure the completed product is functional without any unexpected errors. This will be done by giving the program to a user who was not involved with the development of it and asking them to make use of the system. This will provide a new test of the software as this user will be unaware of the specific checks the system makes for each input and so may provide input data that will break the system or use the system in such a way that it fails.

The feedback from this user can then be used to finalise the design and functionality of the system before it is released in a beta state to be used by the SLST precinct monitors and staff on a trial period to find any long-term issues with the program. Again, the feedback from this beta testing period can then be used to create the finished product which can be released.

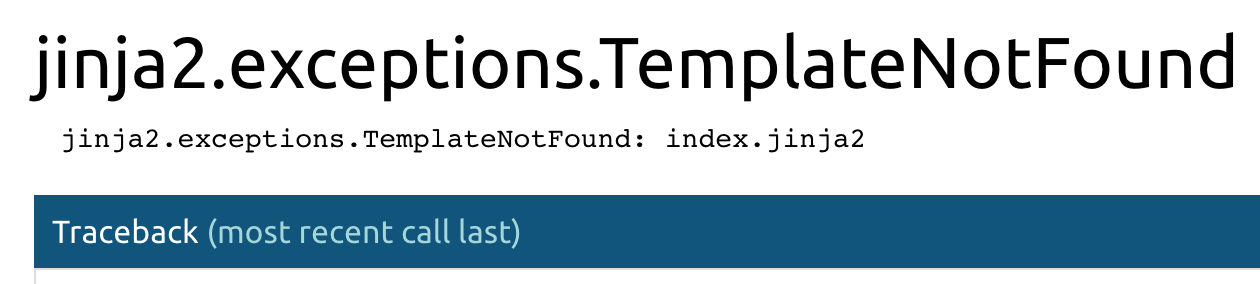
# Development

## 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 websites 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.

### Generating the assignments

With the method for fetching one user to be assigned next done, the next function to be written is one needed to generate all the assignments for the sessions currently in the rota.

For customisability, this function will allow a user defined number of users to assign per session and a pass through for the force user to be assigned option that is offered in the next assignable function.

The first thing the function will do is verify that there are sessions in the system that aren’t archived and that there are users of the student auth level in the system who aren’t disabled.

If either of these checks fail, the function will abort with nothing further done.

With both those checks done, the next thing to do is for the system to remove all old assignments in the system to leave a blank plate for the new generated rota assignments to be put on. This is done by selecting all active assignments and updating their removed value.

Once the old assigments are removed, the system can begin generating the new ones. To do this, the function will loop over all the sessions defined in the rota and assign students to each one.

This loop has multiple check fucitons built in to ensure it does not ever get stuck in an infinite loop. At each fetch of a new user for a session, the loop checks to see if this user was the last one assigned to the session. In the event it was, the loop will abort this repeat as it is clear there are no furhter users to assign for this session.

With the checks done, the function creates the user assignment for the session and commits it to the database.

### Testing the assignment generation

Once the function has been created, a basic test route can be created in the Flask staff controller which will allow me to check the function is operating as expected.

In the system, there are two student accounts available and one disabled account. One of the student accounts has an unavailability set for the first session in the rota.



To test the function fully, it will be called with three users per session and not to force assign. The result of this should be the one available student assigned to the first session and both available students assigned to the other two sessions.

The request for three per session is to ensure that the function won’t assign disabled accounts and won’t get stuck in a loop as it cannot find enough accounts to assign.

If the route and function runs correctly, the program should raise a generic Flask error as the basic test route is not returning a view. Any other error would indicate an issue with the function itself.

During the first run of the test route, an SQL error was returned. This indicated there was an issue within the function that handled the assignment generation.

The error indicated that an invalid value was being passed during the assignment creation for the student id. Looking at the function, it was evident that the full student object was being passed and not the id.



With that issue fixed, the test route was called again. This time the view not returned error occurered, indicating that the function had run without any error. Heading over to the full rota view, I can check that the assignments had been created as expected in the test.

The full rota view revealed that one student was assigned to the first session and both students assigned to the other two as was the expected result from the test.

Checking the DB ensures that no duplicate assignments were created that aren’t being displayed on the front-end view. In the database were the five assignments displayed in the front-end and the previous two assignments that are now remvoed.

### Creating the front-end

The next step for the automatic creation of assignments is a front-end page that the staff can access to create the new rota assignments. To begin this process, a form needs to be created through WTForms which will handle the user inputs for the number of students to assign per session in the rota and a checkbox input for forcing assignments or not.

The page will make use of front-end checks through HTML5. These checks will be a minimum and maximum requirement on the integer input for the students per assignment. Back-end checks will also be implemented to ensure the user hasn’t bypassed the checks presented by HTML5 and to double check any check that cannot be achieved on the front end.

In the form definition, the only check implemented will be a simple requirement for the student count input. The minimum and maximum values will be added during the route call as they will be dynamic.



With the form created it can be added into the route in Flask that will be used for the page. The first checks in the route will be to ensure that at least one session exists in the system and that at least one user exists. If either of these checks fail an error will be displayed to the user.

With the checks completed, the form can be loaded into the route and the min/max values set for the input based on the query to the User database.

Creating the page layout and design is the next part in the process of creating the staff page for controllin this function. The page will follow the same overall design as all the other pages and will feature the integer input, the checkbox input and go and a cancel button.

The page will be rather minimal as it acts as a simple intermediary page between the main rota and the automatic assignments function on the back-end.

When the form was created and rendered it was apparent that there was an issue with the IntegerField type in WTForms. It had been rendered as a text input.



Upon further investigation into the documentation for the IntegerField from WTForms I discovered that it was indeed documented as a text input that coerced all inputs to an integer when submitted. This was not ideal for what I needed here.

After some research, I discovered the solution to force WTForms to render it as a number input was to set the widget the field users to a number input. Implementing this rendered the number input field as intended. The setting of min and max in the route also had to be updated to refer to the widget directly. A default value was also added to the field definition for convenience of the staff member.



The back-end side of the form can now be written to handle the post request sent by the run button on the page. The post processing needs to include the same validation that the front-end provides as well as then calling the assignment function. The form also has built-in validators which can be checked and any errors sent to the user on the page via Flask’s flash method, displaying a message to the user.

The only back-end validation implemented by myself was to ensure that a count had been passed and that it was within the min/max range specified in the front-end. With that written, the call to ‘generate\_assignments’ is then wrapped a try except so that any possible error that occurs is caught and presented cleanly to the end user.

