SLST Precinct Management System

Matt Cowley 4807 – Royal Grammar School High Wycombe 52423

Table of Contents

1. Analysis 6
1.1. Initial Objectives 6
1.2. Problem Identification 6
1.2.1. Computational Solvability 7
1.3. Stakeholders 8
1.3.1. Tom Brownridge 8
1.3.1.1. What system do you currently use for recording attendance? 8
1.3.1.2. Have you considered using a digital solution for the precinct? 9
1.3.1.3. What do other precincts in school currently use? 9
1.3.1.4. What would you like in a new system for the precinct? 9
1.3.1.5. Are there any more advanced features you would like beyond just tracking if you were present or not? 9
1.3.2. Nick Balaam 9
1.3.2.1. I have spoken to Tom and he says there is no system in place for the precinct, can you confirm this? 9
1.3.2.2. Would you like to have a proper system in place to manage the students who monitor the precinct at school? 9
1.3.2.3. In a system, what features would you like to have? 10
1.3.2.4. Would you want the system to have fixed assignment of students to the precinct sessions or allow them to organise themselves and just register whenever they are there? 10
1.4. Problem Research 10
1.4.1. Excel 10
1.4.2. FindMyShift 11
1.5. Proposed Solution 12
1.5.1. Objectives 13

2. Design 17
2.1. Solution Decomposition 17
2.2. Solution Description 19
2.2.1. Possible Limitations 20
2.2.2. Algorithm Application 21
2.2.2.1. How the Algorithm Works 21
2.2.2.2. Student Allocation Implementation 22
2.2.2.2.1. Pseudocode 22
2.2.3. Code Structure & OOP 25
2.2.4. Design Overview 25
2.2.5. SQL – Database 27
2.2.5.1. Database Models UML Diagram 28
2.2.5.2. Database ERD 29
2.3. Testing Plan 29
2.3.1. During Development 29
2.3.1.1. Test Data 30
2.3.2. Post-Development Testing 30
2.3.2.1. Test Data 31

3. Development 32
3.1. Creating the base Flask app 32

3.2. Issue with Jinja2 templating 32
3.3. Setting up authentication 33
3.4. Beginning styling 33
3.5. Student rota view 34
3.6. Student unavailable days 36
3.7. Issue with navbar styling 36
3.8. Issue with updating unavailability 37
3.9. Archived sessions displayed throughout student portal 38
3.10. Beginning attendance tracking on the student portal 39
3.11. Issues with attendance 40
3.12. Attendance report 41
3.13. Password reset 41
3.14. Beginning the staff portal 42
3.15. Staff account management 42
3.15.1. Issue with auth level input 43
3.16. Account list 43
3.17. Create new account 43
3.17.1. Testing the inputs 44
3.18. Deleting an account 45
3.19. Rota control 45
3.20. Edit session 46
3.21. Create session 47
3.21.1. Testing 47
3.22. Updating assignments 48
3.22.1. Button design 48
3.22.2. Button generation 48
3.22.3. Button click events 49
3.22.4. JSON input updating 50
3.22.5. Updating the database 51
3.22.6. Showing unavailabilities 51
3.23. Automating Style Generation 53
3.24. Generating Rota Assignments Automatically 53
3.24.1. Fetching the next user to assign 54
3.24.2. Generating the assignments 54
3.24.3. Testing the assignment generation 55
3.24.4. Creating the front-end 57
3.24.5. The Hungarian Algorithm 60
3.24.5.1. Application in the rota system 61
3.25. Deleting sessions 61
3.25.1. Testing 62
3.26. Testing disabled accounts 63
3.26.1. Staff System 63
3.26.2. Student System 64
3.27. Automatic Student Assignment Issue 64
3.27.1. Testing the new method 65
3.27.1.1. Expected results 66
3.27.1.2. First issue 66
3.27.1.3. Second issue 67
3.28. Staff Attendance Report 68
3.28.1. Student Reports 68
3.28.2. Attendance Home 69
3.28.2.1. Design 69

3.28.2.2. Development ... 70
3.28.2.3. Testing ... 72
3.28.3. Graphs ... 73
3.28.3.1. Development ... 74
3.28.3.2. Testing the macro ... 76
3.28.3.3. Data generation ... 77
3.28.3.4. Testing the chart generation ... 78
3.28.3.5. UI Improvements ... 79
3.29. Student Attendance Graphs ... 81
3.29.1. Data generation ... 81
3.29.2. Graphing the data ... 82
3.29.2.1. Testing the updated macro ... 82
3.30. Creating the new graph ... 83
3.30.1.1. Styling the charts ... 85
3.30.1.2. Updating the overview graph ... 86
3.31. Site UI/UX Improvements ... 87
3.31.1. Iconography ... 87
3.31.1.1. Adding the icons ... 88
3.31.1.1.1. Error page (app/error.jinja2) ... 88
3.31.1.1.2. Footer (app/footer.jinja2) ... 89
3.31.1.1.3. Navigation bar (app/navbar.jinja2) ... 89
3.31.1.1.4. Staff attendance homepage (attendance/home.jinja2) ... 90
3.31.1.1.5. Login page (auth/login.jinja2) ... 92
3.31.1.1.6. All Accounts Management (staff/accounts.jinja2) ... 92
3.31.1.1.7. Automatic assignments generation (staff/automatic_assignments.jinja2) ... 93
3.31.1.1.8. Create a new account (staff/new_account.jinja2) ... 94
3.31.1.1.9. Staff full rota view (staff/rota.jinja2) ... 94
3.31.1.1.10. Delete rota session (staff/session_delete.jinja2) ... 95
3.31.1.1.11. Edit rota session (staff/session_edit.jinja2) ... 96
3.31.1.1.12. Student portal (student/index.jinja2) ... 96
3.31.1.1.13. Student rota view (student/rota.jinja2) ... 97
3.31.1.1.14. Student unavailability management (student/unavailability.jinja2) ... 98
3.31.1.1.15. Edit student availability (student/unavailability_edit.jinja2) ... 98
3.31.1.1.16. Default system homepage (index.jinja2) ... 99
3.31.2. Sorting table lists ... 100
3.31.2.1. Table rendering macro ... 100
3.31.2.2. The JavaScript sorting ... 101
3.31.2.2.1. Headers ... 101
3.31.2.2.2. Rows ... 102
3.31.2.2.3. Events and styles ... 103
3.31.2.2.4. Icons ... 104
3.31.2.2.5. Click event and sorting ... 105
3.31.2.3. Testing ... 108
3.31.2.4. Default sorting ... 110
3.31.2.4.1. Testing ... 110
3.31.2.5. Other tables ... 111
3.31.3. Navigation bar ... 113
3.31.4. Edit assignments ... 113
3.31.5. Checkbox styling ... 114
3.31.5.1. HTML changes ... 115
3.31.5.1.1. Testing ... 116
3.31.5.2. Styling changes ... 116
3.31.5.2.1. Testing ... 117
3.31.5.3. JavaScript changes ... 117
3.31.5.3.1. Testing the selection ... 118
3.31.5.3.2. Event handling ... 118

3.31.5.3.3. Final testing 119
3.31.6. Graph changes 119
3.31.6.1. Attendance per session 119
3.31.6.2. Punctuality graph 120
4. Testing & Evaluation 122
4.1. Testing to inform evaluation 122
4.1.1. Logging in 122
4.1.1.1. Bad data 123
4.1.1.2. Extreme data 123
4.1.1.3. Good data 124
4.1.2. Creating new rota sessions 124
4.1.2.1. Bad data 124
4.1.2.2. Extreme data 126
4.1.2.3. Good data 126
4.1.3. Editing a rota session 126
4.1.4. Deleting a rota session 127
4.1.5. Creating an account 128
4.1.5.1. Bad data 128
4.1.5.2. Extreme data 129
4.1.5.3. Good data 130
4.1.6. Editing an account 131
4.1.6.1. Staff member – Own account 132
4.1.6.2. Staff member – Another account 133
4.1.6.3. Student account – Own account 134
4.1.7. Updating unavailability 135
4.1.7.1. Extreme data 135
4.1.7.2. Bad data 136
4.1.7.3. Good data 136
4.1.8. Updating assignments 136
4.1.8.1. Main page 137
4.1.8.2. Setting an assignment 137
4.1.8.3. Removing an assignment 138
4.1.9. Automatic assignments 139
4.1.10. Signing in/out 140
4.1.10.1. Signing in 141
4.1.10.2. Allowing a student to sign in early 141
4.1.10.3. Signing in early 142
4.1.10.4. Signing out early 142
4.1.10.5. Signing out automatically 143
4.1.11. Attendance report 143
4.1.11.1. Generating test data 144
4.1.11.2. Testing test data generation 146
4.1.11.3. Student attendance report 147
4.2. Stakeholder review 148
4.2.1. Nick Balaam 148
4.2.1.1. Fixing the issues raised 149
4.2.1.1.1. Overlapping sessions 149
4.2.1.1.2. Removing the auth level number 152
4.2.1.1.3. Fixing punctuality graph 152
4.2.2. Tom Brownridge 154
4.2.2.1. Fixing issues mentioned 155
4.2.2.1.1. White outline button styling 155
4.2.2.1.2. White table borders 155
4.2.2.1.3. Table subheading backgrounds 156

4.3. Evaluation of the solution 158
4.3.1. Provide an easy to use rota system for students on duty in the precinct 158
4.3.2. Provide an easy management system of the rota system for staff 159
4.3.3. Allow students to indicate days they cannot complete precinct duty 159
4.3.4. Allow students to sign in and out of their current precinct duty and for this to be tracked 160
4.3.5. Display an easy to understand timetable of duties for the precinct 161
4.3.6. Allow staff to define the precinct duty sessions for the week 162
4.3.7. Allow staff to view when students signed in and out 163
4.3.8. Allow staff to view in-depth analytics regarding student attendance for their duties.... 164
4.3.8.1. Further development ideas 164
4.3.9. Display to staff the attendance levels for individual students on the precinct, the student's reliability 165
4.3.10. Display to staff the duration each student stays for their precinct duty, percentage of defined precinct duration 165
4.3.11. Provide an authentication system for staff and students 166
4.3.11.1. Further development ideas 167
4.3.12. Allow staff to easily manage student accounts and other staff accounts 167
4.3.13. Final thoughts 169
4.3.13.1. Maintenance of final solution 169
4.3.13.1.1. Further development to resolve maintenance issues 170

1. Analysis

1.1. Initial 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

1.2. Problem Identification

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.

1.2.1. Computational Solvability

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 such as graphs to display attendance and reliability as well as showing reports about the users 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.

1.3. Stakeholders

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.

To better understand the requirements for the system, I decided to interview two of the actual stakeholders for the final system. Tom Brownridge who is one of the students that is responsible for the precinct and Nick Balaam who is the staff member in charge of the precinct at school.

1.3.1. Tom Brownridge

Tom is the current precinct monitor for the stage at school and so he is a perfect stakeholder for the system. With this in mind, I asked him questions about what he'd like to see in a precinct system to make his life easier as well as the other precinct monitors.

1.3.1.1. What system do you currently use for recording attendance?

We don't have anything at the moment. Sometimes we will just write down in notes who was here but nothing proper. I guess it would be nice to have something better.

1.3.1.2. Have you considered using a digital solution for the precinct?

Not really, everything online was too complicated to set up for just our stage precinct. If it was easy, generally it cost money which we don't want to use.

1.3.1.3. What do other precincts in school currently use?

I talked to a few friends who also are precinct monitors and most just sign in with their staff member but Balaam isn't around to do ours, he trusts us to track it ourselves.

1.3.1.4. What would you like in a new system for the precinct?

I really like the new school system for signing in for lessons. We just scan our fingerprint on the reader and then press the button for the lesson to sign in. If we need to sign out, it also has that with a simple button.

It would be cool if we could have something like that for the precinct where we just need to log in to something online and then press a button to sign in and out.

1.3.1.5. Are there any more advanced features you would like beyond just tracking if you were present or not?

Well if it was digital I guess you could make it so that it could also show each of us our own attendance over time, in like a graph or something?
A full preview of the rota and which of us are assigned on each day would make it easier as well instead of us getting confused as to who is on duty each day.

1.3.2. Nick Balaam

Nick is the staff member than runs the stage team and is the staff member who oversees the stage precinct. All precinct monitors report to him for monitoring the stage area during the week at school.

1.3.2.1. I have spoken to Tom and he says there is no system in place for the precinct, can you confirm this?

Yes, currently the boys have no way to track who does the precinct, they just take it in turns and I trust them that there is always one of them there handling the precinct.

1.3.2.2. Would you like to have a proper system in place to manage the students who monitor the precinct at school?

Whilst I don't have a way currently, if it is possible, it would be great to have something that allows the boys to sign in and organise themselves.

1.3.2.3. In a system, what features would you like to have?

It would be great if I could set up the rota layout in the system, as a website maybe, which the students could also log into and mark themselves as present for the sessions.

Currently they all just organise themselves and cover for each other when needed, but it would be nice if I could properly allocate students to each precinct session at school. I imagine from this data you could also generate proper reports of the attendance of each student for me?

1.3.2.4. Would you want the system to have fixed assignment of students to the precinct sessions or allow them to organise themselves and just register whenever they are there?

I think it would be better if I could set it up so that the students had fixed assignments. If we had a system in place, I feel it should be organised properly as opposed to letting the students work it out themselves. When it starts out though, it would be nice if the students had a system to let me know what they couldn't do before I assigned them all.

With these interviews completed, I had a better idea of what exactly the stakeholders in the system are looking for and I am pleased to see that my initial objectives align pretty well.

Once I have looked further into the existing solutions available I should be able to create a strong outline of objectives for the development of the final system.

1.4. Problem Research

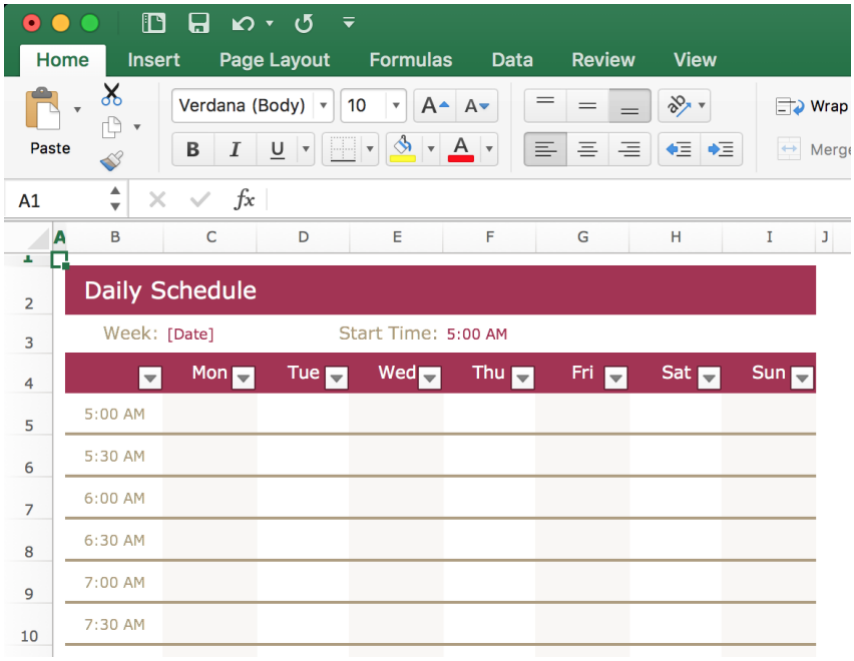
1.4.1. Excel

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 sign in and out for duties.



In addition, using Excel as a solution to the rota would make it very hard to track attendance and to produce useful, analytical reports to staff which the system I am developing intends to do to allow for easy administration of the precinct the rota is responsible for.

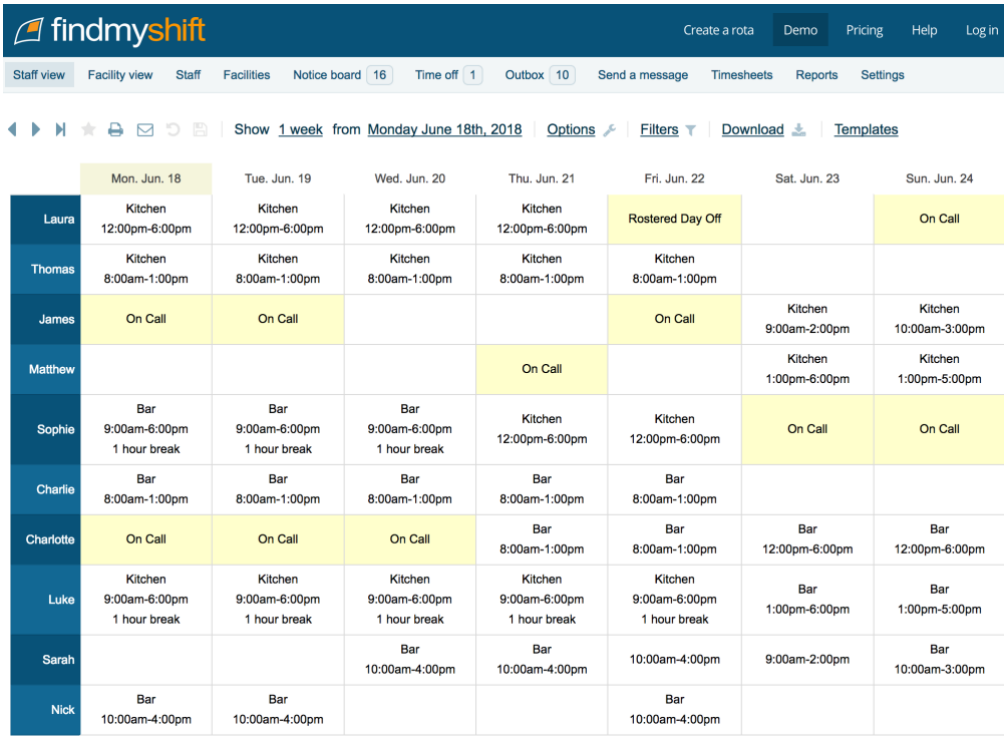
This solution may be possible to achieve and use with very complicated VBA embedded into the document but this is not a practical solution. The project aims to make the precinct management process easy and simple unlike this.

1.4.2. 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.



	Mon. Jun. 18	Tue. Jun. 19	Wed. Jun. 20	Thu. Jun. 21	Fri. Jun. 22	Sat. Jun. 23	Sun. Jun. 24
Laura	Kitchen 12:00pm-6:00pm	Kitchen 12:00pm-6:00pm	Kitchen 12:00pm-6:00pm	Kitchen 12:00pm-6:00pm	Rostered Day Off		On Call
Thomas	Kitchen 8:00am-1:00pm	Kitchen 8:00am-1:00pm	Kitchen 8:00am-1:00pm	Kitchen 8:00am-1:00pm	Kitchen 8:00am-1:00pm		
James	On Call	On Call			On Call	Kitchen 9:00am-2:00pm	Kitchen 10:00am-3:00pm
Matthew				On Call		Kitchen 1:00pm-6:00pm	Kitchen 1:00pm-5:00pm
Sophie	Bar 9:00am-6:00pm 1 hour break	Bar 9:00am-6:00pm 1 hour break	Bar 9:00am-6:00pm 1 hour break	Kitchen 12:00pm-6:00pm	Kitchen 12:00pm-6:00pm	On Call	On Call
Charlie	Bar 8:00am-1:00pm	Bar 8:00am-1:00pm	Bar 8:00am-1:00pm	Bar 8:00am-1:00pm	Bar 8:00am-1:00pm		
Charlotte	On Call	On Call	On Call	Bar 8:00am-1:00pm	Bar 8:00am-1:00pm	Bar 12:00pm-6:00pm	Bar 12:00pm-6:00pm
Luke	Kitchen 9:00am-6:00pm 1 hour break	Kitchen 9:00am-6:00pm 1 hour break	Kitchen 9:00am-6:00pm 1 hour break	Kitchen 9:00am-6:00pm 1 hour break	Kitchen 9:00am-6:00pm 1 hour break	Bar 1:00pm-6:00pm	Bar 1:00pm-5:00pm
Sarah			Bar 10:00am-4:00pm	Bar 10:00am-4:00pm	10:00am-4:00pm	9:00am-2:00pm	Bar 10:00am-3:00pm
Nick	Bar 10:00am-4:00pm	Bar 10:00am-4:00pm			Bar 10:00am-4:00pm		

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.

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.

1.5. Proposed Solution

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.

1.5.1. 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
            - *Nb: Username/email & password*
            - Success
                - *Nb: Go to home/default view for student (personal rota)*
            - Failure
                - *Nb: Redirect to login form with error message*
        - Logout
            - *Nb: Wipe session and redirect to login page for next user*
        - Password Reset
            - *Nb: Username/email input and reset button, will send email with reset token/link*

                - 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
                - *Nb: Will be displayed as a table to allow for best formatting of the data*
                - Personal view showing on current student assignments
                    - *Nb: Also, show basic student stats on this page for them to see (attendance percentage etc)*
                - Full view showing all students' assignments
            - 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
            - *Nb: A simple form to set the day of the week for the session, the start time (hours & minutes) as well as the end time*
        - Edit students assigned to sessions

        - *Nb: Warn if student is marked as unavailable on manually assigned session*
    - Generate automatic rota configuration
        - *Nb: Will take user to a form to set number of students per session and toggle for forcing assignments (ignoring unavailability)*
- 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

2. Design

2.1. Solution Decomposition

Alongside the full set of objectives for the final software solution outlined in **section 1.5.1**, this decomposition details how these objectives will be developed as I work through the iterative development process to create the planned computational solution.

1. Create the base Flask application
    a. Basic flask application loading config file
        i. Template loading directory - config
        ii. Static files directory – config
        iii. Local file database – app.db
        iv. CSRF secret/session keys
    b. Compiling SASS styling
        i. Empty sass files/structure
        ii. Compiling script in Python
    c. Interacting with the database
        i. Database engine controller in Python
    d. Setup global template context
        i. Access to Python globals
        ii. Additional common modules (Datetime, json etc.)
    e. Basic flask error handler
        i. Lookup code messages from API and generate pretty error page for user
2. Setup basic templating for the project
    a. Base template for all pages
        i. Dark mode – better for eyes + matches the theme of backstage/tech
        ii. Top navigation bar
        iii. Footer with copyright info
    b. Index page template using the base template
        i. Testing templating is working correctly + page to render during initial development
3. Create authentication module (staff + student access)
    a. Create user database model
        i. Id, username, password, email etc.

        ii. As a Python class definition, table automatically generated by database engine
    b. Create flask blueprint
    c. Create form class from WTForms
    d. Produce login route with form handling
        i. Full validation of all inputs (done client side as well)
        ii. Redirect to default view if login successful
        iii. Render template normally (with errors displayed if necessary)
    e. Create login template in Jinja with form
        i. Ability to display appropriate errors from server side validation
    f. Create logout route, redirect to index
4. Create basic base templates and styling for the site
    a. Begin to write major styling in SASS and include on templates
        i. Make use of skeleton css framework
        ii. Use custom components in SASS for styling
    b. Use framework from styling in templates to provide clean layout
5. Create required database models for student rota (sessions, assignments)
    a. Produce ERD
6. Create student rota view mock-up
7. Student rota routes and generation
8. Create db models for unavailability
    a. Produce ERD
9. Create views & post routes for unavailability
10. Create db model for attendance
    a. Produce 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
    a. Request reset page
        i. Generates one time token and sends to user
    b. Reset page checking token and setting new password

15. Begin staff controller
16. Staff account management
    a. Individual account edit
    b. Account list
17. Create new account

    a. Test inputs
18. Delete accounts
    a. Disable account in database
19. Rota
    a. Current full rota view
    b. Edit button for each session & new session button
        i. Test inputs
    c. Edit page allows users to be assigned or unassigned
    d. Show student unavailabilities on edit page
    e. Auto generate button on rota view
        i. Find user to assign function
        ii. Assign users to sessions function
        iii. Front end control for function
        iv. Test page
        v. Discuss Hungarian algorithm
    f. Ability to delete sessions (must remove assignments)
20. Test disabled accounts
    a. Fix routes that fail to check for disabled accounts
21. Attendance report
    a. Student overview report
        i. Scatter graph for in diff and out diff per assignment
    b. Per assignment report for each student
    c. All students overview (home)
22. Staff home page
23. UI improvements
    a. Iconography throughout site
    b. Sort (and provide filters for) staff account list by not disabled, staff only, students only
    c. Make everything easier to understand, general ui tidy

2.2. Solution Description

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. Communication with the SQL based database is handled automatically by the database engine being used in Python, SQLAlchemy and so I only need to make use of Python models to store data. 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. Again, all these changes will be saved to the database storage by the database engine once I have updated the relevant objects in Python.

2.2.1. Possible Limitations

Due to the nature of the system being created with a web interface, this may pose limitations for accessibility of the rota to everyone on the team.

For final deployment, the school will need to be able to host the software package on a web server within the school that all students and staff on the SLST can access through any device on the network. The site is designed to be operated with a mouse or through a touchscreen so that both laptops and iPads can be used to interact with the system.

However, instead of this, it could be hosted on a private virtual network within the school and only be accessible through dedicated touchscreen units located around the school, much like the recently deployed student portal devices that allow students to sign in for study periods themselves.

Further, the system, in its current proposed form would not integrate with the existing authentication system within the school and will come with its own packaged solution which may lead to confusion for staff and students not trained on usage of it. This may be something that could be implemented through further development with the school.

2.2.2. 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 an advanced resource allocation algorithm, most likely one based around the Hungarian Algorithm.

The Hungarian Algorithm handles resource allocation by using a cost factor for resources which I can augment from the availability of each student, which can be represented as 1 or 0 dependant on whether the student is available for each session in the rota. Using a resource allocation algorithm ensures that students are best automatically assigned to the sessions in the rota without issues.

2.2.2.1. How the Algorithm Works

The problem that the Hungarian algorithm is written to solve can normally be represented in a matrix layout. In the example, I will consider the cost of multiple contractors and multiple jobs for a company.

	Job 1	Job 2	Job 3
Contractor 1	$4m	$6m	$7m
Contractor 2	$2m	$1m	$4m
Contractor 3	$9m	$11m	$7m

Step 1 of the algorithm is to iterate over each row, subtracting the cheapest cost from all costs in that row.

	Job 1	Job 2	Job 3
Contractor 1	0	$2m	$3m
Contractor 2	$1m	0	$3m
Contractor 3	$2m	$4m	0

Step 2 is then to do the same, but iterating over each column instead of row.
In this case, each column contains a zero so nothing changes.

Step 3 is to then work out the minimum number of lines that are needed to cover all the zeros. For this graph, each row simply becomes covered by a line.

	Job 1	Job 2	Job 3	
Contractor 1	0	$2m	$3m	X
Contractor 2	$1m	0	$3m	X
Contractor 3	$2m	$4m	0	X

With that done, the algorithm knows it is done as the line count is the same as matrix size. After this, the best assignment layout can be created. For this, Contractor 1 would handle job 1, Contractor 2 would handle job 2 and the third contractor would do the third job.

2.2.2.2. Student Allocation Implementation

To implement this into my program properly, I wrote my own implementation of the methods described in the Hungarian algorithm, treating the costs of the resources (students) as the number of assignments that they currently had. This makes the system most fair as it generates all the assignment allocations.

2.2.2.2.1. Pseudocode

```
// Get the next user that we can assign to a session based on previous assignments of all users
function nextAssignableStudent(totalAssignments:byRef, session:byVal, force:byVal)
  // Abort if we have no users
  if totalAssignments.length == 0
    return null
  end if

  // This will be the next assignable user
  nextUser = null

  // Sort the values of the dictionary totalAssignments by the first item in each subarray
  usersSorted = sort(totalAssignments.values, by x[0])

  // Loop over the sorted user data
```

```
    for i = 0 to usersSorted.length
        // Get the current user data and user
        userData = usersSorted[i]
        user = userData[1]

        // If they have already been assigned, skip
        if userData[0] > 0
            continue
        end if

        // Check if unavailable for this session
        unavailabilities = database.unavailability.get(session.id, user.id)
        if unavailabilities.length > 0
            continue
        end if

        // They're good, accept and break
        nextUser = userData
        break

        // Continue loop
        next i
    end for

    // If we haven't found any user that can be accepted and the force flag is set, select the
user that has least assignments so far
    if nextUser == null AND force == true
        nextUser = usersSorted[0]
    end if

    // If we have an accepted user, update their assigned stat
    if nextUser != null
        // Scrap the extra data, just the user itself
        nextUser = nextUser[1]

        // Update assignment stat
        totalAssignments[nextUser.id][0] = totalAssignments[nextUser.id][0] + 1
    end if

    // Done, so return what is hopefully a user (or null)
    return nextUser
end function

// Assign the target number of students to a session
function generateSessionAssignments(session:byRef, totalAssigned:byRef,
targetStudents:byVal, force:byVal)
    // Target number to assign
```

```
    target = targetStudents

    // Loop until we hit target or run out of students
    while target > 0
        // Get the next student to assign
        student = nextAssignableStudent(totalAssignments, session, force)

        // If we have no student, we have no hope, so abort
        if student == null
            break
        end if

        // We have a student, so update target
        target = target – 1

        // Save the target to the database
        database.assignments.create(session.id, user.id)
    end while
end function

// Assign students to all sessions in the system
function generateAllAssignments(targetStudents:byVal, force:byVal)
    // Get all sessions, abort if there are none
    sessions = database.sessions.all()
    if sessions.length == 0
        return
    end if

    // Get all student users, again, abort if there are none
    users = database.users.all(student=true)
    if users.length == 0
        return
    end if

    // Delete all old assignments as we're creating a fresh set of them
    assignments = database.assignments.all(removed=null)
    for i = 0 to assignments.length
        // Set removed attribute to the date/time now
        database.assignments.update(assignments[i].id, removed=datetime.now())

        // Continue loop
        next i
    end for

    // Create the blank total assigned stat data
    totalAssigned = []
    for i = 0 to users.length
```

```
        // Get the user
        user = users[i]

        // Add to dict with 0 assignments
        totalAssigned[user.id] = [0, user]

        // Continue loop
        next i
    end for

    // Generate assignments for each session
    for i = 0 to sessions.length
        generateSessionAssignments(sessions[i], totalAssigned, targetStudents, force)

        // Continue loop
        next i
    end for
end function
```

2.2.3. Code Structure & OOP

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 does however 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.

2.2.4. 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.

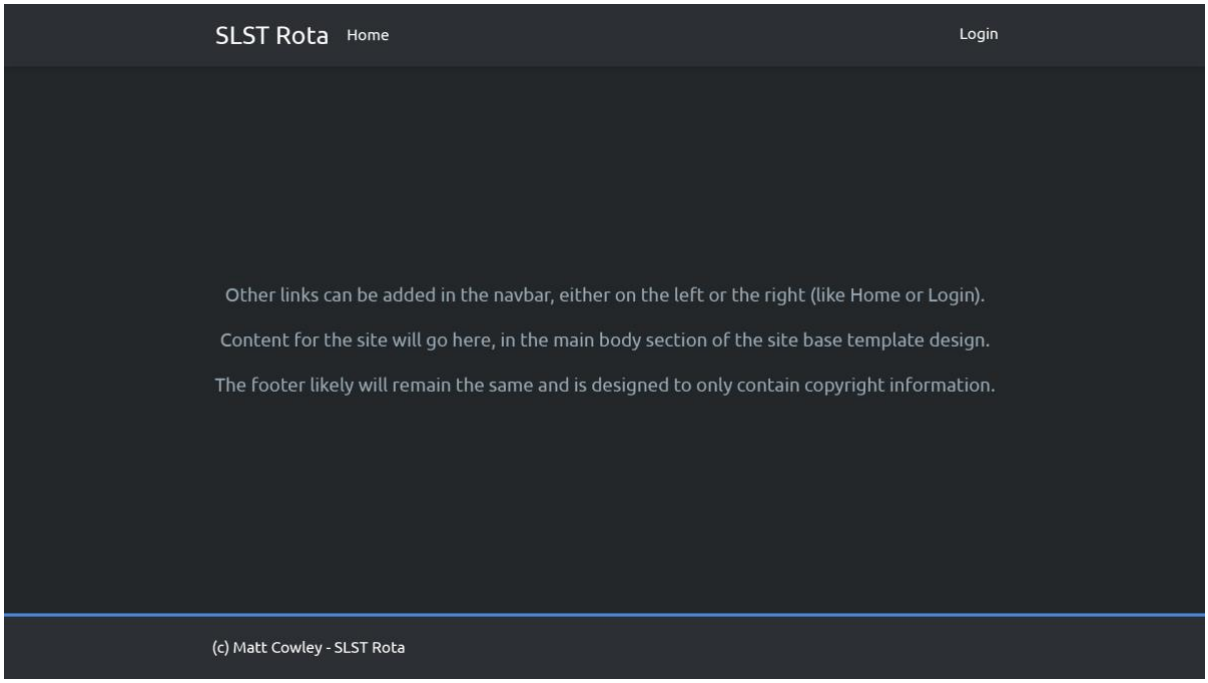
Two dark colours were picked to be used as the backgrounds for the majority of the site with a lighter grey also being picked out for accent elements that won't be blue.

These colours provide a good contrast ratio for the site, making the text easy to read on the dark background. Other colours used also provide good contrast against the background ensuring all elements will be visible to the user. Using a dark background with light text is much easier on the eyes and fits the theme of stage well.

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, simple to read for most users 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 (https://fonts.google.com/) online service such as Open Sans.



In the website, the font library Font Awesome (https://fontawesome.com/) 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)'.

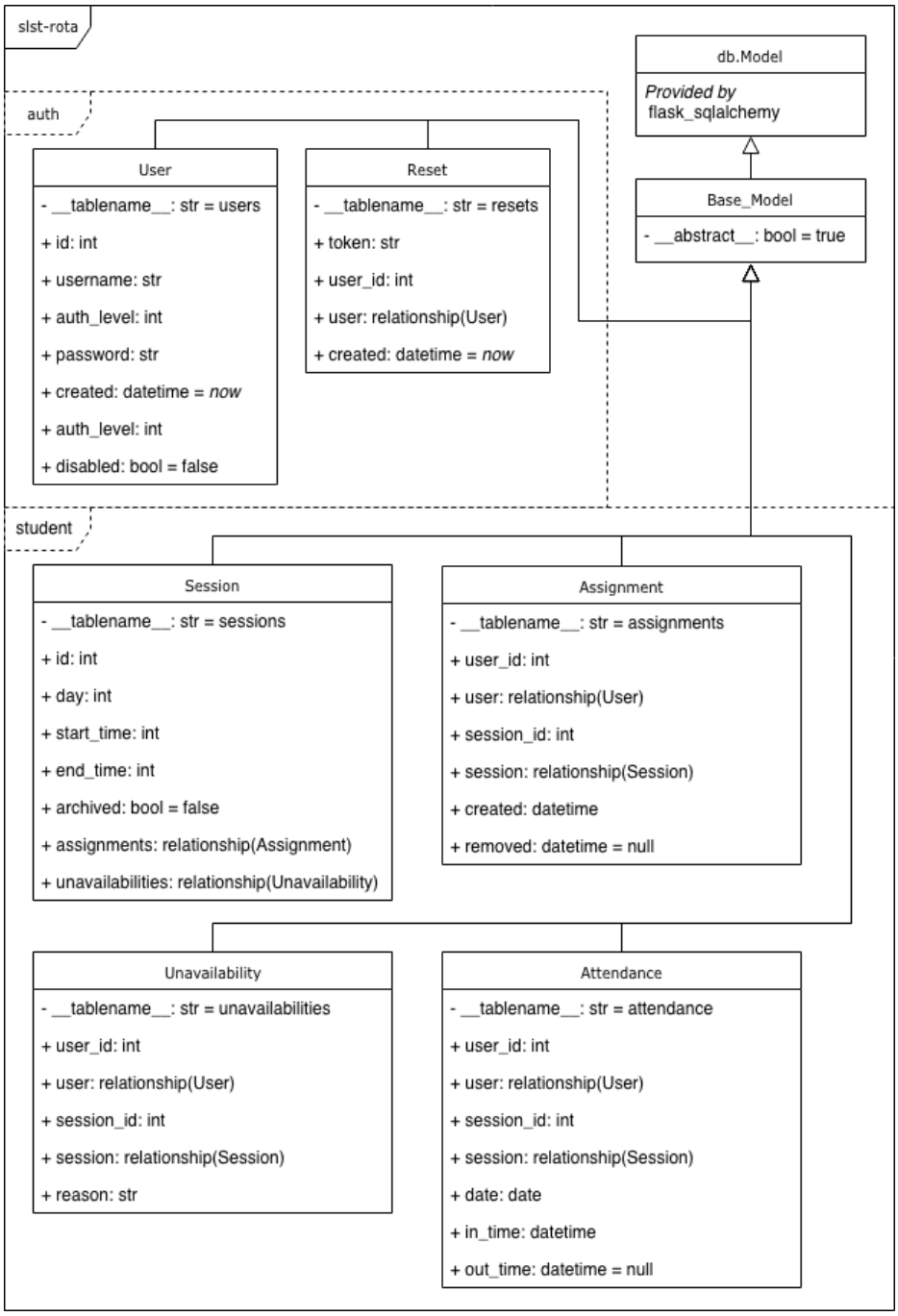
2.2.5. SQL – Database

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.