With this all written it is time to test that the full form works as expected, including the validation. First I attempted to submit the form with no count inputted. Upon pressing run I was presented with the standard HTML5 validation requirement error for Chrome.

Entering a negative value returns a similar Chrome error, instructing the user that the input must be one or more. Similarly, entering a value that is above the maximum set will also produce a Chrome error.

The extreme data tested returned the expected errors to the client, handled correctly. Normal data can now be tested to see if the assignments are still generated as expected.

When a normal piece of data was entered, and the run button pressed, the function executed successfully and the route redirects the user to the rota view with no success message displayed. This may be a rather misleading result. As such, I decided to instead flash a success message back to the automatic generation page and to then provide a back button to the rota.

Heading back to the full rota view revealed that the assignments had again been generated as expected. However, there is currently no button on the rota view allowing staff to access the automatic assignments page. This was resolved by adding a button at the top with the create session button already present.

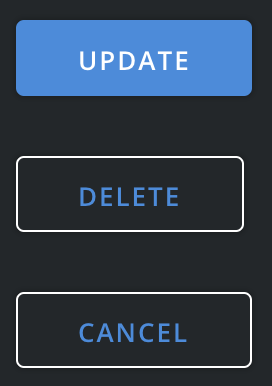
### The Hungarian Algorithm

The algorithm used for the generation of the assignments was based heavily on the Hungarian Algorithm, a resource allocation algorithm based on item cost. The algorithm will produce the cheapest combination of resources for the required tasks, perfect for the task of generating assignments for a set number of sessions with a known set of students as the resources available. This is what made this system perfect for being handled through computational means.

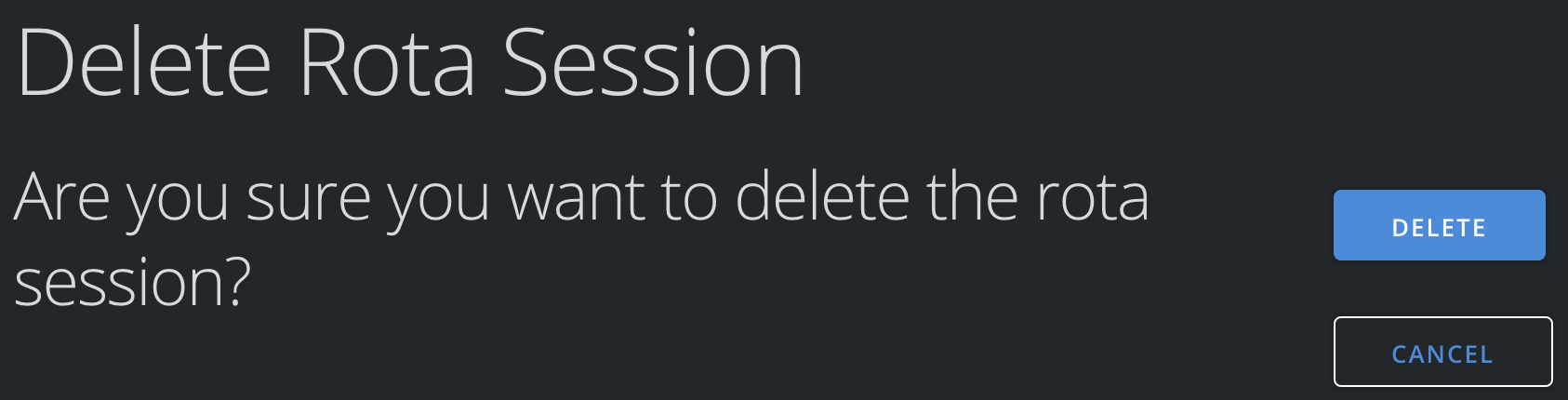
In the context of the rota assignment generation, the cost for each student in a session was either 1 or 0. The cost would be 1 if the student was marked as unavailable for the session otherwise it would be set to 0.

Using the Hungarian algorithm as the basis allowed me to ensure that the function for generating the assignments was as efficient as it could be whilst meeting the requirements set out in the design of this system, ensuring students were allocated fairly and only if they were available if stipulated by the staff member.

## Deleting sessions

Whilst working on the automatic assignment generation, I observed a flaw in the session management system. Currently there are methods provided for creating new sessions and editing existing sessions but no way to actually delete a session.

To implement this, the update page for a session will be modified to include a delete button which will take the staff member to a secondary page to confirm the deletion.

With the button created, the confirmation page for deleting the session needs to be created. The page will make use of a very basic form that has no inputs and will simply post to the route when the user presses the button in the confirmation page.

Upon the form being submitted the system need to remove all assignments for this session first and then remove the session from the system. This is similar to how the atuomatic assignment generation works with removing assignments so the code from that can be re-used and modified to only affect the single session.

To remove an assignment, the system simply has to sets its removed attrivute to the current datetime. To remove a session its archived attribute must be set to true or 1.

### Testing

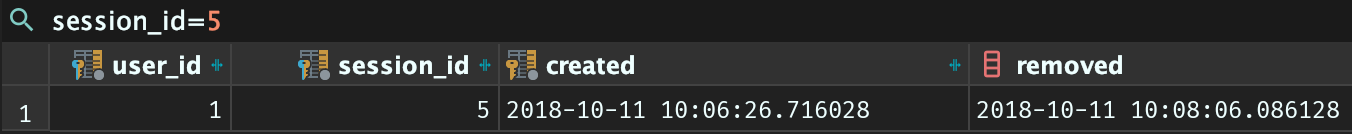
With this done the session should be removed from the system. It must now be tested to ensure it works as expected for the user. To do this, a test session will be created on a Friday where no other session currently exists.

The test\_student account is then assigned to the session so that the assignment removal can also be tested. Once the student is assigned, the session deletion test can be done. This is done by navigating to the edit page for the session, then pressing the delete button.

Pressing the delete button should take the staff member to a confirmation page for deleting the session. As expected, this is what occurs. A secondary delete button is then displayed which must also be pressed to confirm the deletion of the session.

Upon pressing the second button the page redirects the staff member back to the rota page and upon checking the rota, the session has been removed successfully without any erros occuring.

The database can then be checked to ensure that the assignent was also removed correctly and is not still active whilst being hidden because the session is removed.

In the database, searching for session\_id = 5 will find the assignment for the session we just deleted.

The result in the database was one row as expected and the removed attribute has been set to the correct timestamp for when it and the session was deleted. The test was a success and the session deletion part of the rota system is completed and working correctly.

## Testing disabled accounts

In the account management part of the system, staff members can “delete” accounts by disabling them. These accounts should then not be able to log in and should not be displayed at all throughout the system except the account management page. This needs to be tested as I believe there are parts of the system where I have currently forgotten to implement disabled checks.

To complete the testing, the student account “disabled\_student” can be used.

### Staff System

Every page on the staff system will be checked first and any page where the student account shows up will be logged below with the resolution listed. Any observation that needs to be fixed will be highlighted in red as an error in the current version of the system.

|  |  |  |
| --- | --- | --- |
| Page | Observation | Resolution |
| All Accounts | Disabled accounts shown | Disabled accounts should be displayed |
| Edit Account (Disabled account) | Disabled dropdown set to No | Set current values for all inputs in the form at page render |
| Update session assignments | Disabled students shown as available to assign | Add correct filters to assigned/unassigned database fetch to remove disabled students |
| Update session assignment (Disabled student unavailability) | Unavailability for disabled student is shown | Filter the unavailabilities to only show active student accounts |
| Rota View (Disabled student assigned to session) | Disabled students listed as assigned | Filter out disabled students when rendering rota |

### Student System

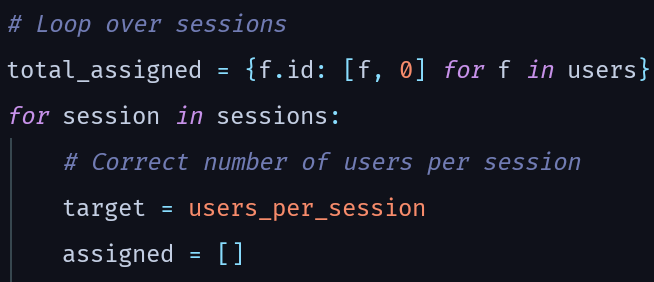
The student system was also checked with a disabled account. The login page has the correct checks in place and so will not let a disabled user ever log into the system.

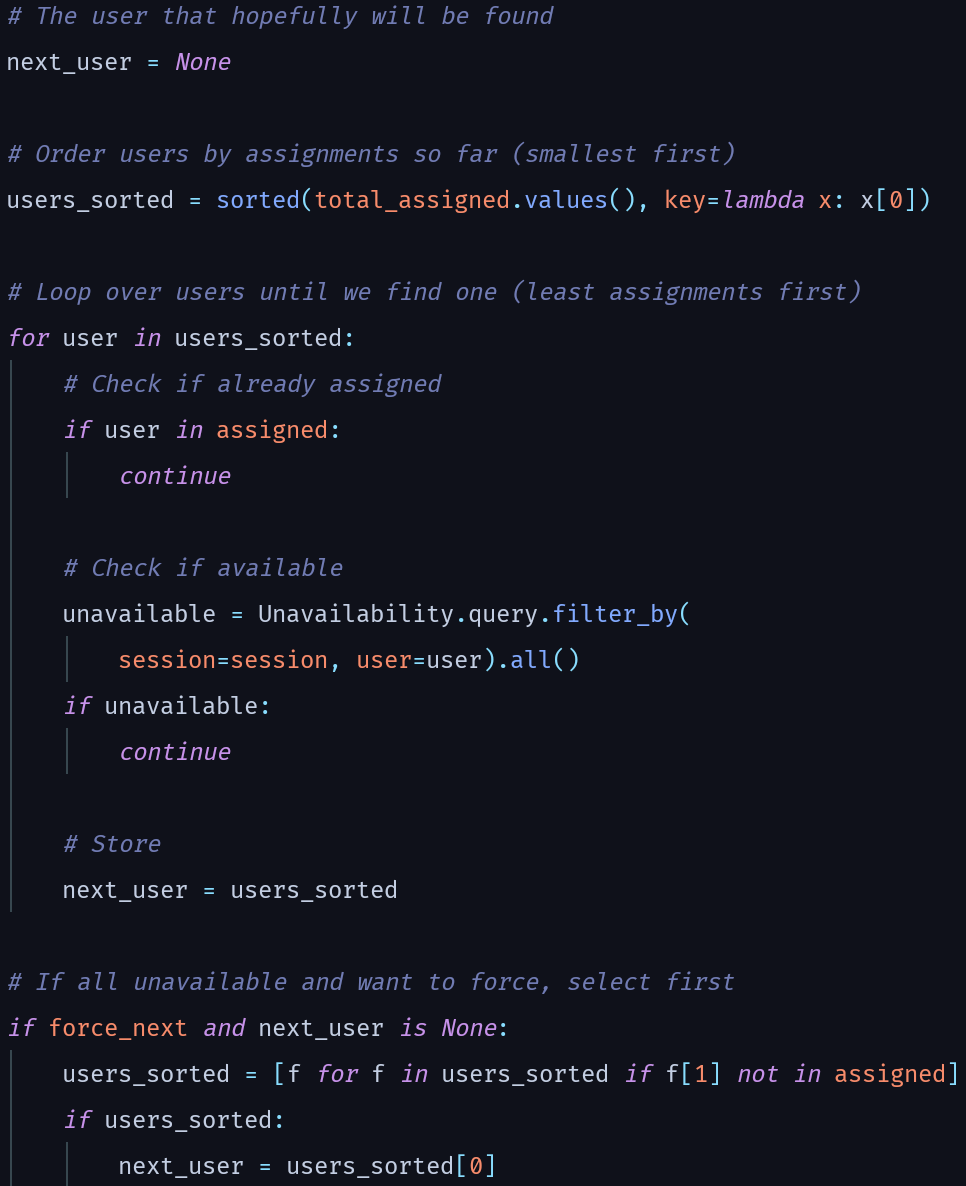
However, much like the staff rota view, if a normal student logs in and a disabled student is assigned to a session in the rota, this disabled user will still be displayed in the full rota view. This issue was rectified in the same manner as the staff rota view, filtering out disabled accounts before the rota is rendered.

## Automatic Student Assignment Issue

Whilst working on the program, I noticed a possible issue with the current implementation of the resource allocation algorithm that is used for automatically assigning the students to the sessions in the rota.