These diagrams show the planned final layout of the database and all the tables, which will be fully normalised, complying with 3rd normal form. The use of primary keys and foreign keys throughout the design of the database ensures that integrity of all links between tables are maintained when one item of data it updated or removed.

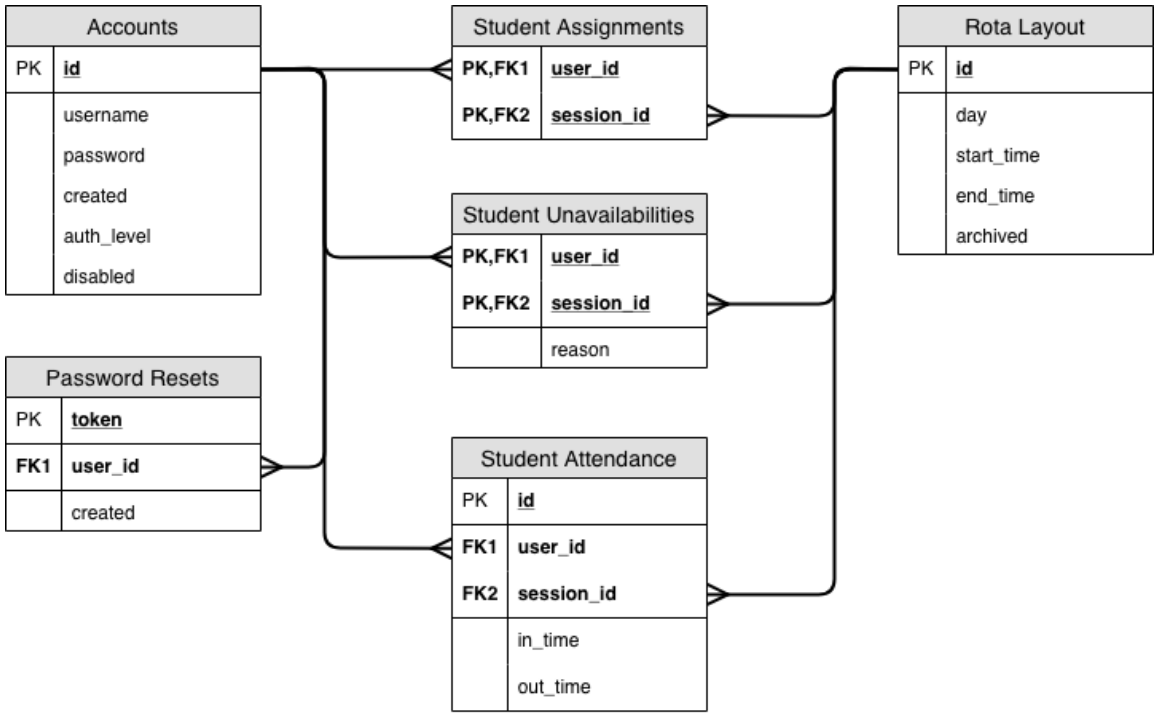
2.2.5.1. Database Models UML Diagram



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.

2.2.5.2. 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.

2.3. Testing Plan

2.3.1. During Development

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 and understandable error for the user where appropriate, whilst also maintaining data consistency if errors do occur.

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.

2.3.1.1. Test Data

During the development process, test data will be chosen for each test as the test is carried out. I decided this was a better strategy than attempting to define all the test data now as is it likely that inputs may change from the original plan, so keeping the test data dynamic up until the point at which I perform the testing is better.

In each situation, for each section of the development, tests will be carried out using data that will be considered normal (such as a valid, known username for a login field) as well as bad data (an unknown username in a login field) alongside an example of extreme data designed to escape the checks in the software (such as an SQL escape character to perform SQL injection).

Any errors will be recorded in my development log with screenshots, ensuring that they are then fixed correctly and fully so that the final software solution works correctly and effectively.

2.3.2. Post-Development Testing

Once the development is completed, further testing must be carried out to ensure the completed product is fully functional without any unexpected errors. This will first be carried out by myself, where I will access every page and use every feature available to both staff and student user accounts, testing that all features and inputs

work as expected. All input data as well as the expected result and actual result for each test will be documented as this testing is carried out. Normal, bad and extreme data will be tested on all inputs where applicable and all other functionality will be tested to ensure it does not fail.

Once my testing is completed, closed beta testing can be carried out on the solution. 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, in this case the two stakeholders originally interviewed for the proposed solution. This will provide a new test of the software as these users 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 these users can then be used to finalise the design and functionality of the system before it is released in a public 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 in further development of the computational solution to create the finished product which can be released for live use by everyone.

2.3.2.1. Test Data

As with the interactive development plan for testing, I have decided that it will be better to define the test data for the final testing once I am at that stage, so that it will best fit the inputs in the system after I have finished the development process. This is due to the fact that inputs may change from the current design and plan, so any planned test data may not fully cover them.

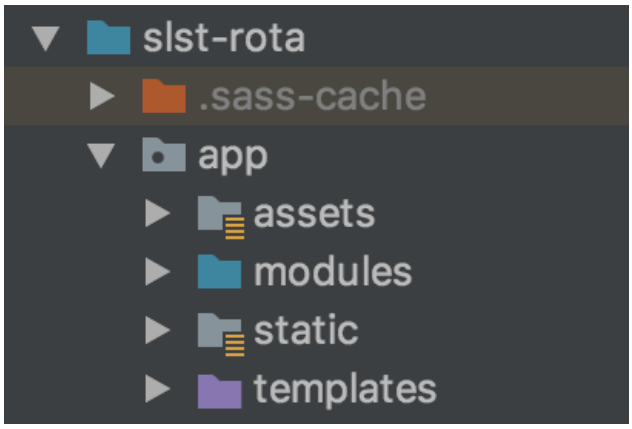
For the final testing, every section of testing will make use of a table that will record the type of data used (normal, bad or extreme) as well as the expected result from the test, the actual result of the test and any comments I make on the result of the test.

Any issues found, much like in the iterative development section, will be logged, fixed and then tested again to ensure the program is functioning perfectly.

3. Development

3.1. Creating the base Flask app

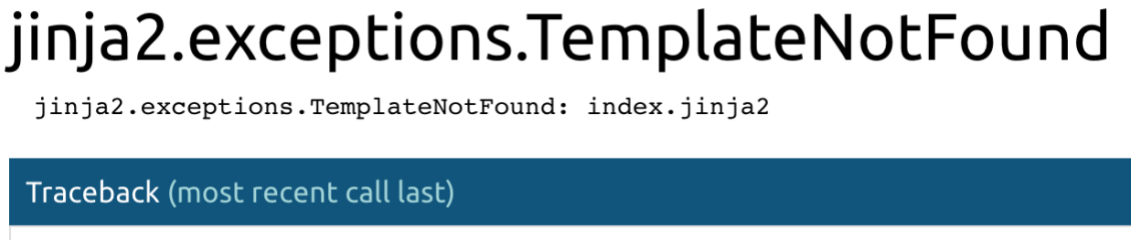
To begin the process of making the SLST rota system, a base Flask app had to be created.



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.

3.2. 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.

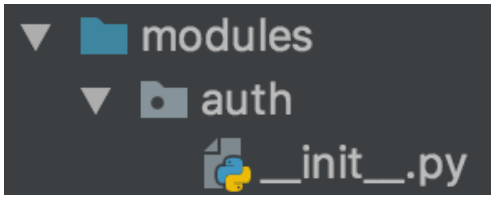
```
app = Flask("SLST Rota")
loader = jinja2.ChoiceLoader([
    app.jinja_loader,
    jinja2.FileSystemLoader(os.path.join(os.path.dirname(os.path.abspath(__file__)), 'templates')),
])
app.jinja_loader = loader
```

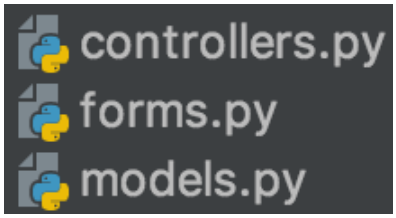
This issue was resolved by implementing a custom Jinja2 loader for the Flask app which was set to provide an absolute path to the templates folder being used for this project. This absolute path was created using the os module to ensure it worked

cross platform and would base itself off of the location of the current file to ensure portability.

3.3. 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.

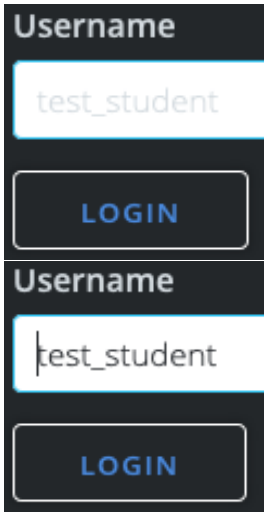
3.4. Beginning styling

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.

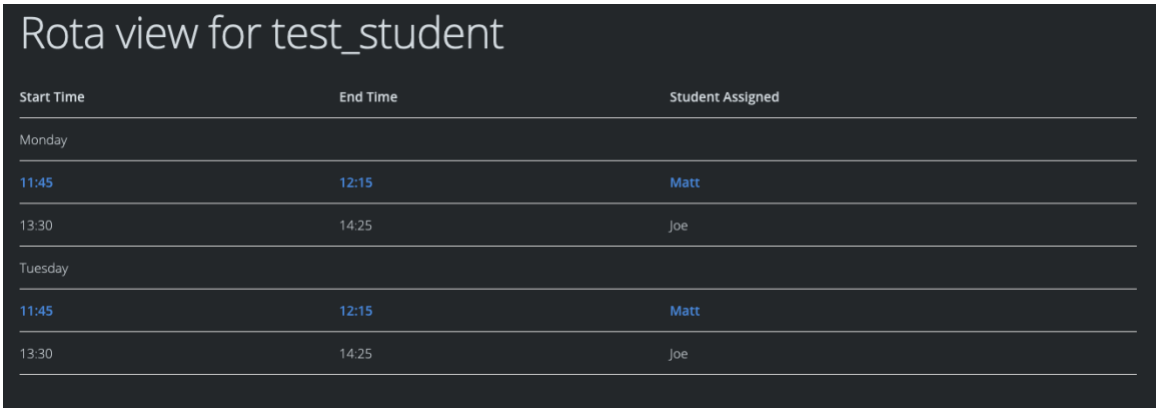
Once the frameworks were set up, I began writing the basic global styling that will be used throughout the site. This comprised of a few component files; base contained styles that will provide the base of the site on all pages, footer provides the styling that will create the footer used in the base 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.

3.5. Student rota view

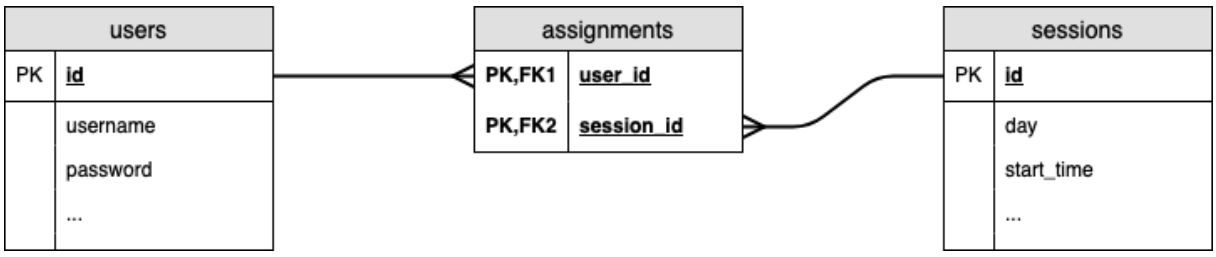
The key part of the site is the rota view provided to both the students and staff. To begin this process, a frontend only mock-up of what the rota view would look like was created. This has no link to any database tables or backend logic and was simply to test layout and design.



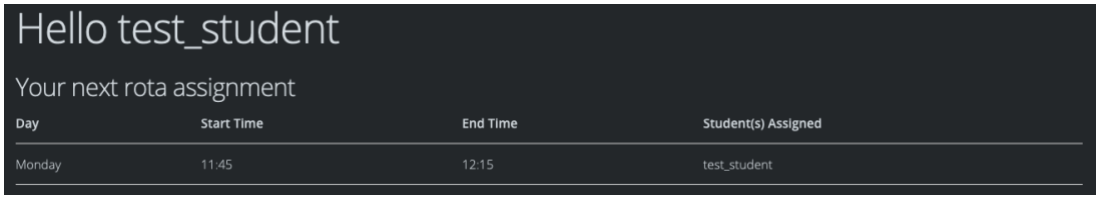
The concept of the design here is that each day is shown and within that each session is shown with a simple start and end time along with the student assigned. There is a highlight class created which is shown in two rows here that will act as a highlight when the student is looking at a view of the 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.

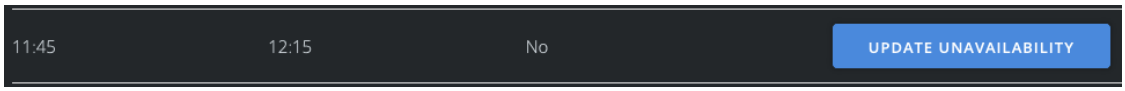


Once the rota view was setup, a basic student portal homepage was created which at this time simply displays the next assigned session for the student currently logged in.



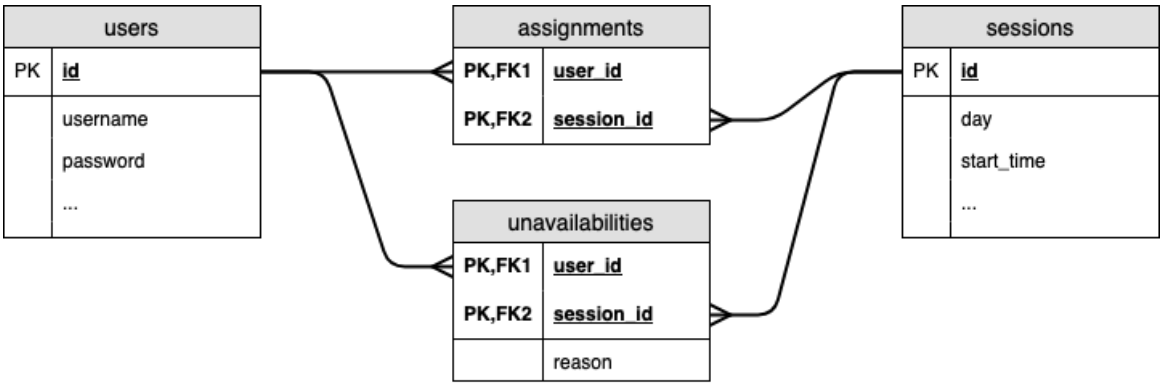
3.6. Student unavailable days

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.

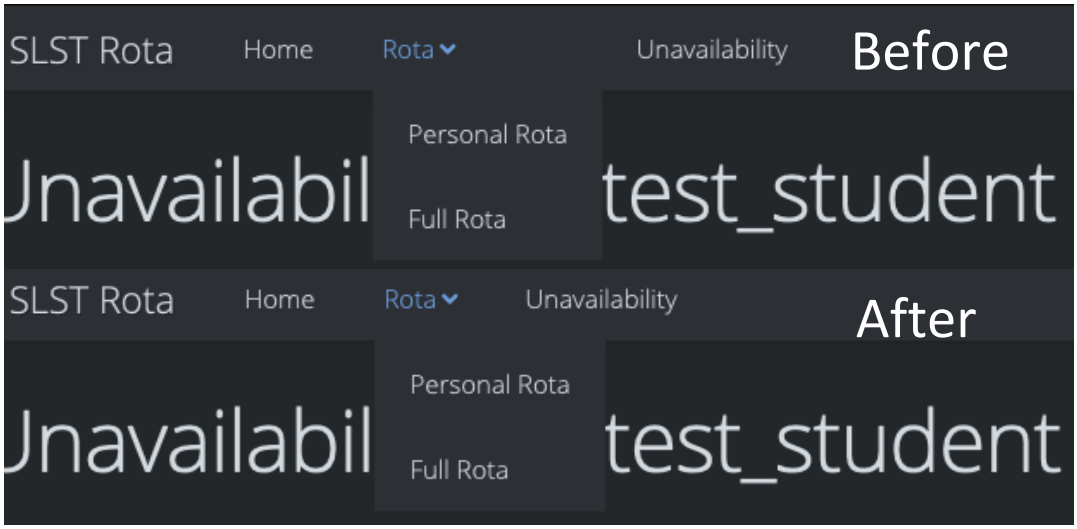
Now that this is done, the entity relationship diagram from earlier can be updated to also include the new unavailabilities table that had been created.



3.7. 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

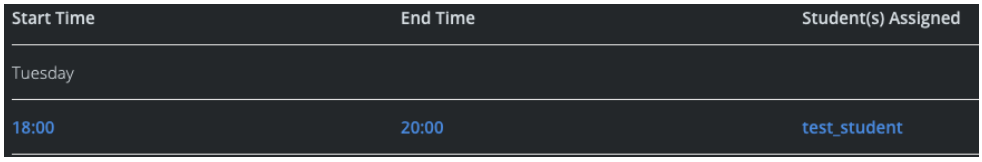
some flex elements and using standard block ones, the dropdown behaved correctly and I could move on.



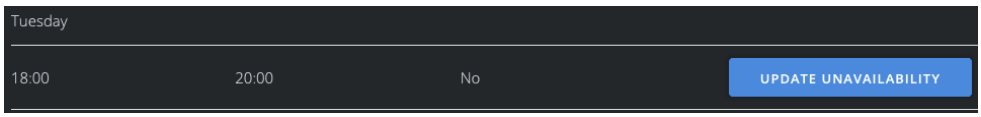
3.8. Issue with updating unavailability

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:

```
for session in data:

    # Don't show assigned sessions
    assigned = False
    if user.id in [f.user.id for f in session.assignments]:
        assigned = True
```

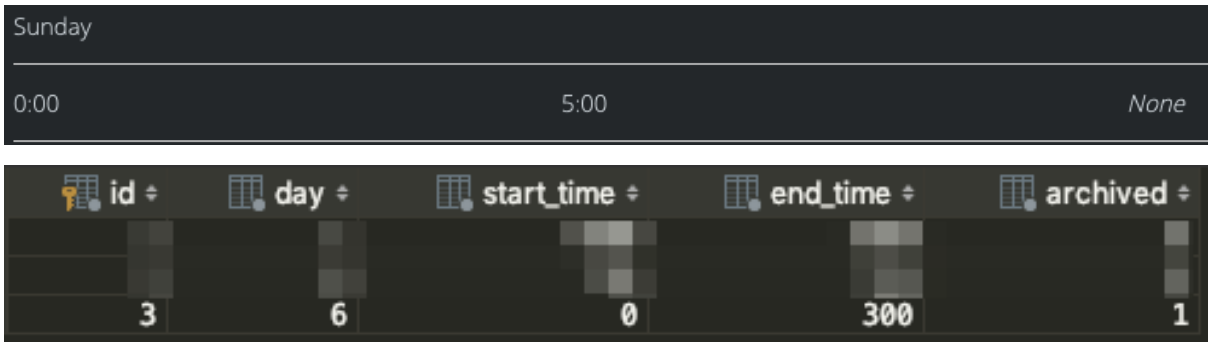
Further, to ensure there was no way to get around this UI only block, a check was also added to the edit route directly to ensure that a student would not be able to access the unavailability for an assigned session.

```
# If student assigned
if user.id in [f.user.id for f in session.assignments]:
    return error_render("Currently assigned to session",
                        "You cannot update your unavailability on a session you are currently assigned to."
                        "\nPlease seek help from a staff user to un-assign you from the session")
```

3.9. 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.

```
Session.query.filter_by(archived=False)
```

```
or session.archived is True:
```

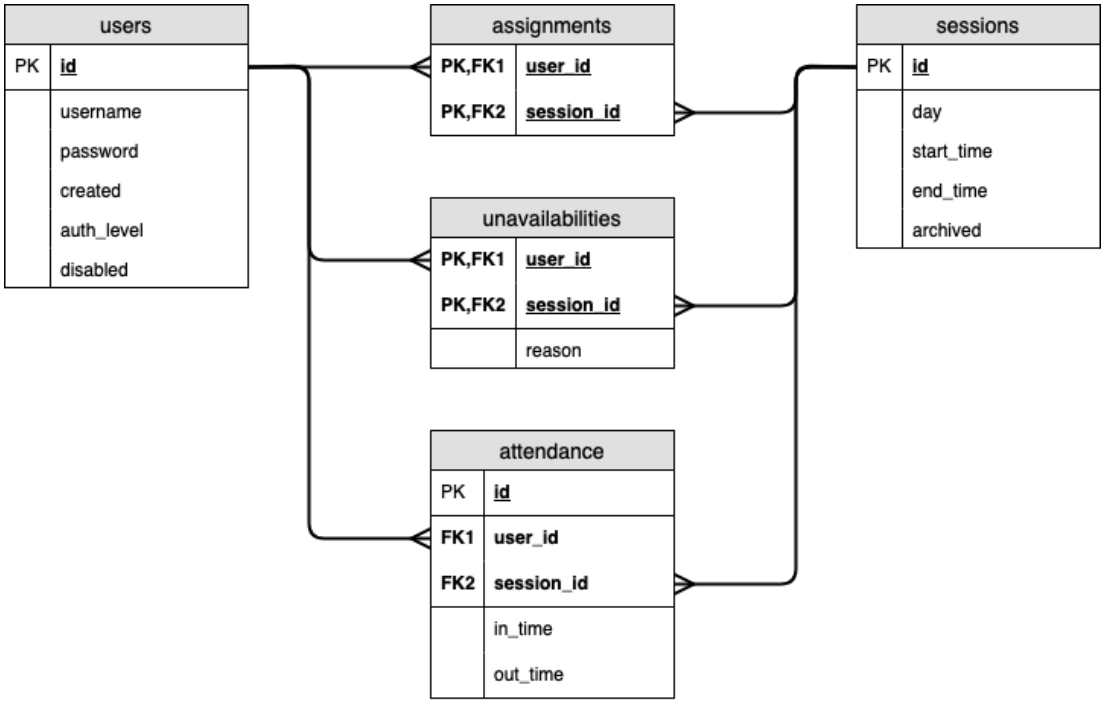
```
if not f.session.archived]
```

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.

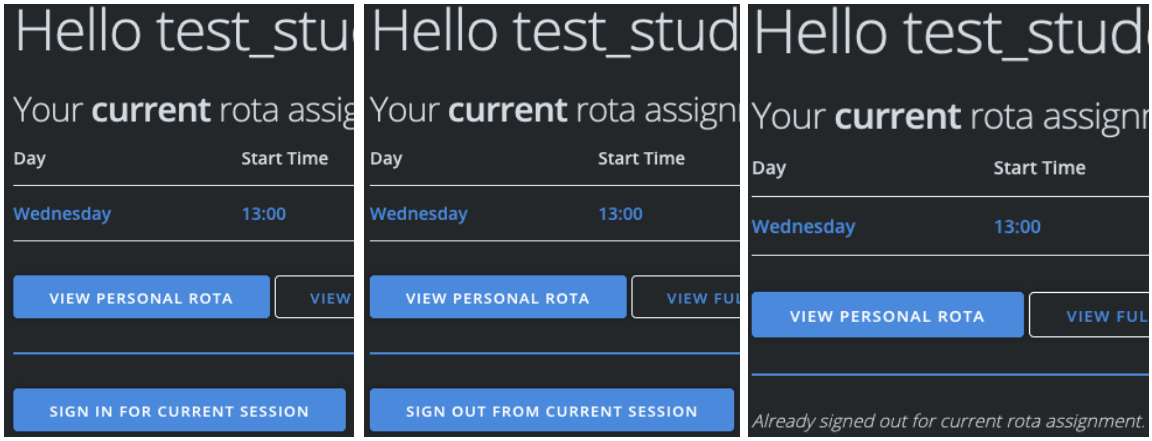
3.10. Beginning attendance tracking on the student portal

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.



3.11. 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.

id	user_id	session_id	in_time	out_time
1	1	1	2018-07-25 12:27:36.321509	<null>
2	1	1	2018-07-25 15:48:25.372419	<null>

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.

```
# Get most recent sign in for this session
data = signed_in_out_session(user, next_session)

# If already signed in
if data[1]:
    return error_render("Already signed in for assigned session",
                        "Your user is already signed in for the current assinged session")
```

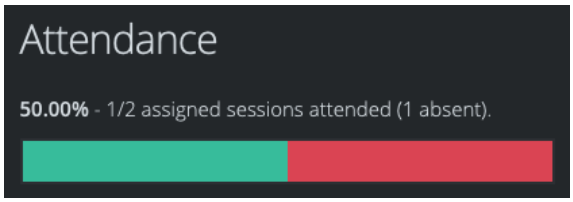
A further issue that I soon realised whilst playing around with the attendance system is that a student could forget to sign out which lead to an infinite attendance when calculated upon. To counter this the time that the assignment would end is stored in the attendance model so a simple or comparison 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.

3.12. 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.



3.13. 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.

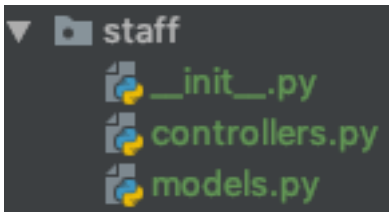
Therefore, if a student forgets their password, staff will be able to reset the password from their management panel. However, when a student is logged in, they may change their password through their own account management page.

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.

```
# Verify new = new confirm
if form.new_password.data == form.new_password_confirm.data:

    # Verify old = current
    if check_password_hash(user.password, form.old_password.data):

        # Update password
        dbsession = db_session()
        user_session = User.query.with_session(dbsession).filter_by(id=user.id).first()
        user_session.password = generate_password_hash(form.new_password.data)
        dbsession.commit()
        flash('Password updated')

    else:
        flash('Old Password is not correct')

else:
    flash('New Password and New Password Confirmation do not match')
```

3.14. 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.

3.15. 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.

3.15.1.Issue with auth level input

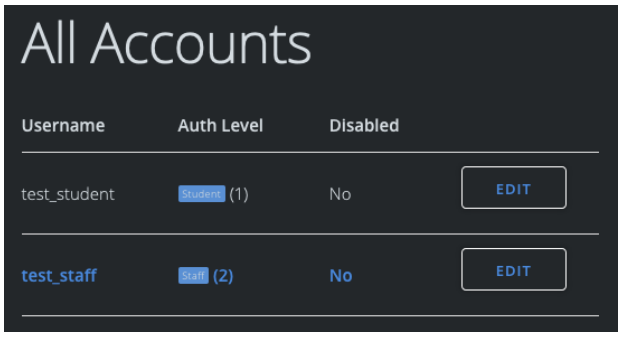
The auth level input selector has a value of either 1 or 2. However, due to how html inputs work, flask was seeing these as strings when selected leading to the form not validating and throwing an error.

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.

```
SelectField('Auth Level', choices=[(1, "Student"), (2, "Staff")], coerce=int)
```

3.16. 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.

3.17. 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.

```
class AccountForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired(message='Please enter a username.')])
    password = PasswordField('Password', validators=[DataRequired(message='Please enter a password.')])
    auth_level = SelectField('Auth Level', choices=[(1, "Student"), (2, "Staff")], coerce=int,
                             validators=[DataRequired(message='Please select an auth level.')])
```

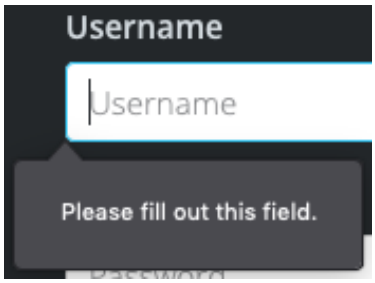
However, standard backend checks will still be made as the validators cannot provide all the checking functionality we need, such as ensuring the username is unique to this account. Whilst the username has no constraint in the database to be unique, having two accounts with the same username would be 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.

```
result = User.query.filter_by(username=form.username.data).first()

# Verify username not already used
if not result:
```

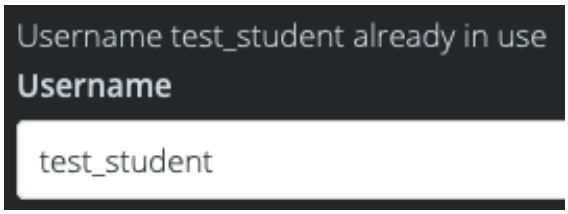
With the backend complete the frontend for the form could be created. Similar to the backend, the frontend could be copied from the edit account template and modified slightly to support the updated form attributes.

3.17.1.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.



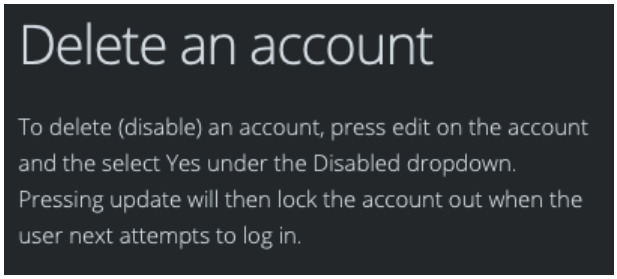
This is all the frontend validation that is present. On the backend further validation is done, such as ensuring the username is unqiue. If a taken username is entered and create is pressed, an error is displayed in a formatted form a expected.

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.

3.18. 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.

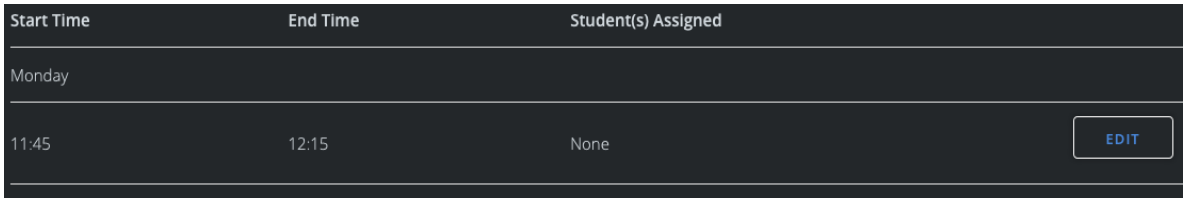
This may be confusing for a new user to the system and so on the main staff account management page a note was added to the frontend design explaining this, below the newly added create new account button in the sidebar next to the main account table.



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.

3.19. 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.

3.20. 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.

```
SelectField('Day of Week', choices=[(f, calendar.day_name[f]) for f in range(len(calendar.day_name))],
```

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

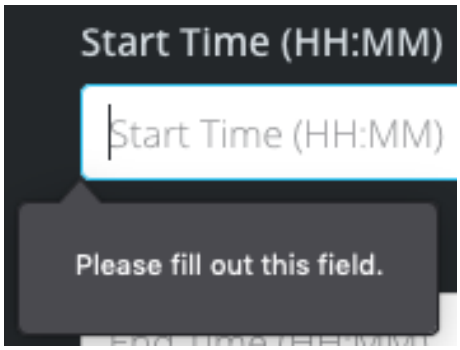
field on browsers that do not support it. WTForms handles parsing the input from this into a standard datetime.time instance.

3.21. 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.

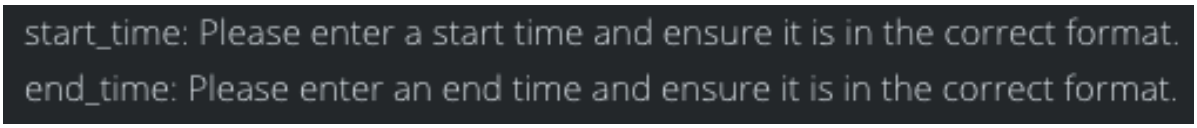
3.21.1.Testing

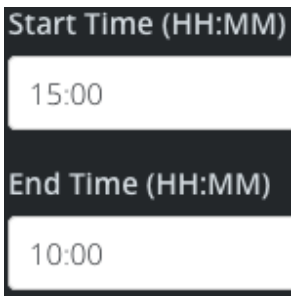
With the method written, testing can be carried out on the edit and create session pages.



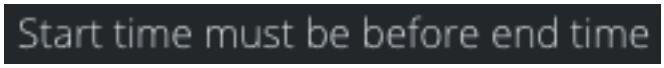
Failing to input a value in the start time or end time results in a popup showing the field is required before submission. This is a frontend check based on the required attribute in the input fields. In addition to this there is also a backend check that also ensures a value is present.

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.



3.22. Updating assignments

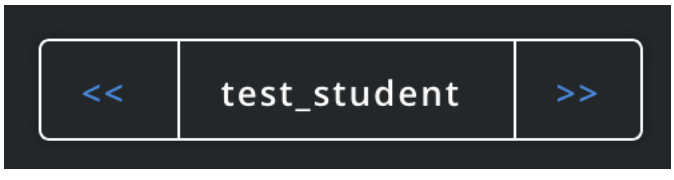
The ability for staff to update assignments on sessions in the rota is a key feature of the system. Looking at all the HTML5 input options available to me, nothing struck me as an easy to use input option for selecting multiple people from one list to have them in another list. Based on this, I decided 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.

3.22.1.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,




```
<div class="button group">
    <a>&lt;&lt;</a>
    <span>test_student</span>
    <a>&gt;&gt;</a>
</div>
```

both will be displayed.

3.22.2.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.

```
{% block extra_js %}
<script>
    function create_button(name, id, assigned) {
        // Create outer div
        var div = document.createElement("div");
        div.className = "button group";

        // Create span with name in it
        var span = document.createElement("span");
        span.innerText = name;
        div.appendChild(span);

        // Create a element with id
        var a = document.createElement("a");
        a.setAttribute("data-id", id);

        // Set a text and position in div
        if(assigned) {
            a.innerText = ">>";
            div.appendChild(a);
        } else {
            a.innerText = "<<";
            div.insertBefore(a, div.childNodes[0]);
        }

        // Done
        return div;
    }

    console.log(create_button("test_student", 1, true).outerHTML);
</script>
{% endblock %}
```

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.

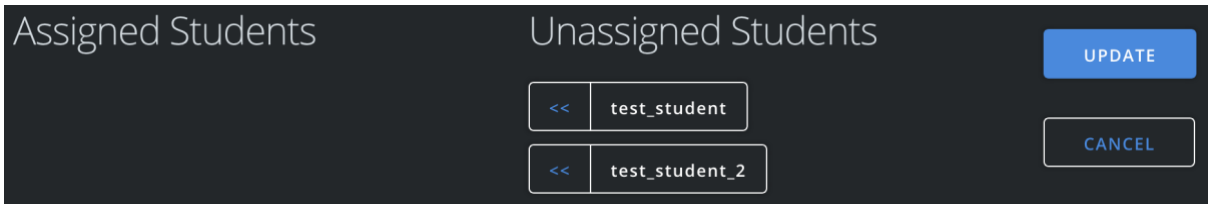
With this done, the on load JavaScript can be setup that will generate all the initial buttons based on the data passed by the template.

This requires a slight modification to the script created which

```
<div class="button group"><span>test_student</span><a data-id="1">&gt;&gt;</a></div>
```

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.



3.22.3.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.

```
// Click handler
function click_handle(element) {
    // Get data
    var id = parseInt(element.getAttribute("data-id")); // force int
    var name = element.getAttribute("data-name");
    var assigned = !!parseInt(element.getAttribute("data-assigned")); // force int then bool

    // Remove old and create new
    element.parentElement.remove();
    create_button(name, id, !assigned);
}
```

```
a.setAttribute("data-id", id);
a.setAttribute("data-name", name);
a.setAttribute("data-assigned", (assigned?"1":"0"));
a.onclick = function(e) {
    e.preventDefault();
    click_handle(a);
};
```

3.22.4.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.

```
// Get existing JSON value
var input = document.getElementById("{{ form.assigned.id }}");
var value = JSON.parse(input.value);

// Update
if(assigned) {
    value.splice(value.indexOf(id),1);
} else {
    value.push(id);
}

// Set new JSON value
input.value = JSON.stringify(value);
```

3.22.5.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.

```
dbsession = db_session()

for remove in to_remove:
    assignment = Assignment.query.with_session(dbsession).filter_by(user_id=remove, session_id=session.id,
                                                                    removed=None).first()
    assignment.removed = datetime.now()
    dbsession.commit()

for add in to_add:
    assignment = Assignment(add, session.id)
    dbsession.add(assignment)
    dbsession.commit()
```

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.

3.22.6.Showing unavailabilities

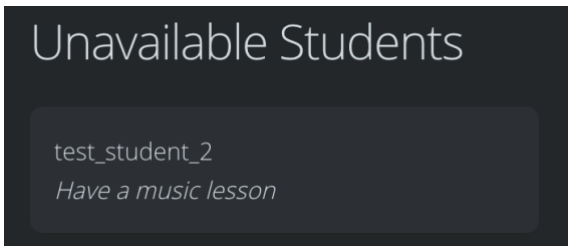
A key part of the system is that students can set themselves as unavailable for some sessions on the rota. This should show up for staff when they are assigning students so that hopefully the unavailable students won't be assigned to that session.