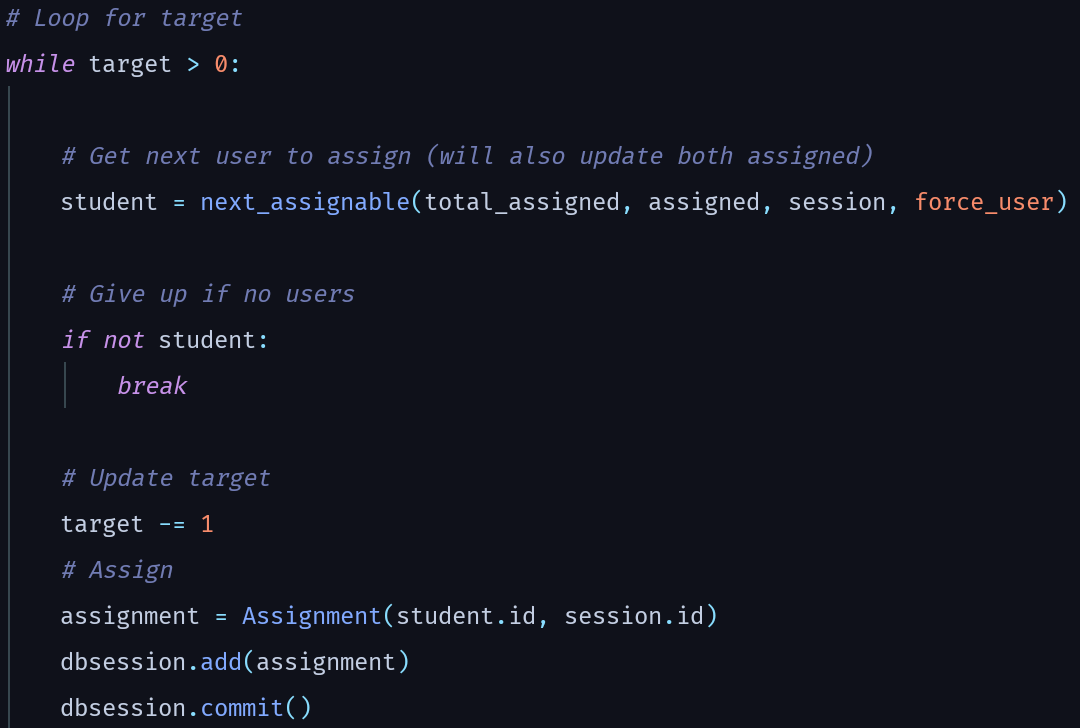
Currently the system just iterates over a list of all students and moves the assigned student to the bottom. A better way for the system to approach this would be to store how many assignments each student has and then always pick the student with the lowest number of assignments. This should result in a fairer assignment system than just looping over the users as some may get picked more than others if some are unavailable.

The first change is that two variables will now be used in the assignment generation to correctly track the users. There will be a main ‘total\_assigned’ variable which will be fixed for the entire generation. This will track the total number of assignments for each user. For each session, there will also be an ‘assigned’ variable created which will store a simple list of all users that get assigned to this specific session.



The function to determine the next user for assignment can then make use of both of these to pick the best possible user to be assigned next. This will work by first creating a sorted list of users by the total number of assignments each user has so far. The list will be sorted so that the user with the least number of assignments will be first in the list. Iterating over the list and performing the relevant checks on each user will then allow the system to pick the best user to assign, making the system far fairer for automatic assignments than before.

For each user, the system iterates over whilst trying to pick one, it will check if they have already been assigned to the session and if they are marked as unavailable. If either of these checks fail, the system will skip to the next user. If the system iterates over every user passed and does not find any viable users, it will check if the force flag is set. If it is, the system will automatically pick the user with the lowest number of current assignments that isn’t already assigned to this session.



This user (or none) can then be passed back into the main session loop where it can be processed and assigned if valid. There is a simple check to test if a user was returned or none. If none was returned, then the loop will abort as there are no users left, else it will assign the user returned by the ‘next\_assignable’ function.

### Testing the new method

To test the new automatic assignment system reliably I wiped the assignments and attendance database tables to start from a blank slate so that any issues would be very obvious.

In the user table for students there are still four accounts, three active and one disabled account. In the sessions table, there are three active sessions; Monday, Tuesday and Wednesday. For unavailabilities there is only one, the second student account is set to be unavailable for the Monday session.

For the test, the staff account will request automatic assignment of two users per session, with the force option disabled. The expected result here is that on the first session, student **one** and **three** will be assigned as two is unavailable. For the second session, it would then be expected that student **two** would be assigned as they have the least assignments so far as well as either account one or three, dependant on how Python decides to internally sort (I believe it will do database order and so the other account assigned should be account **one**).

Based on that assumption, the first account now has two assignments, the second student has one assignment as does the third. Therefore, for the third and final session the system should automatically assign students **two** and **three**.

#### Expected results

|  |  |
| --- | --- |
| Session | Students |
| 1 | One, Three |
| 2 | Two, One |
| 3 | Two, Three |

The total assignments are the important factor here, as well as the fact that account two is not assigned for the first session. The students assigned in two and three should not matter as long as the total assignments is as follows:

|  |  |
| --- | --- |
| Student | Total Assignments |
| One | 2 |
| Two | 2 |
| Three | 2 |

#### First issue

When the run button was pressed, the system came back with "An error occurred whilst generating the automatic assignments. Please try again.”.

It appears that somewhere in the updated algorithm there is an issue that has gone undetected.

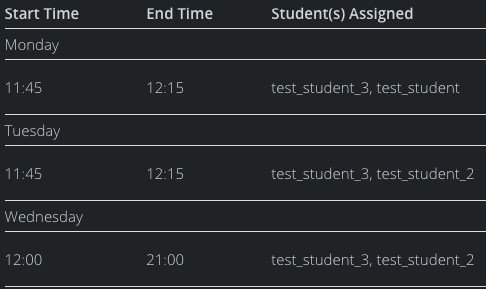
At this stage, I added a line to print the full error to console in the try catch block around the assignment generation so I could better understand what was broken.

The error that came back was an internal one from the database library that had a traceback pointing to a key-word argument of a user being passed into the query in my code. Upon digging into this, instead of passing just the user into the query I was passing in the list that was beig used for tracking the current assignments as well as the user.



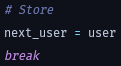
The fix was at the start of the internal loop to select just the user and pass that into the query.

With this error fixed, pressing the run button produced a message of “Assignments have been successfully generated. Press the Back button to view the rota.”. Now I can check that the system behaved as predicted prior to the test.