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.

```
    # Fetch unavailabilities
    unavailable = Unavailability.query.filter_by(session=session).all()
# Render
return render_template("staff/assignment_edit.jinja2", form=form,
                       assigned=assigned, unassigned=unassigned,
                       unavailable=unavailable)
```

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.

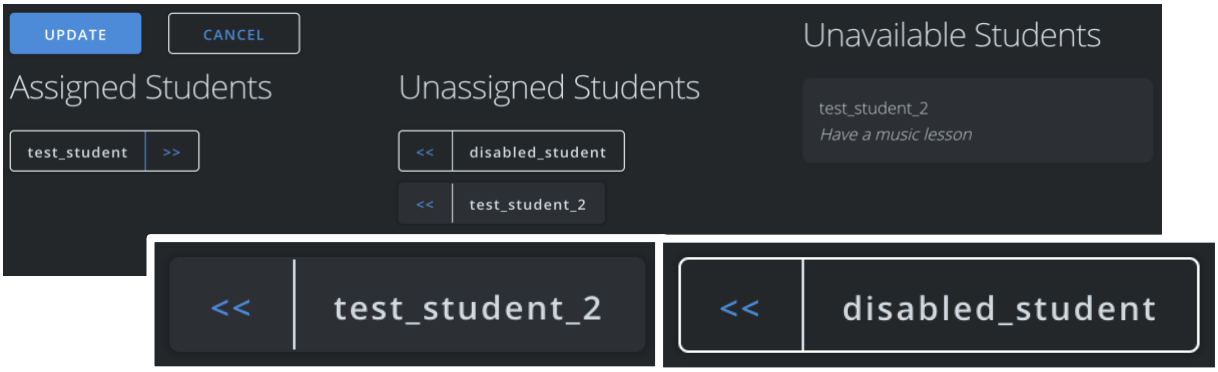
The relied on a simple for loop in the template to render each unavailability passed through, with a fall back if there are no student unavailable.



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.

3.23. 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.

```scss
$margin-value: (0, .5rem, 1rem, 1.5rem, 2rem);

@each $g in $margin-value {
  $i: index($margin-value, $g);

  // Bottom
  .mb-#{$i - 1} {
    margin-bottom: $g;
  }
  // Top
  .mt-#{$i - 1} {
    margin-top: $g;
  }

  // Bottom/Top
  .mbt-#{$i - 1} {
    margin-top: $g;
    margin-bottom: $g;
  }
  .mtb-#{$i - 1} {
    margin-top: $g;
    margin-bottom: $g;
  }

  // Left
  .ml-#{$i - 1} {
    margin-left: $g;
  }
  // Right
  .mr-#{$i - 1} {
    margin-right: $g;
  }
}
```

```scss
// Margins: Bottom
.mb-0 { margin-bottom: 0; }
.mb-1 { margin-bottom: .5rem; }
.mb-2 { margin-bottom: 1rem; }
.mb-3 { margin-bottom: 1.5rem; }
.mb-4 { margin-bottom: 2rem; }

// Margins: Top
.mt-0 { margin-top: 0; }
.mt-1 { margin-top: .5rem; }
.mt-2 { margin-top: 1rem; }
.mt-3 { margin-top: 1.5rem; }
.mt-4 { margin-top: 2rem; }

// Margins: Left
.ml-0 { margin-left: 0; }
.ml-1 { margin-left: .5rem; }
.ml-2 { margin-left: 1rem; }
.ml-3 { margin-left: 1.5rem; }
.ml-4 { margin-left: 2rem; }

// Margins: Right
.mr-0 { margin-right: 0; }
.mr-1 { margin-right: .5rem; }
.mr-2 { margin-right: 1rem; }
.mr-3 { margin-right: 1.5rem; }
.mr-4 { margin-right: 2rem; }

// Margins: Bottom/Top
.mbt-0 { @extend #{'.mb-0', '.mt-0'}; }
.mbt-1 { @extend #{'.mb-1', '.mt-1'}; }
.mbt-2 { @extend #{'.mb-2', '.mt-2'}; }
.mbt-3 { @extend #{'.mb-3', '.mt-3'}; }
.mbt-4 { @extend #{'.mb-4', '.mt-4'}; }
```

3.24. 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.

3.24.1.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.

```python
# Fetch the next user in the rotation that is assignable
def next_assignable(users: List[User], session: Session,
                    force_next: bool = False) \
        -> Tuple[List[User],Union[None, User]]:

    # Loop over users, check if unavailable, select if not
    next_user = None
    for user in users:
        unavailable = Unavailability.query.filter_by(
            session=session, user=user).all()
        if unavailable: continue
        next_user = user
        break

    # If all unavailable and want to force, select first
    if force_next and next_user is None:
        next_user = users[0]

    # If have user, move them to end
    if next_user is not None:
        users.remove(next_user)
        users.append(next_user)

    # Done
    return users, next_user
```

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.

3.24.2.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.

```python
# Generate assignments
def generate_assignments(users_per_session: int = 1, force_user: bool = True):
```

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.

```
# Fetch all sessions
sessions = Session.query.filter_by(archived=False).all()
if not sessions: return
# Fetch all users
users = User.query.filter_by(auth_level=1, disabled=False).all()
if not users: return
```

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

```
# Remove old assignments
assignments = Assignment.query.with_session(dbsession).filter_by(removed=None).all()
for assignment in assignments:
    assignment.removed = datetime.now()
dbsession.commit()
```

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.

```
# Loop over sessions
for session in sessions:
    # Correct number of users per session
    target = users_per_session
    assigned = []
    while target > 0:

        # Get next user to assign
        users, student = next_assignable(users, session, force_user)

        # Give up if no users
        if not student and not assigned:
            break

        # If valid student
        if student:

            # Give up if we're out of users (last assigned is this user)
            if assigned and student == assigned[-1]:
                break
            # Skip if already assigned
            if student in assigned:
                continue
```

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

3.24.3.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.

```
# Rota edit - automatic assignments
@staff.route('/rota/auto', methods=['GET', 'POST'])
def rota_automatic_assignments():
    user, error = auth_check()
    if error:
        return error

    generate_assignments(3, False)

    return
```

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.

sqlalchemy.exc.InterfaceError

```
sqlalchemy.exc.InterfaceError: (sqlite3.InterfaceError) Error binding
parameter 0 - probably unsupported type. [SQL: 'INSERT INTO assignments
(user_id, session_id, created, removed) VALUES (?, ?, ?, ?)'] [parameters:
(<User 'test_student'>, 1, '2018-10-01 17:47:13.494745', None)]
(Background on this error at: http://sqlalche.me/e/rvf5)
```

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.

```
assignment = Assignment(student.id, session.id)
```

Start Time	End Time	Student(s) Assigned
Monday		
11:45	12:15	test_student
Tuesday		
11:45	12:15	test_student_2, test_student
Thursday		
12:00	21:00	test_student, test_student_2

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

	user_id	session_id	created	removed
1	1	2	2018-08-08 17:27:41.849615	2018-08-08 17:27:46.531239
2	1	2	2018-08-08 17:28:28.448173	2018-10-01 17:47:13.485715
3	1	1	2018-10-01 17:50:42.744317	<null>
4	5	1	2018-10-01 17:50:42.752201	<null>
5	1	2	2018-10-01 17:50:42.761121	<null>
6	5	4	2018-10-01 17:50:42.766749	<null>
7	1	4	2018-10-01 17:50:42.770599	<null>

displayed in the front-end and the previous two assignments that are now remvoed.

3.24.4.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.

```
# Automatic Assignments
class AutomaticAssignmentForm(FlaskForm):
    count = IntegerField('Students per Session', validators=[DataRequired(
        message='Please enter a number of students to assign per session')])
    force = BooleanField('Force students to be assigned')
```

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

```
# Check sessions
sessions = Session.query.filter_by(archived=False).all()
if not sessions:
    return error_render(503, "No sessions are currently in the rota. Please define a session before automatically "
                             "assigning students to the rota.")

# Check students
users = User.query.filter_by(auth_level=1, disabled=False).all()
if not users:
    return error_render(503, "No students are in the system. Please create a user (not disabled) before attempting "
                             "to assign them to rota sessions.")
```

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.

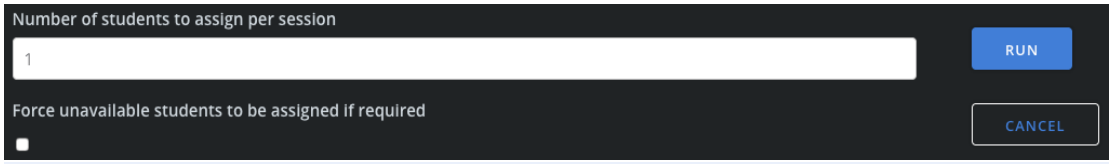
```
# Get form and set checks
form = AutomaticAssignmentForm(request.form)
form.count.min = 1
form.count.max = len(users)
```

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.



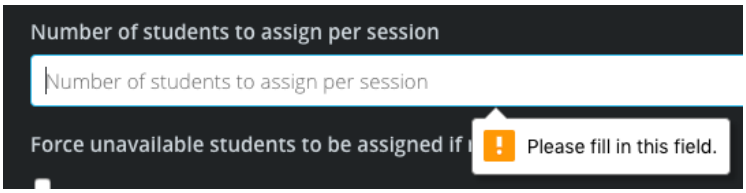

```
<input autofocus class="u-full-width " id="count" max="2" min="1" name="count" placeholder="Number of students to assign per session" required type="number" value="1">
```

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.

```
# Errors
if form.errors:
    for field, error in form.errors.items():
        flash('{}: {}'.format(field, ", ".join(error)))
```

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.

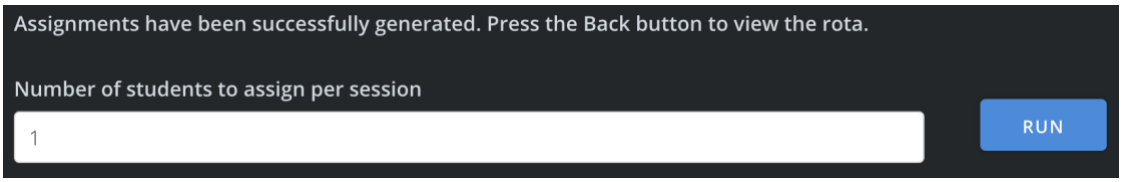
With this all written it is time to test that the full form works as expected, including the validation. First I attempted to submit the form with no count inputted. Upon pressing run I was presented with the standard HTML5 validation requirement error for Chrome.



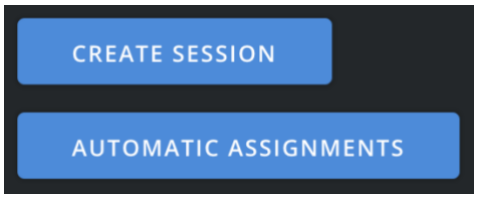
Entering a negative value returns a similar Chrome error, instructing the user that the input must be one or more. Similarly, entering a value that is above the maximum set will also produce a Chrome error.

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.



3.24.5.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

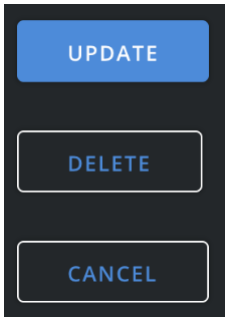
known set of students as the resources available. This is what made this system perfect for being handled through computational means.

3.24.5.1. Application in the rota system

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.

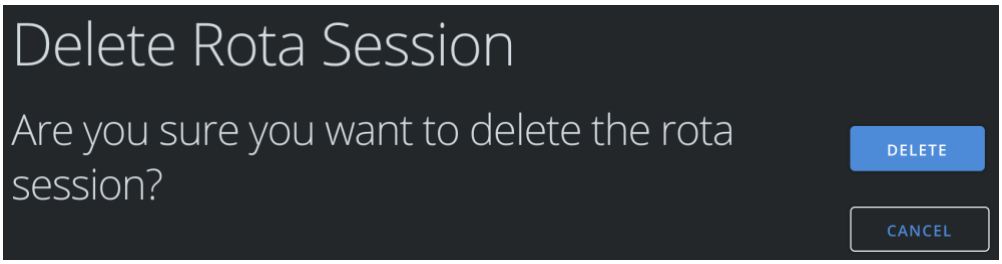
Using the Hungarian algorithm as the basis allowed me to ensure that the function for generating the assignments was as efficient as it could be whilst meeting the requirements set out in the design of this system, ensuring students were allocated fairly and only if they were available if stipulated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

3.25. 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



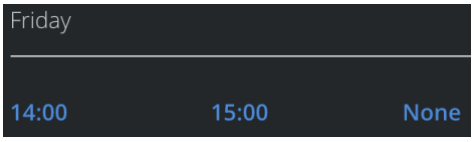
simply post to the route when the user presses the button in the confirmation page.

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.

```
# Get session
dbsession = db_session()
session = Session.query.with_session(dbsession).filter_by(id=id).first()

# Remove old assignments
assignments = Assignment.query.with_session(dbsession).filter_by(session_id=id, removed=None).all()
for assignment in assignments:
    assignment.removed = datetime.now()
dbsession.commit()

# Remove session
session.archived = 1
```

3.25.1.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.

session_id=5

	user_id	session_id	created	removed
1	1	5	2018-10-11 10:06:26.716028	2018-10-11 10:08:06.086128

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.

3.26. 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.

3.26.1.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

3.26.2.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.

3.27. 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.

```
# Loop over sessions
total_assigned = {f.id: [f, 0] for f in users}
for session in sessions:
    # Correct number of users per session
    target = users_per_session
    assigned = []
```

```
# The user that hopefully will be found
next_user = None

# Order users by assignments so far (smallest first)
users_sorted = sorted(total_assigned.values(), key=lambda x: x[0])

# Loop over users until we find one (least assignments first)
for user in users_sorted:
    # Check if already assigned
    if user in assigned:
        continue

    # Check if available
    unavailable = Unavailability.query.filter_by(
        session=session, user=user).all()
    if unavailable:
        continue

    # Store
    next_user = users_sorted

# If all unavailable and want to force, select first
if force_next and next_user is None:
    users_sorted = [f for f in users_sorted if f[1] not in assigned]
    if users_sorted:
        next_user = users_sorted[0]
```

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.

```
# Get next user to assign (will also update both assigned)
student = next_assignable(total_assigned, assigned, session, force_user)

# Give up if no users
if not student:
    break

# Update target
target -= 1
# Assign
assignment = Assignment(student.id, session.id)
dbsession.add(assignment)
dbsession.commit()
```

3.27.1.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**.

3.27.1.1. 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

3.27.1.2. 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.

```
try:
    generate_assignments(form.count.data, force)
except:
    traceback.print_exc()
```

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.

```
# Get the user object
user_obj = user[1]  # [int (assignments), user]
```

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.

3.27.1.3. 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.

Start Time	End Time	Student(s) Assigned
Monday		
11:45	12:15	test_student_3, test_student
Tuesday		
11:45	12:15	test_student_3, test_student_2
Wednesday		
12:00	21:00	test_student_3, test_student_2

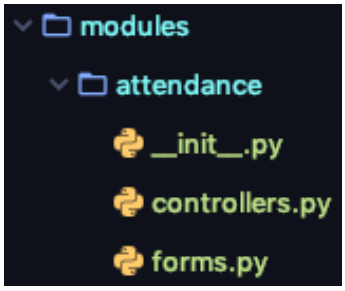
After some digging and debugging, the issue appeared to be that the algorithm simply was not selecting the correct user, even if all the checks passed. Then I noticed the issue, a missing break at the end of the for loop to select the user. This meant that every time the loop was continuing on even if 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.

```
# Store
next_user = user
break
```

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.

3.28. 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.

3.28.1.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.

```
def auth_check(student_page_id: int = None):
    # If not staff
    if not error and user.auth_level != 2:
        # Allow student to view own page
        if not student_page_id or user.id != student_page_id:
            error = error_render("Staff access only",
```

3.28.2.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.

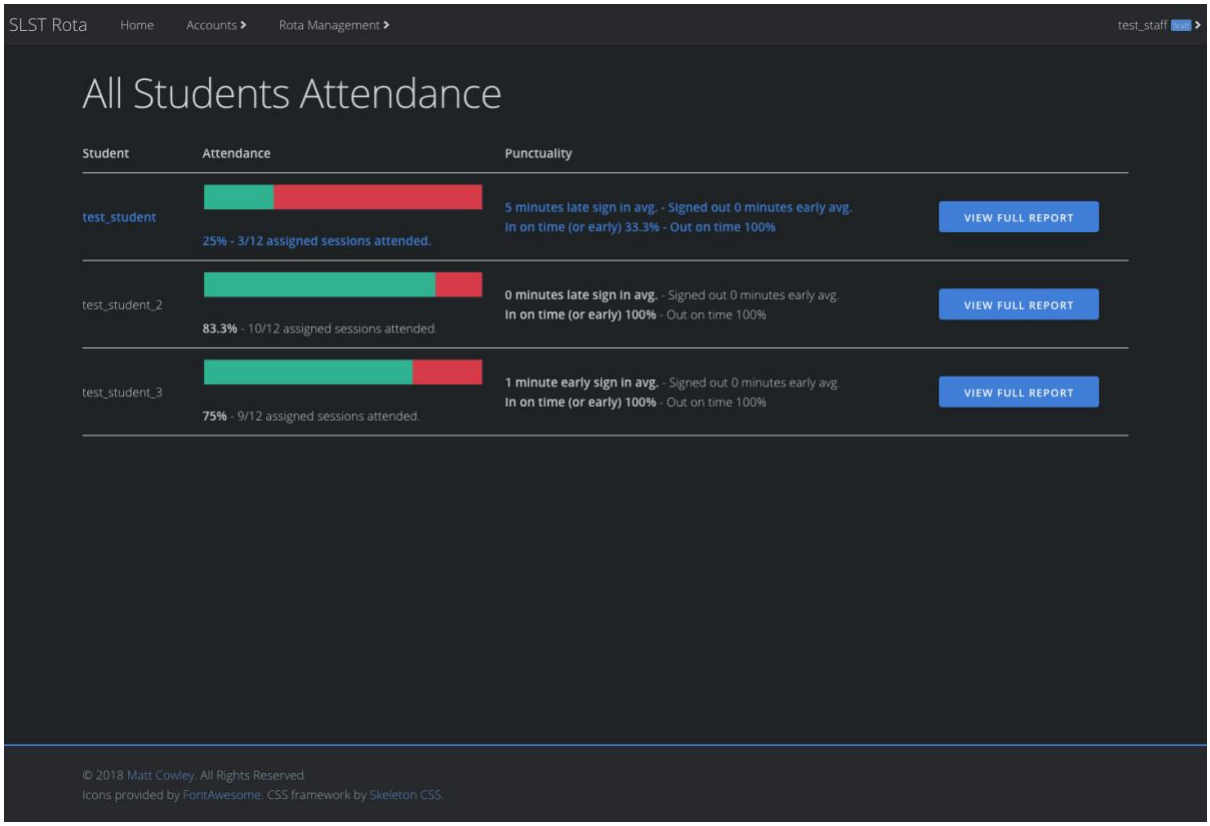
This home page will provide an attendance bar similar to the student page, but for each student in the system within a table view. It will also provide the overall punctuality stat from the student overview page on the table. To make this easier, instead of duplicating code from the student overview, 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.