#### Second issue

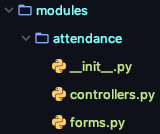
Upon pressing back however, the result was not what I expeceted at all. Student three had been assigned three times with the first student only being assigned once ever, something else in the algorithm was also broken.

After some digging and debugging, the issue appeared to be that the algorithm simply was not selecting the correct user, even if all the checks passed. Then I noticed the issue, a missing break at the end of the for loop to select the user. This meant that every time the loop was continuing on even if it found a correct user and so it always ended up picking the available user with the most prior assignments instead of the least. Luckily the fix for this was simple, add a break at the end of the loop when it selects the user.

With this change made, the assignments can be wiped and the test can be run again. As before, the success message was returned on the form when the Run button was pressed. Returning to the list gives a good looking rota with correctly generated assignments that match up with the expected results.

Whilst the assignments for session two and three do not match the expected results, the total assignments are correct and the important assignments, session one, are correct. With this in mind I am happy to move on as this algorithm is now functioning correctly and far more farily than before.

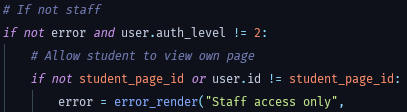
## Staff Attendance Report

Key to the end product is that staff can view a well-designed and easy to understand overview of the attendance for all students on the precinct. This attendance report will provide an overview of all attendance for the precinct as well as overviews for each student and in-depth data on each student’s attendance for assignments.

### Student Reports

To make the system easier to develop, I decided to make the attendance section into its own route and controller to split it up from the already rather large staff controller. To begin the process, I decided to move the user attendance view into this controller and redesign to it work to produce any student attendance review with the student id passed in the url.

To enable staff to access any student page but for the student to be able to view their own page required an edit to the staff page authentication check, accepting a new optional argument which is the student id for the current page. Using this, the check can then allow the student to view their own page as well as staff to view any page.



### Attendance Home

With the student report page moved over from the student controller to the attendance controller, the next item for the controller was the home page for staff to access to view a quick overview of attendance for all students in the system.

This home page will provide an attendance bar similar to the student page, but for each student in the system within a table view. It will also provide the overall punctuality stat from the student overview page on the table. To make this easier, instead of duplicating code from the student overview, I decided to move the student attendance report data to its own class that could be accessed from anywhere in the controller.

The new report class contained almost identical code to the old student overview report page however it was updated to work using class attributes instead of local variables and the assignment information generation was also moved out to its own class called AssignmentReport. This allowed the code to be made far more readable as it was split up and tidied.

#### Design

To begin the process of creating the home view for the attendance controller, a design had to be created of what the page would look like. This was done in the browser using inspect element on an already existing page as this was much easier than using Photoshop as the designed elements from the site css were available to use.

The page will consist of one large table that will house a row for each student user known to the system. Beside each student there will be a punctuality bar similar to that found on the individual student attendance report as well as the overall punctuality stats for the students. To aid the staff member with quality identifying problematic students, any student with an attendance below 75% will have their row highlighted for attention by the staff member. Each row will also have a button linking to the full student attendance report.



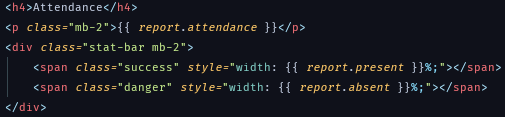
#### Development

The first step to aid in the development of this view was to move the attendance bar that was used in the student report from the Jinja2 (html) template into the StudentReport class so that it could be easily re-used without code duplication.



Whilst doing this, a bug was also fixed where if the student had no attendance records (present or absent) then both the present and absent percentages would be zero leading to a broken bar. This was fixed with an if statement that would set the absent bar to 100% if both present and absent were at 0.

This change meant that the student report template also became cleaner and shorter in code length, making it easier to follow when updating it as the bar was now a simple attribute of the report passed into it.



**Before**



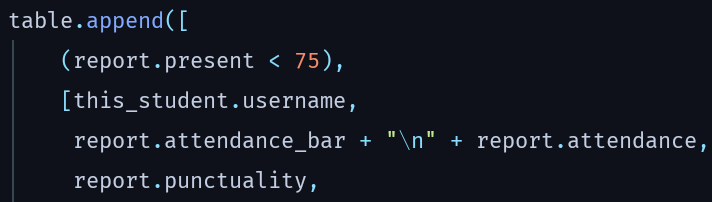
**After**

Having the attendance bar within the student report class means that I can easily and repeatedly access it throughout the code, meaning I can make use of it in the attendance homepage without code duplication. With this done I can now begin work on the table generation for the homepage based on the above design.

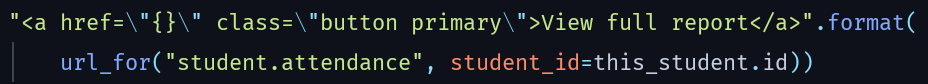
To generate the table I will look over each active student in the database and generate a full attendance report for them. Attributes from this report can then be inserted into the table. The first item in the table entry is the flag for highlighting the row, this is being done if the student attendance is below 75%. This can therefore be done with a simple if statement that will produce a Boolean.

The second item in the table list is then the list of elements to be displayed in the table row on the page. For this page design, the first item will be the student username followed by their attedance then the punctuality with a button to view their full report at the end.

The username was the easiest item, just referring back to the student object I already have in the loop. The attedance item required two parts of the student report to be combined, the attedance bar and the worded stats. These were joined together with a line break to make the site source more readable should anyone ever need to look through it. Punctuality is a single item which is the worded stats from the student attedance report and so this can be diretly referenced.



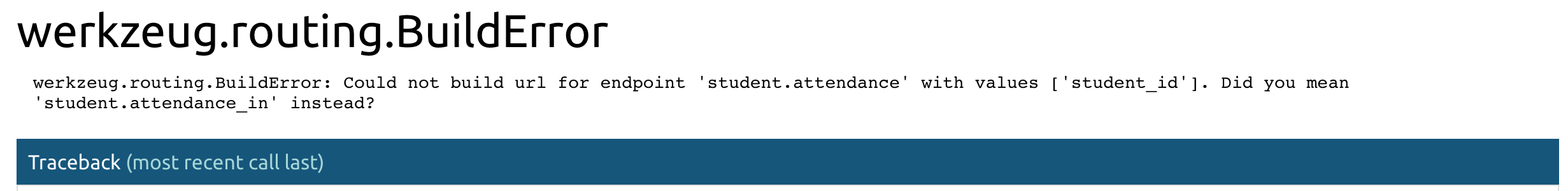
The last item is the button to access the full student report page. This has to have the html written for it in the controller to be inserted into the table. To generate the correct link, the built-in Flask function ‘url\_for’ can be used. This takes a string argument to lookup the correct route and then any additional arguments that the requested route takes. In this case the route is the student attendance page and the only arugment it requires is the student id.



#### Testing

With this completed the table generation should be ready to test as this is the only item that will be rendered on this overview page for now. To test this route, existing data in the database will be used with three student accounts active all with assignments. Upon loading the page, it should show all three accounts with their attendance and punctuality stats. None of the accounts have been signed in and so they should all be highlighted as their attendance will be below 75%, at 0%.

However, when the web server was started and I attempted to access the new page I was greeted by a Flask error. The error was a url\_for build error as I had given it an invalid route to find.



Upon reviewing the line that was the source of the error, the issue was apparent. In the url\_for call I had specified the route “student.attendance” when in should actually be “attendance.student” as I had recently moved the student attendance report to the new attendance controller so it was no longer in the student controller.

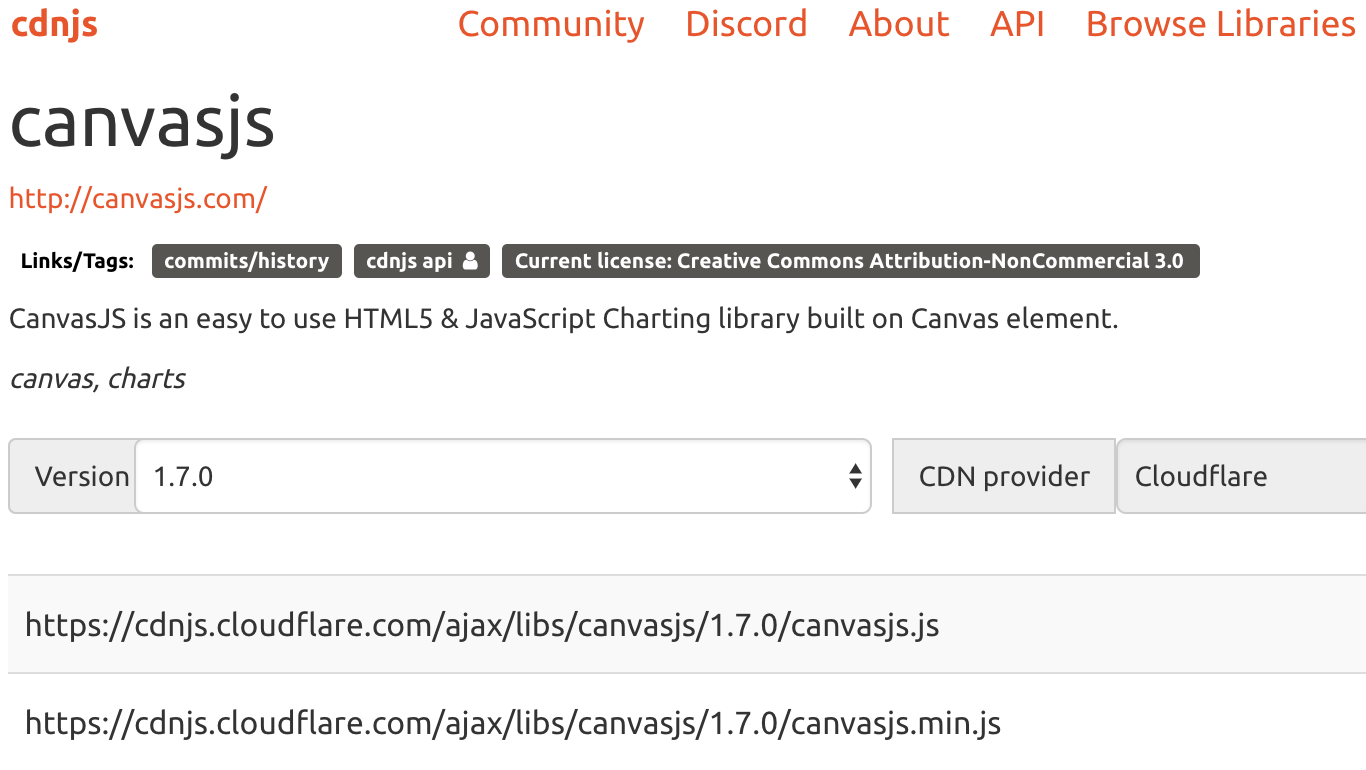
With this error fixed, the route worked correctly once the web server was rebooted, rendering the content as expected.



### Graphs

To make the page more appealing and interesting to the staff member who views it, I decided I would also produce some graphs for the page. The first graph that I will implement is a graph to display attendance per session, instead of per student. This will hopefully highlight to staff specific sessions which may have issues that are not due to a specific student’s issues.

To produce this I will make use of a well-known JavaScript library for rendering graphs on websites, CanvasJS. This library makes use of the canvas HTML element to generate custom and well-designed graphs for an end user to interact with. To embed the scripts and stylesheets required for CanvasJS I will make use of a CDN (central distribution network), cdnjs.com which is powered by the massive online CloudFlare network.



A CDN service is a massive network of servers distributed around the world that all have the same content on them. When the user visits your website, it will request the resources from the CDN, using the closest available CDN server which should be much faster than loading it from the web server directly as most likely the end user is closer to a CDN server than your web server.