3.28.2.1. 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.



3.28.2.2. 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.

```
# Attendance bar
self.__attendance_bar = "<div class=\"stat-bar\">" \
                        "<span class=\"success\" style=\"width: {}%;\"></span>" \
                        "<span class=\"danger\" style=\"width: {}%;\"></span>" \
                        "</div>".format(self.__attendance_present,
                                        self.__attendance_absent if self.__attendance_total else 0)
```

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

```
<h4>Attendance</h4>
<p class="mb-2">{{ report.attendance }}</p>
<div class="stat-bar mb-2">
    <span class="success" style="width: {{ report.present }}%;"></span>
    <span class="danger" style="width: {{ report.absent }}%;"></span>
</div>
```

After

```
<h4>Attendance</h4>
<p class="mb-2">{{ report.attendance }}</p>
<div class="mb-2">{{ report.attendance_bar }}</div>
```

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.

```
report = StudentReport(this_student.id)
table.append([
    (report.present < 75),
```

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.

```
table.append([
    (report.present < 75),
    [this_student.username,
     report.attendance_bar + "\n" + report.attendance,
     report.punctuality,
```

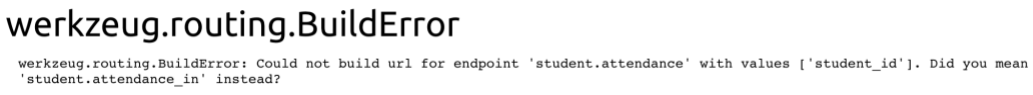
The last item is the button to access the full student report page. This has to have the html written for it in the controller to be inserted into the table. To generate the correct link, the built-in Flask function ‘url_for’ can be used. This takes a string argument to lookup the correct route and then 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.

```
"<a href=\"{}\" class=\"button primary\">View full report</a>".format(
    url_for("student.attendance", student_id=this_student.id))
```

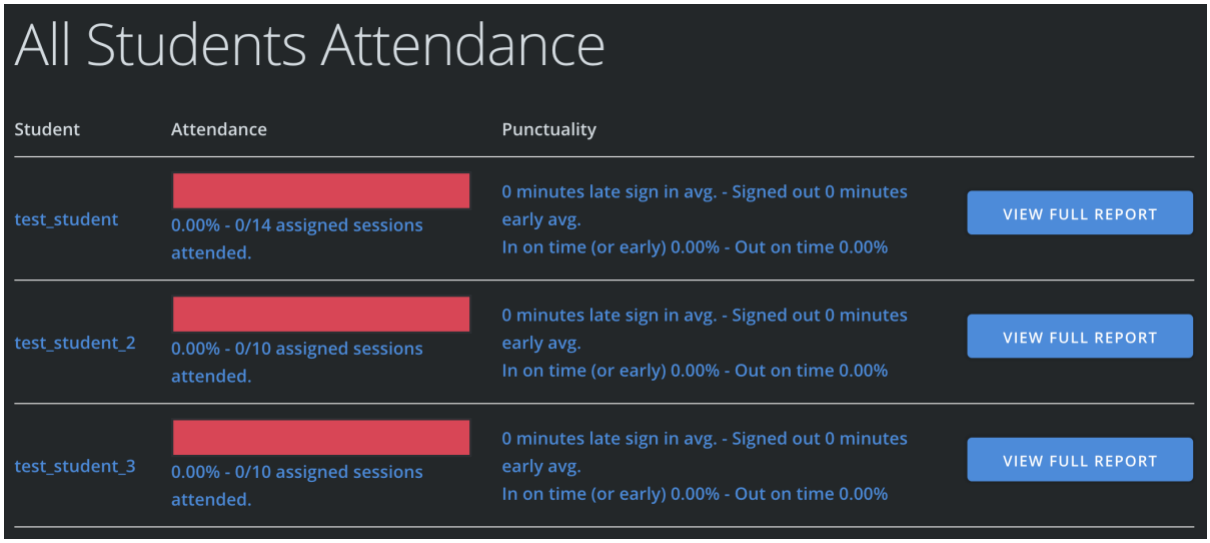
3.28.2.3. Testing

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.



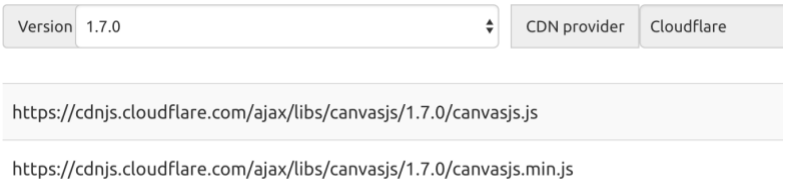
Upon reviewing the line that was the source of the error, the issue was apparent. In the url_for call I had specified the route “student.attendance” when in should actually be “attendance.student” as I had recently moved the student attendance report to the new attendance controller so it was no longer in the student controller. With this error fixed, the route worked correctly once the web server was rebooted, rendering the content as expected.



3.28.3.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.

3.28.3.1. Development

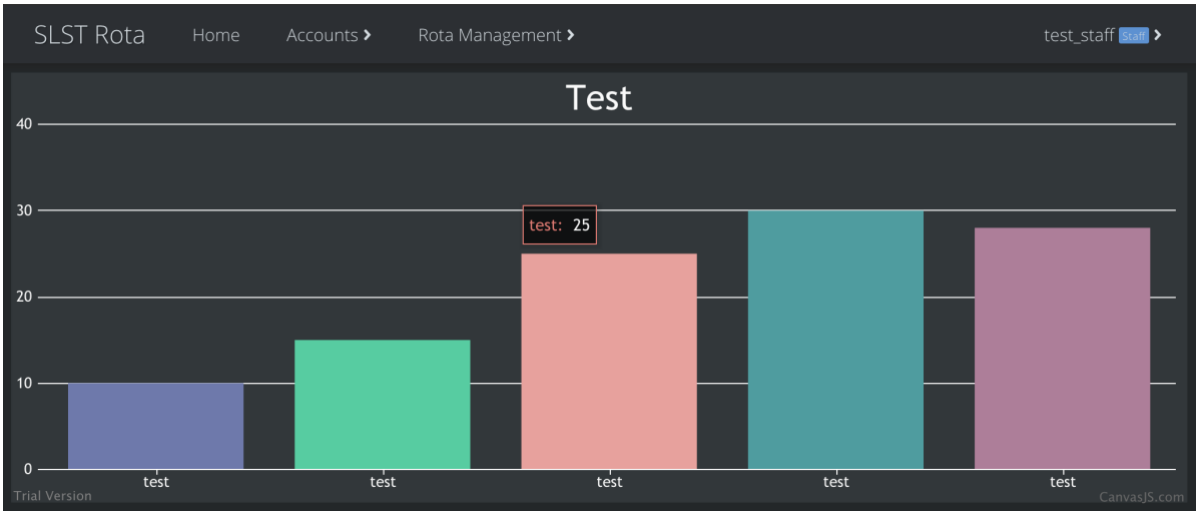
To begin the development process for a graph, I decided to tackle the front end first. I imported the canvasjs script and setup a basic test graph to render. This is done within the window.onload JavaScript function to ensure it is only run once the script file has been loaded in.

```
window.onload = function () {

    const chart = new CanvasJS.Chart("session_attendance_chart", {
        theme: "dark2",
        animationEnabled: true,
        title: {
            text: "Test"
        },
        data: [
            {
                type: "column",
                dataPoints: [
                    {label: "test", y: 10},
                    {label: "test", y: 15},
                    {label: "test", y: 25},
                    {label: "test", y: 30},
                    {label: "test", y: 28}
                ]
            }
        ]
    });
    chart.render();

}
```

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.

```
<div class="row">
    <div class="twelve columns">
        <div id="session_attendance_chart" style="height: 370px; width: 100%;"></div>
    </div>
</div>
```

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.

```
{% macro render_chart(id, title, type, data_points) %}
    const {{ id }}_CanvasJSChart = new CanvasJS.Chart("{{ id }}", {
        theme: "dark2",
        animationEnabled: true,
        title: {
            text: "{{ title }}"
        },
        data: [
            {
                type: "{{ type }}",
                dataPoints: {{ json.dumps(data_points)|safe }}
            }
        ]
    });
    {{ id }}_CanvasJSChart.render();
{% endmacro %}
```

3.28.3.2. 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.

```
window.onload = function () {
    {% from "app/macros/chart.jinja2" import render_chart with context %}
    {{ render_chart("session_attendance_chart", "Test", "column", [{"label": "test", "y": 10}]) }}
}
```

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.

jinja2.exceptions.UndefinedError

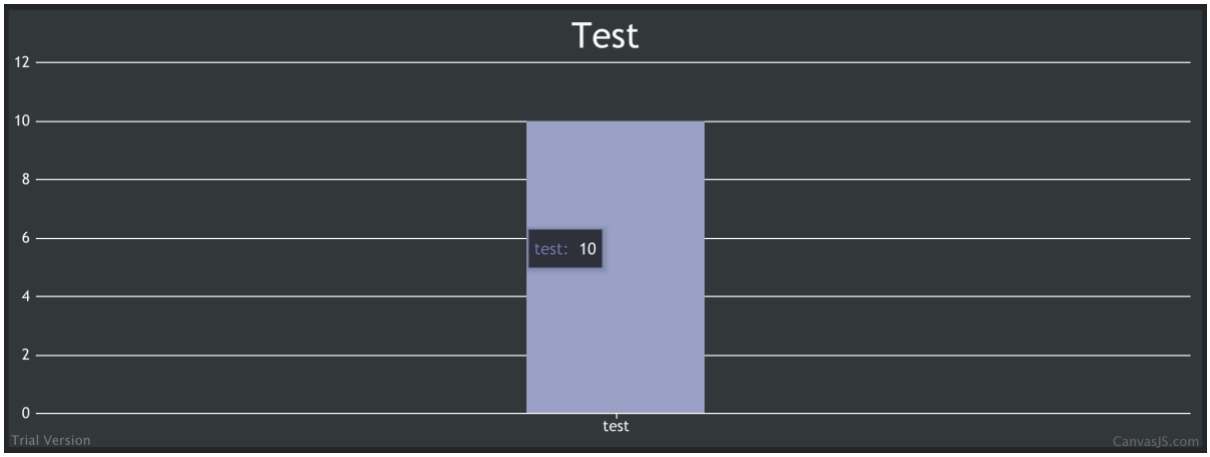
```
jinja2.exceptions.UndefinedError: 'json' is undefined
```

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.

```
@app.context_processor
def variables() -> dict:
    a = [
        datetime,
        auth,
        json
    ]  # Convince pycharm things are used
    return dict(**globals())
```

```
__init__.py
1  import json
```

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



3.28.3.3. 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.

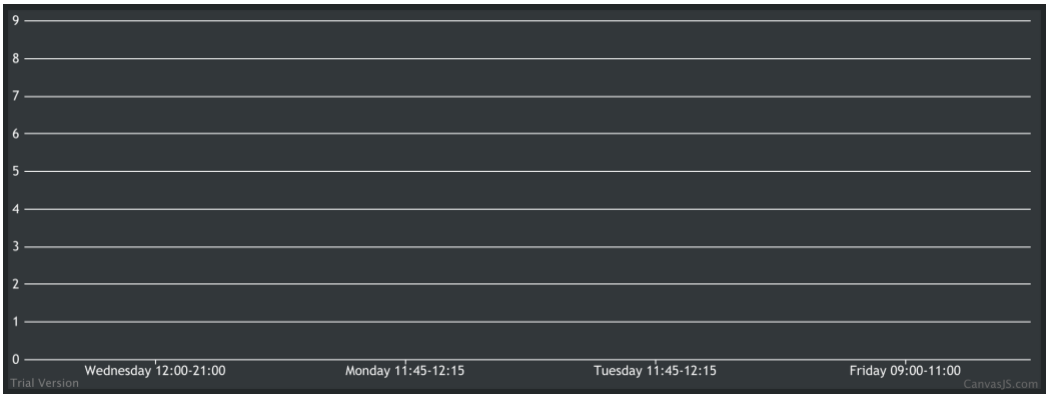
```
# Compile session data
for assignment in report.breakdown:
    # Get raw assignment
    assignment_raw = assignment.assignment
    # Added to session_data if not there
    if assignment_raw.session.id not in session_data:
        session_data[assignment_raw.session.id] = [assignment_raw.session, 0, 0]  # session, present, total
    # Update with present and total from this assignment
    session_data[assignment_raw.session.id][1] += assignment.present
    session_data[assignment_raw.session.id][2] += assignment.total
```

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.

```
# Generate chart
chart = []
for data in session_data.values():
    chart.append({"label": "{0.day_frmt} {0.start_time_frmt}-{0.end_time_frmt}".format(data[0]),
                  "y": data[1]/data[2]*100})
```

3.28.3.4. 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

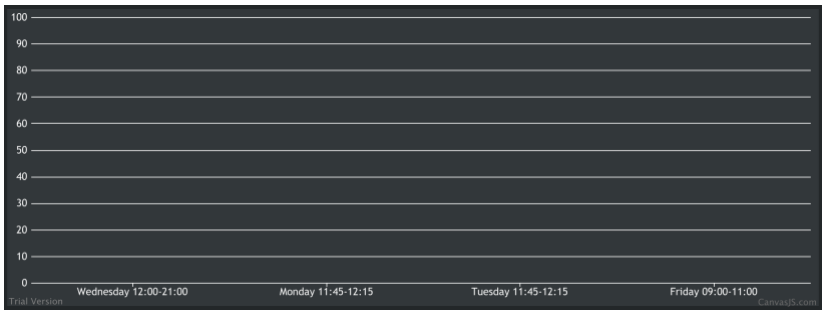
how to solve this, I referred to the CanvasJS documentation (https://canvasjs.com/docs/) in hope that they provided a way to fix an axis. Under the axisY section of the chart documentation there is a minimum and maximum value that can be specified to provide a fixed axis. I now need to implement this 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.

```
{% for key, val in extra_data.items() %}
    {{ key }}: {{ json.dumps(val)|safe }},
{% endfor %}
```

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

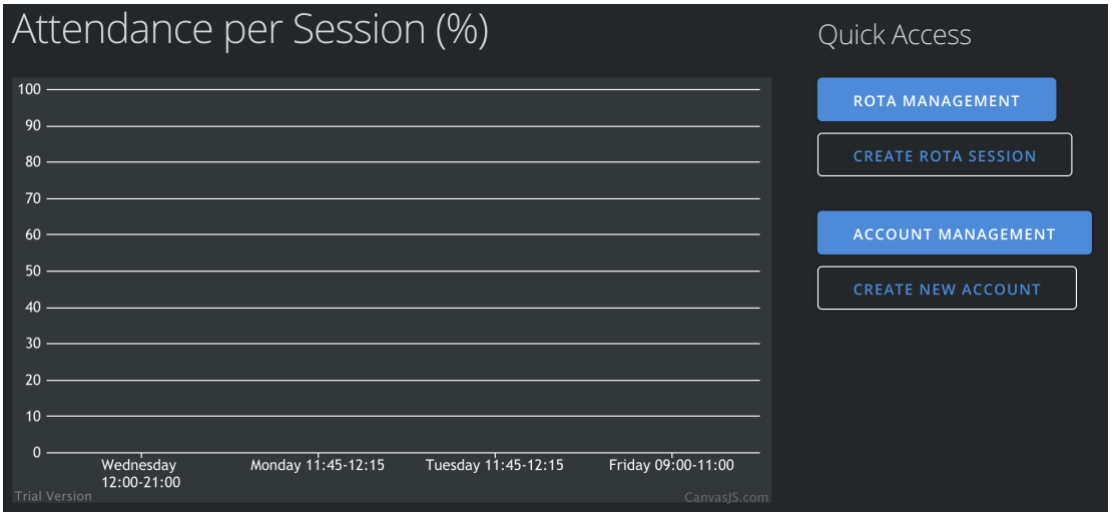
```
"axisY":{"minimum":0,"maximum":100}
```



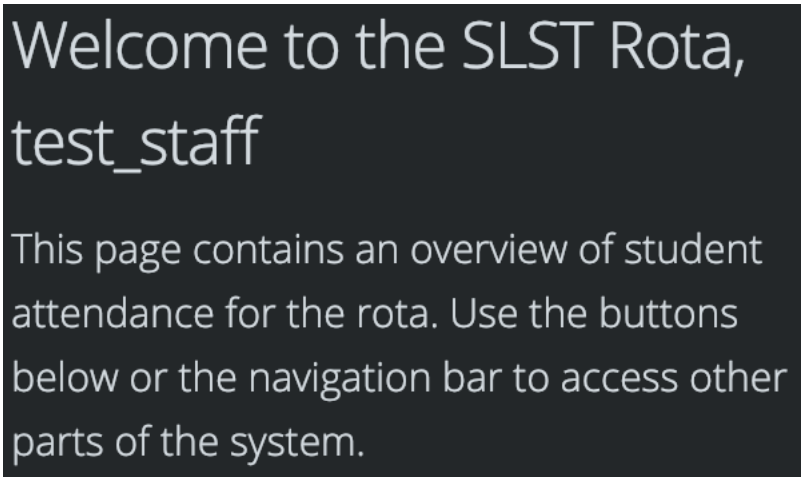
When viewing the page now the chart is rendered as I wanted it to with the fixed axis of 0% to 100% and the sessions at the bottom.

3.28.3.5. 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.