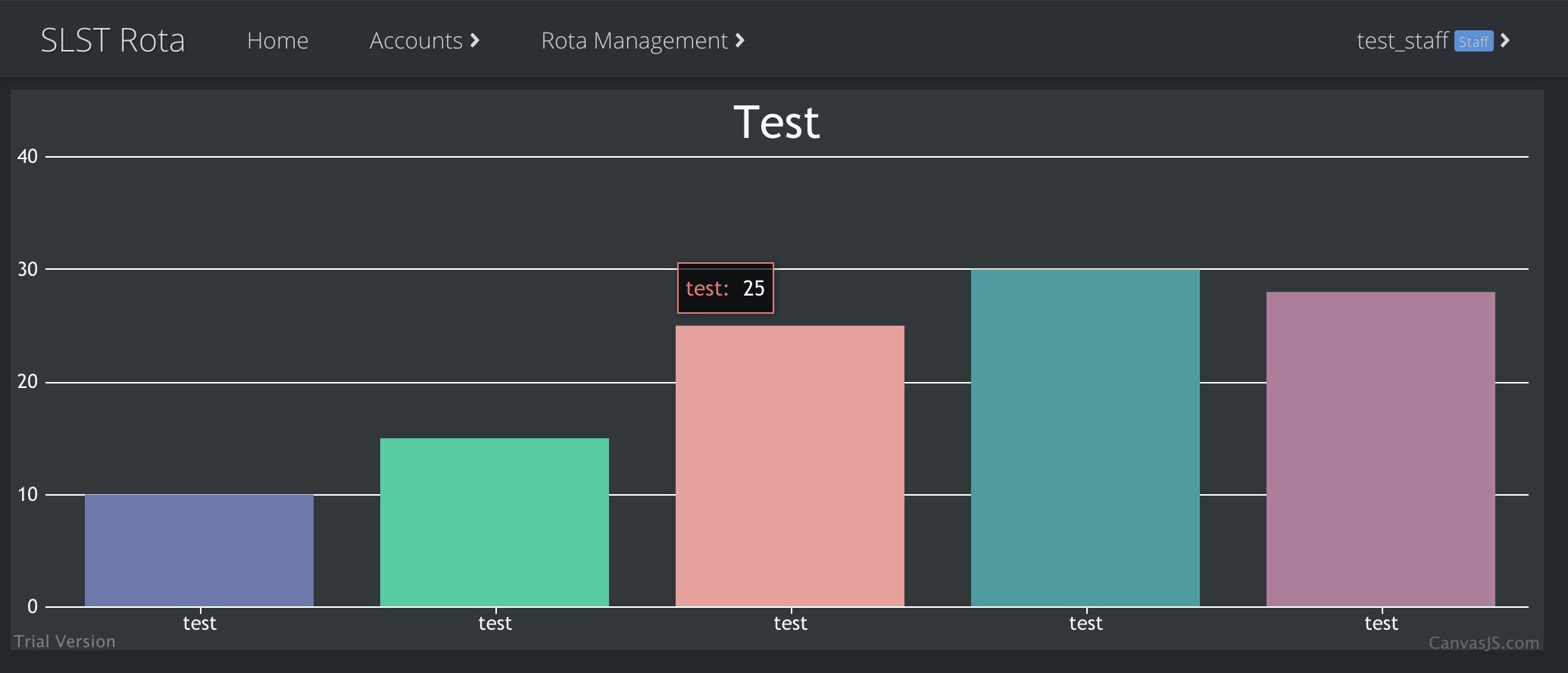
#### Development

To begin the development process for a graph, I decided to tackle the front end first. I imported the canvasjs script and setup a basic test graph to render. This is done within the window.onload JavaScript function to ensure it is only run once the script file has been loaded in.

The test graph contains 5 columns of differing heights. It should render in the dark theme with a title of “Test” above the graph. It will render in the “session\_attendance\_chart” div which has been inserted into the template body at the top.



Upon accessing the main attendance homepage for staff, the graph was rendered as expected and looked awesome.



To make the continued generation of graphs far easier in the templates without making lots of messy code, I decided to move the chart generation to a macro in Jinja2, the templating language I am using for this project. Doing this allows me to keep one copy of the long code required to render a graph and in each template, there can then be a single line to trigger the macro with the data required passed through from the python backend controller.

The macro code will be very similar to the test code written earlier but wrapped in a Jinja macro declaration and making use of variables passed instead of defined items. To output the datapoints, I decided it would be best to simply dump the data in json format using the python json.dumps method. This should produce the pythonic data in a safe format that JavaScript can make use of.



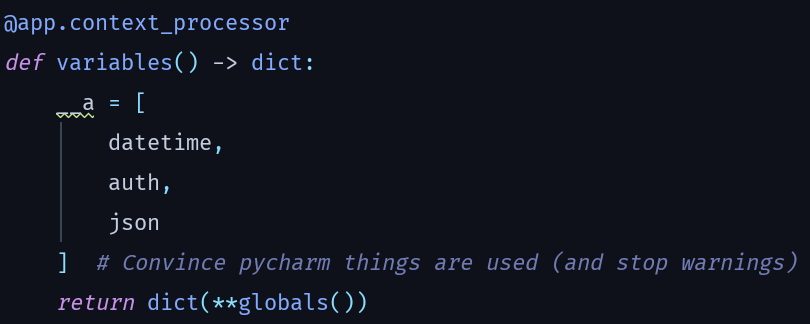
#### Testing the macro

With the macro code written, it was time to test it works properly before beginning work on generating the final data that will populate the attendance graph on the staff attendance homepage. To test this, I replaced the original test code in the template with a reference to the macro and passed through a single column to render in the graph.



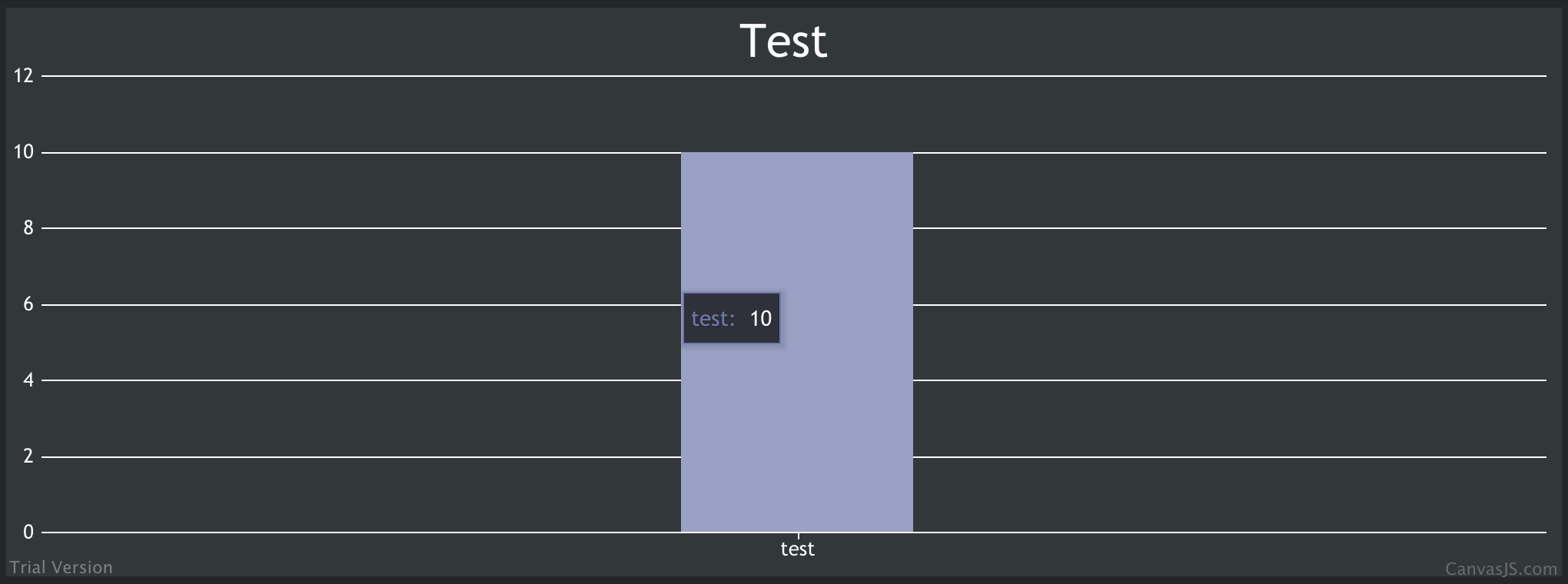
Upon restarting the web server and accessing the page, unfortunately I was greeted with an error from the Jinja template stating that the variable used in the macro, json, was undefined. I had forgotten that Jinja doesn’t by default have access to normal python functions and that they must be passed in by the context generator in the controller.





The fix for this was simple, importing json in the app controller and ensuring it is passed into context through the exploded globals. To ensure that PyCharm, my IDE of choice, doesn’t clean up and remove it, json it also held in an unused variable list.

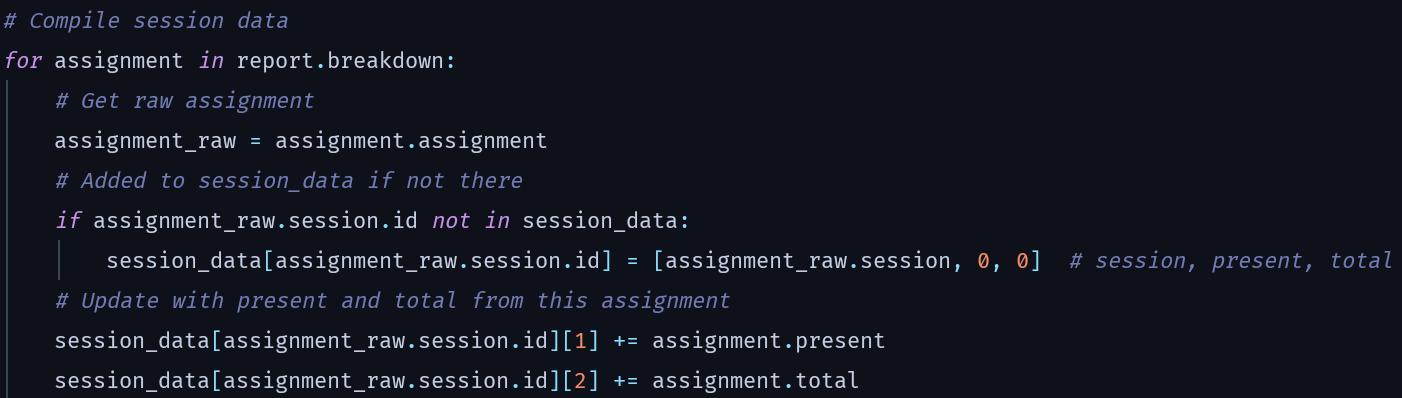
With this fixed, opening the attendance homepage results in the graph being rendered correctly with the single column of data that was passed in.



#### Data generation

The data that has to be generated needs to be total attendance and present attendance for each session in the system. The best way to do this in the home controller is to hijack the existing table generation and within that also generate all the data needed for the chart.

The StudentReport provides a breakdown attribute which contains each assignment that the report discovered for the student. By iterating over this within the table generation, I can access each and every AssignmentReport in the system. I can then create a total counter and a present counter for each session referenced by the assignment reports which can then be used to generate the final graph data.



To generate the final chart data, I need to iterate over the session data collected and parse it into the format needed for the chart to render. The chart expects an array of objects where each object contains a label value and a y co-ordinate value. I plan for the chart to show percentage attendance and so the y co-ordinate will be the present count divided by the total count times 100 to get a percentage. The label will be used to identify the session, day and start/end times.



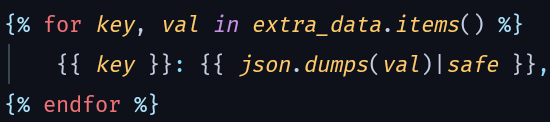
#### Testing the chart generation

With the generation done the chart data can be passed into the template and used by the macro to render the chart for the attendance homepage. With the current data in the database for the system I am expecting four sessions to show on the graph but with no columns above them as there currently is no attendance recorded in the database.



The chart had rendered correctly but the y axis was not to my satisfaction as it had automatically shrunk to only display 0 to 9. As this graph will show percentage based columns, I would like the chart to have a fixed axis of 0 to 100 at all times. Unsure of how to solve this, I referred to the CanvasJS documentation (https://canvasjs.com/docs/) in hope that they provided a way to fix an axis.

Under the axisY section of the chart documentation there is a minimum and maximum value that can be specified to provide a fixed axis. I now need to implement this into the chart that is being rendered on the page. As the macro controls the rendering I need to modify this to support custom extra data being given to the chart.

To achieve the ability to pass custom extra data into the macro an extra argument of a dictionary was added to the macro definition with a default value of a default dict. I can then iterate over this dictionary in the chart creation with the same json.dumps procedure as the data points to give the JavaScript the pythonic data.

Using this I can then pass through the custom y axis data to have a fixed scale of 0 to 100.



When viewing the page now the chart is rendered as I wanted it to with the fixed axis of 0% to 100% and the sessions at the bottom.

#### UI Improvements

With the graph now working I can move on to improving the overall UI of the attendance home page. The first step for me will be to give the graph a title and to reduce its width as it currently takes the whole screen width. Next to the graph I will then put buttons to quick access rota and account management for the staff logged in as I plan for this page to ultimately become the default staff page.