These two changes made the attendance home page suddenly feel much more like a useable and welcoming page for the staff member to use who was currently signed in. In addition to the buttons, I decided to also add a welcome message for the staff member to make the page more inviting and to fill the dead space currently present on the left. To get the username of the current authenticated staff member I can use the global auth module and the 'current_user' helper that I have created within it.




```
<h5 class="mt-4 mb-0">Welcome to the {{ app.name }}, {{ auth.current_user().username }}</h5>

<p class="mt-2 mb-0">
    This page contains an overview of student attendance for the rota.
    Use the buttons below or the navigation bar to access other parts of the system.
</p>
```

The page was now much more home-y but was currently inaccessible to a normal user as it was not linked from any other page on the site. I had originally considered creating a separate link for the page in the navigation bar but instead opted to make

this the full homepage of the staff portal and so made the home link in the navigation bar redirect to the attendance page if it was a staff account that was logged in.

```
# Staff index
if user and user.auth_level == 2:
    return redirect(url_for("attendance.home"))
```

3.29. Student Attendance Graphs

With the main attendance graph created on the staff attendance overview page, I decided I would also implement more graphs on the student report page to give that page more flair for both the student and staff when viewing it.

The graph I wish to implement is the in difference and out difference per assignment for the student. The in difference and out difference is how early or late the student was on average for signing in and out compared to the defined start and end time for the session in the rota. Showing this data in a graph on the student report page should allow the staff and student to more easily spot problematic assignments where the student may be more late than others.

3.29.1.Data generation

To begin the graph, the data for it needs to be generated which will consist of two datasets unlike the previous graph. These two data sets will both be scatter sets, one for the in diff of each assignment and the other for the out diff of each assignment.

```
# Generate graph data
graph = [{
    "type": "scatter",
    "legendText": "Sign in difference average",
    "dataPoints": []
}, {
    "type": "scatter",
    "legendText": "Sign out difference average",
    "dataPoints": []
}]
```

Both will have legend text to indicate which set is the in and which is the out difference.

I can then fetch the same student report data that is used elsewhere in this route to iterate over the breakdown attribute which provides an item for each assignment the student has. Using this I can then save the in_diff_avg and out_diff_avg attributes of each assignment report to the graph data sets.

```
for assignment in report.breakdown:
    label = "{0.day_frmt} {0.start_time_frmt}-{0.end_time_frmt}".format(assignment.assignment.session)
    # In time diff
    graph[0]["dataPoints"].append({"x": label, "y": assignment.in_diff_avg})
    # Out time diff
    graph[1]["dataPoints"].append({"x": label, "y": assignment.out_diff_avg})
```

To test that all the data was being generated correctly I simply printed the full collection of datasets to the console when the route was accessed. This will allow me to see the full variable as it will be presented in the template to the graph creation macro.

```
[{'type': 'scatter', 'legendText': 'Sign in difference average', 'dataPoints': [{'x': 'Wednesday 12:00-21:00',
 'y': 0}, {'x': 'Wednesday 12:00-21:00', 'y': 0}, {'x': 'Monday 11:45-12:15', 'y': 0}, {'x': 'Monday 11:45-12:15',
  'y': 0}, {'x': 'Tuesday 11:45-12:15', 'y': 0}, {'x': 'Friday 09:00-11:00', 'y': 0}]}, {'type': 'scatter',
 'legendText': 'Sign out difference average', 'dataPoints': [{'x': 'Wednesday 12:00-21:00', 'y': 0}, {'x':
 'Wednesday 12:00-21:00', 'y': 0}, {'x': 'Monday 11:45-12:15', 'y': 0}, {'x': 'Monday 11:45-12:15', 'y': 0},
 {'x': 'Tuesday 11:45-12:15', 'y': 0}, {'x': 'Friday 09:00-11:00', 'y': 0}]}]
```

All the y values in this dump are 0 as there is currently no attendance data in the system for the assignment report to average out. However, the structure of the data is how I expected it to be and so now I can begin work on creating the front-end graph that will render on the student page based on this data.

3.29.2.Graphing the data

The current graph macro has an input for the datapoints of a single dataset, this will not work for this graph as it has two full datasets in it. To handle this I will need to update the macro and any reference to the macro that already exists. Searching the code reveals, as expected, that there is only one current place where the macro is used. This is the attendance home page which I can update to provide a full dataset instead of just the data points to the macro.

```
{% macro render_chart(id, title, datasets, extra_data = {}) %}
    data: {{ json.dumps(datasets)|safe }},
```

```
return render_template("attendance/home.jinja2", attendance_table=table,
                       session_attendnace={"type": "column", "data": chart},
                       chart_extras={"axisY": {"minimum": 0, "maximum": 100}})
```

```
{{ render_chart("session_attendance_chart", "", session_attendnace, chart_extras) }}
```

3.29.2.1. Testing the updated macro

Upon testing this code by restarting the web server and accessing the attendance homepage, it was not working as expected. The chart had not rendered and so there was just a blank rectangle.

Using inspect element I could check what code had been generated by the macro and the issue became obvious. The data attribute of the chart call was expecting an array of datasets, but I had only provided it with the single dataset not in an array.

```
            data: {"type": "column", "data": [{"label": "Wednesday 12:00-21:00", "y": 0.0}, {"label": "Monday
11:45-12:15", "y": 0.0}, {"label": "Tuesday 11:45-12:15", "y": 0.0}, {"label": "Friday 09:00-11:00", "y":
0.0}]},
```

This was not actually an issue with the macro but an issue with the current call to the macro. The fix was simple, updating the data generation in the python controller to wrap the dataset in a list. With this change made the graph then rendered as expected, although still blank as there was no attendance data in the system.

```
session_attendnace=[{"type": "column", "data": chart}],
```

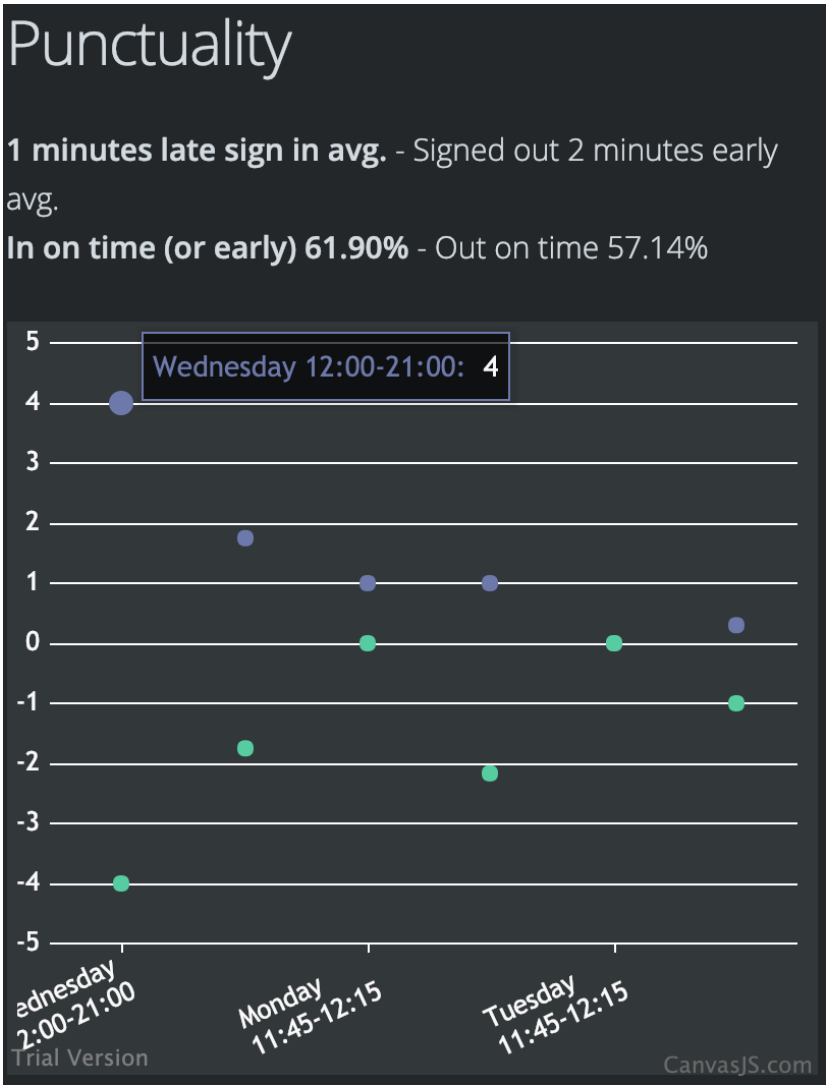
3.30. Creating the new graph

Creating the graph was a very similar process to the previous graph in the attendance home page as it was just a case of passing the generated graph data from the controller through to the macro. Unlike the homepage, this graph had no extra data to pass through and so the macro call just had the chart ID and the datasets.

```
<div id="assignment_diff_chart" class="mt-4" style="height: 370px; width: 100%;"></div>
        <script src="/static/canvasjs-2.2/canvasjs.min.js"></script>
        <script type="text/javascript">
            window.onload = function () {
                {% from "app/macros/chart.jinja2" import render_chart with context %}
                {{ render_chart("assignment_diff_chart", "", graph) }}
            };
        </script>
```

The graph should now render on the student report page when accessed on the rota system website. To ensure that the graph was working correctly I created some fake attendance data for a student that was randomised to ensure the in and out differences would be populated.

The graph rendered well with the correct data displayed. However, there were some issues with the graph concept that I had no considered before.



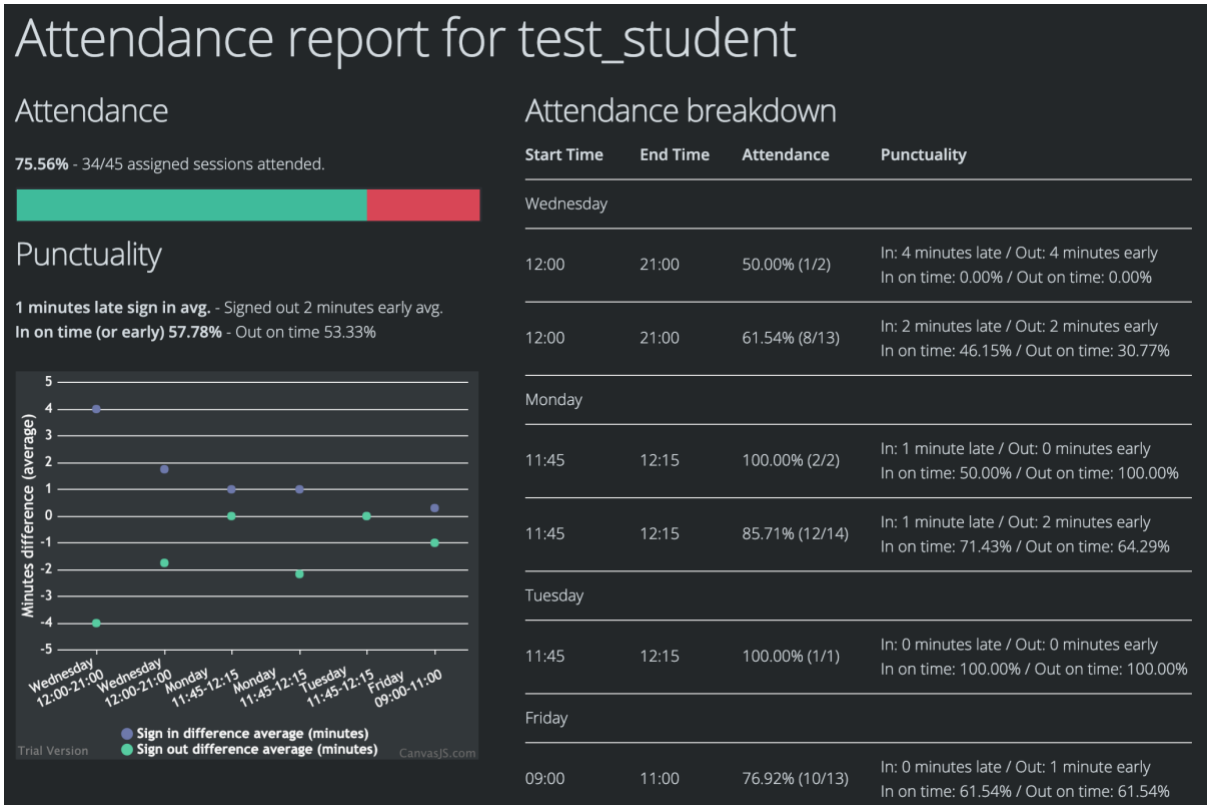
The Y axis was not labeled which could lead to confusion as to what the graph reprenseted and the two items of data per X position were also not labeled with their in or out labels which would easily lead to confusion.

The data set labels were specified in the code but were not being displayed. After reading through the CanvasJS documentation further it became apparent that the key/legend was not shown by default and required the 'showInLegend' attribute being set to true for each data set.

To set a custom title for the Y axis, the 'title' attribute of the 'axisY' object has to be set in the chart definition, the 'titleFontSize' attribute can also be used to adjust the size for longer text. I will pass this into the template under the variable graph_extra which can then be passed into the chart macro.

```
graph_extra={"axisY": {"title": "Minutes difference (average)", "titleFontSize": 15}})
render_chart("assignment_diff_chart", "", graph, graph_extra)
```

With this change made to the graph, it rendered well with clear explanantions of the data it was displaying. It was positioned on the student report in the left hand sidebar underneath the exisitng punctuality overview as it is most related to this element of the page.



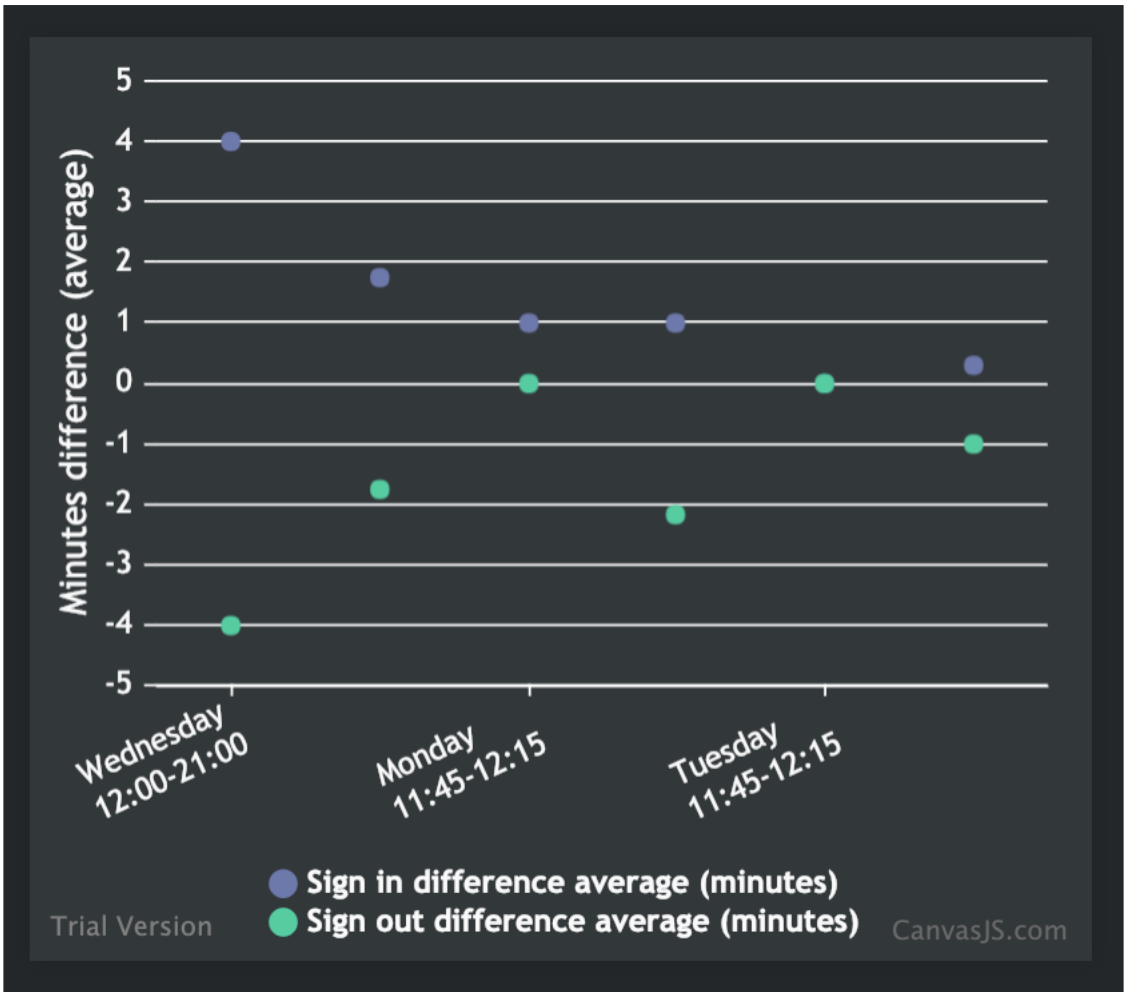
3.30.1.1. Styling the charts

The charts currently have very basic styling which is provided by the library itself, making use of its ‘dark2’ theme. To work on that to provide cleaner styling for the charts, I will make a chart wrapper class that will go around the existing chart divs to style them better and to provide some padding as currently the chart content goes right to the edge of the lighter box.

```
.chart {
  background: #32373a;
  padding: 0.5em;
  box-shadow: 0 0 15px 0 rgba(0, 0, 0, 0.1);
}
```

```
<div class="chart mt-4">
    <div id="assignment_diff_chart" style="height: 370px; width: 100%;"></div>
</div>
```

The wrapper class uses the same background colour as the chart library to ensure a seamless transition whilst providing padding around the chart and a nice subtle box shadow to the element to give it a raised effect on the page. The final product was visually appealing and to my satisfaction.



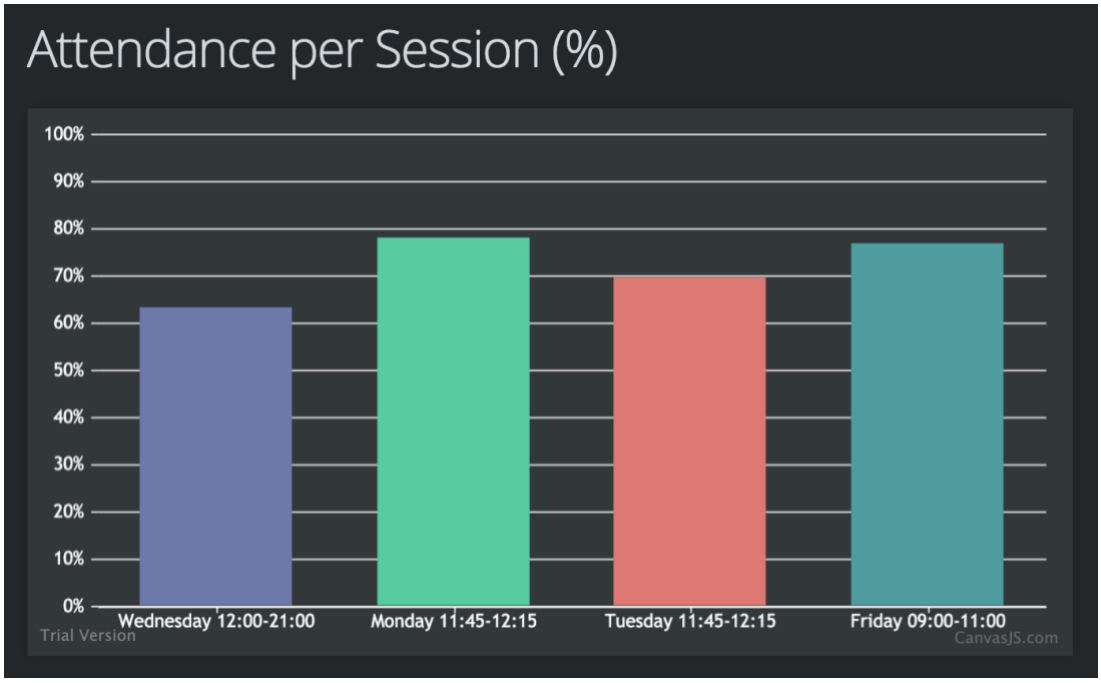
3.30.1.2. Updating the overview graph

With the new styling for the graph applied to the student report, the same could be done to the original graph on the attendance overview page. Additionally, whilst reading the CanvasJS documentation I observed that you can add a prefix or suffix to values on the axis. The homepage graph is done in percentages and so I will pass through some extra graph data to have a suffix of the percent sign on the Y axis values.

```
<div class="chart">
    <div id="session_attendance_chart" style="height: 370px; width: 100%;"></div>
</div>
```

```
chart_extras={"axisY": {"minimum": 0, "maximum": 100, "suffix": "%"}})
```

These changes made little difference to the graph on the attendance homepage but did make it more visually appealing at first glance which is important in website design.



3.31. Site UI/UX Improvements

Now with the bulk of the development completed, I could move onto bringing the design of the website and system up to scratch to be the best it can be for the users on it. This involves improving the user interface (UI) in terms of design and layout as well as the user experience (UX) through simplifying complicated actions and making the system more obvious to use.

3.31.1.Iconography

The usage of icons in a website design is beneficial as it provides the users with a much better way to understand what an element does or means at first glance. Unlike words, which are conventionally used on buttons, icons often convey far more meaning in a smaller package that the human brain can often process much faster than the equivalent words. Additionally, icons often become recognisable to users from other sites and so are already familiar with similar actions.

To implement icons throughout the rota system site, I will be making use of the amazing icon library FontAwesome, the standard web icon library used by many developers and sites.

```
<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.6.3/css/all.css"
      integrity="sha384-UHRtZLI+pbxtHCWp1t77Bi1L4ZtiqrqD80Kn4Z8NTSRyMA2Fd33n5dQ8lWUE00s/"
```

The library will be included directly from FontAwesome's own CDN as it requires many additional resources that would be very large to package directly into the site's source. The integrity attribute can also be used here to ensure that the content received by the browser when it requests this resource is correct and what is expected from the CDN. This is to ensure that no one has intercepted the resource request and changed the content transmitted.

3.31.1.1. Adding the icons

Adding the icons to the system will be best done by me iterating through each template in the source code and adding icons in each where appropriate. This is the best methodical approach to ensuring each and every page is checked for relevant places to implement icons.

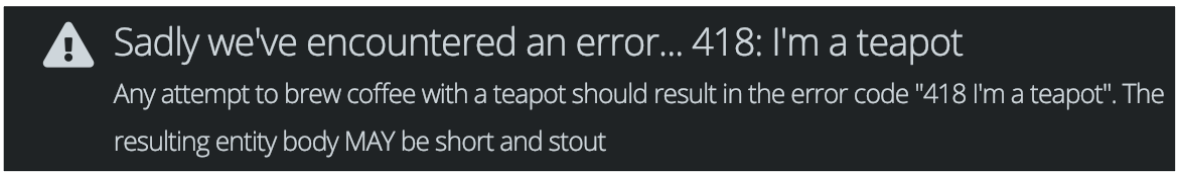
3.31.1.1.1. Error page (app/error.jinja2)

Whilst hopefully this page will never be seen by an end user, it is always best to make it look welcoming and easy to navigate and understand for the user, who may be confused in a situation where they have encountered it.

Here a warning sign icon can be used to convey to the user that there is an issue. I will use a custom written CSS layout here to force all the text to move to the right of the icon, similar to what a media object will do in many CSS libraries.

```
<div class="media">
    <div class="left">
        <h1><i class="fas fa-exclamation-triangle"></i></h1>
    </div>
    <div class="main">
        <h2>
            Sadly we've encountered an error... {{ error_message }}
            <br/>
            <small>{{ error_details }}</small>
            <br/>
            <small>{{ error_extra }}</small>
        </h2>
    </div>
</div>
```

```
.media {
  display: flex;
  flex-direction: row;

  .left, .main, .right {
    padding: .5em;
  }

  .left {
    align-self: flex-start;
    flex-grow: 0;
    margin-right: .5em;
  }
```



3.31.1.1.2. Footer (app/footer.jinja2)

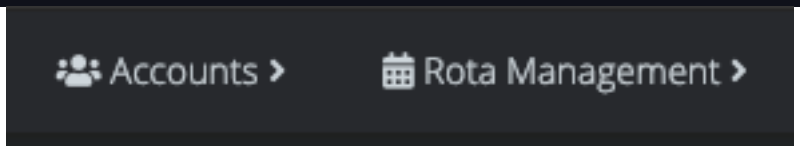
In the footer, there should be a copyright symbol next to my name as the author of the website and system. This can be done with FontAwesome's icons, specifically their regular copyright symbol "far fa-copyright".

```
<i class="far fa-copyright"></i> {{ datetime.now().year }}
<a href="https://mattcowley.co.uk" target="_blank">Matt Cowley</a>.
All Rights Reserved.
```

3.31.1.1.3. Navigation bar (app/navbar.jinja2)

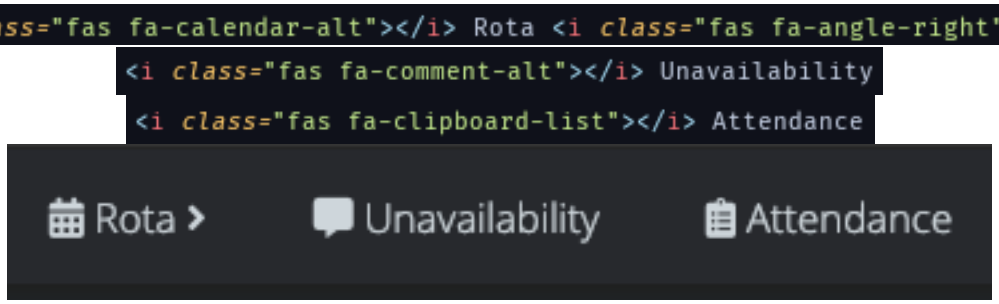
The navigation bar is a classical place in websites where icons can be used to make links and actions far more identifiable to the user. Starting with the links that will be shown to a staff member: Accounts can have an icon with a group of users and Rota Management can have a calendar type icon.

```
<i class="fas fa-users"></i> Accounts <i class="fas fa-angle-right"></i>
<i class="fas fa-calendar-alt"></i> Rota Management <i class="fas fa-angle-right"></i>
```



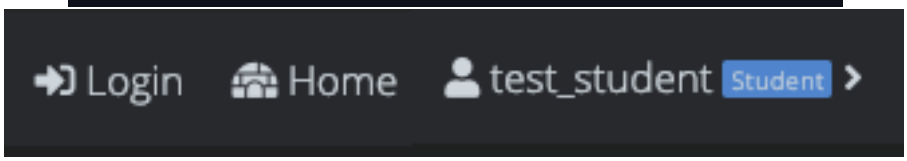
From the student perspective, they see three elements in the navigation bar; Rota which can use the same calendar icon as the rota in the staff navigation, Unavailability which can use a comment icon and Attendance which can be represented with a report card style icon.

```
<i class="fas fa-calendar-alt"></i> Rota <i class="fas fa-angle-right"></i>
<i class="fas fa-comment-alt"></i> Unavailability
<i class="fas fa-clipboard-list"></i> Attendance
```



The navigation bar also contains generic actions which include the home button and the account button which may also be replaced with the login button is no user is authenticated. These will also be given icons to give the navigation bar some uniformity with its icons. The home button would normally have a house but to change it up I went for an igloo which conveys the same meaning but is a bit different. The account button when signed in will simply be a user and when not signed in will be a sign in icon, which FontAwesome has as "fas fa-sign-in-alt".

```
<i class="fas fa-igloo"></i> Home
<i class="fas fa-user"></i> {{ this_user.username }}
<i class="fas fa-sign-in-alt"></i> Login
```

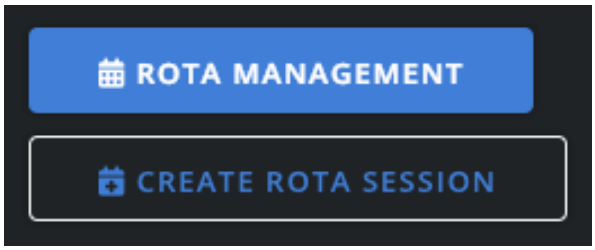


3.31.1.1.4. Staff attendance homepage (attendance/home.jinja2)

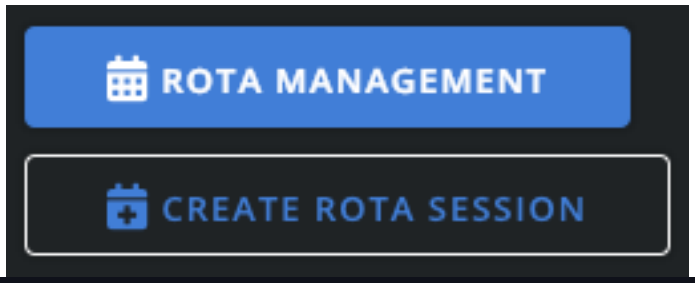
On the attendance homepage for staff there is a collection of buttons presented on the right-hand side to access the main parts of the staff portal. In addition, the attendance breakdown below those also have buttons to view each individual report. All of these buttons can have icons added to them to make them more identifiable quickly to staff on the page.

The two rota buttons will be given calendar icons, with the rota management button sharing the same icon as in the navigation bar for consistency in the site and the create new session button having an icon that is a calendar with a plus sign in it, representative of the action the button carries out.

When these buttons were implemented however, they render quite small on the buttons next to the text, making them less obvious at first glance for users.

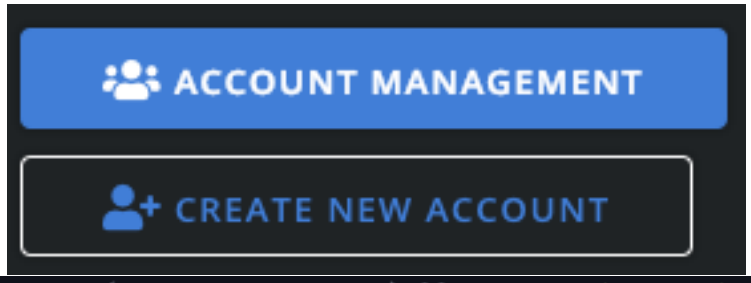


To fix this, I will make use of the icon modifier classes provided by FontAwesome that allow you to modify the size of the icons. In this situation, I can use the class “fa-lg” to make the icon larger than the text.




```
<a href="{{ url_for("staff.rota") }}" class="button button-primary">
    <i class="fas fa-lg fa-calendar-alt"></i> Rota Management
</a>
<a href="{{ url_for("staff.rota_new") }}" class="button">
    <i class="fas fa-lg fa-calendar-plus"></i> Create Rota Session
</a>
```

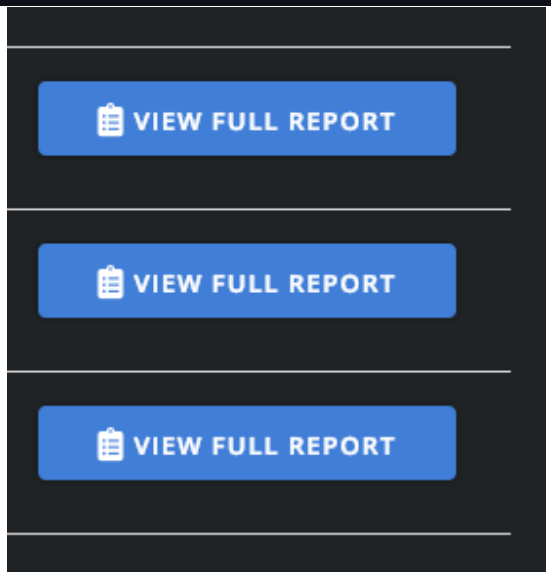
The account related buttons will also be given icons here, with the account management icon being the same one as was used in the navigation bar and the create new account button being given an icon that shows a user with a plus sign next to it, indicating that the button will create a new user. As with the two other buttons above, these icons will be given the large modifier class.




```
<a href="{{ url_for("staff.accounts") }}" class="button button-primary">
    <i class="fas fa-lg fa-users"></i> Account Management
</a>
<a href="{{ url_for("staff.new_account") }}" class="button">
    <i class="fas fa-lg fa-user-plus"></i> Create New Account
</a>
```

To add icons to the buttons for the table on the attendance homepage, they will need to be added in the python controller file as this is where the content for the table is generated. Following the idea of consistency in the site, these buttons will use the same icon that the student would see when accessing their own attendance report in the navigation bar, a clipboard icon.

```
"<a href=\"{}\" class=\"button primary\"><i class=\"fas fa-lg fa-clipboard-list\"></i> View full report"
"</a>".format(url_for("attendance.student", student_id=this_student.id))
```



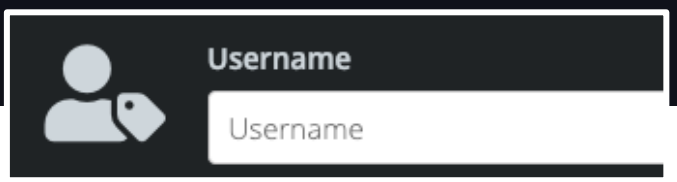
3.31.1.1.5. Login page (auth/login.jinja2)

The login page for the system is a page that has not been touched since it was originally created and is very much an un-loved page. It has a simple title and two labelled fields as well as a submit button.

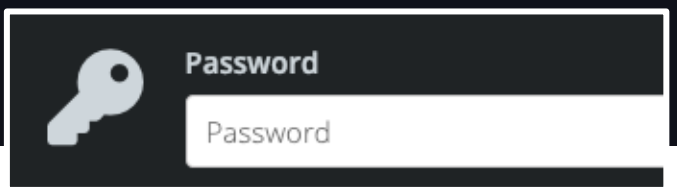
To spice the page up a bit, I will make use of the media classes I created above for the error page to add some large icons to the login form, making it more attractive to the user before they enter the full system.

For the username field, I will use an icon that represents a user and a name tag combined. The password field can be indicated by an icon in the shape of a key. Both the icons will be in the left-hand side of the media layout and will be wrapped in header 1 HTML tags to give them the correct font size in the rendered page.

```
<div class="row">
    <div class="one-half column">
        <div class="media">
            <div class="left">
                <h1><i class="fas fa-user-tag"></i></h1>
            </div>
            <div class="main">
                {{ render_field(form.username, placeholder="Username", autofocus="") }}
            </div>
        </div>
    </div>
```




```
    <div class="one-half column">
        <div class="media">
            <div class="left">
                <h1><i class="fas fa-key"></i></h1>
            </div>
            <div class="main">
                {{ render_field(form.password, placeholder="Password") }}
            </div>
        </div>
    </div>
</div>
```



3.31.1.1.6. All Accounts Management (staff/accounts.jinja2)

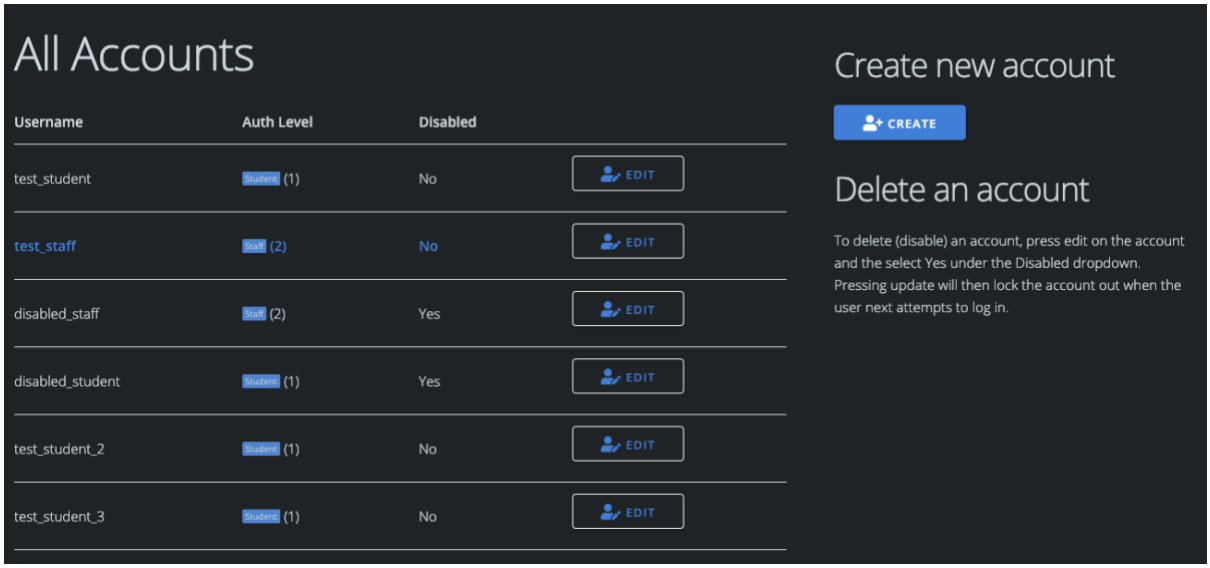
The account management page (all accounts) for staff has many buttons on it. Most are edits buttons next to each account in the table but there is also a create account button on the right-hand side in the sidebar.

The edit account buttons will be given an icon the combines a user and a pencil, a common icon used in the web industry to indicate such an action. The create new account button will be given the same icon as on the main staff homepage (attendance overview), a user combined with a plus icon. Like the previous buttons they will also be given the large class to make them bigger in comparison to the text.

The edit button HTML was created within the python controller and then passed to the table creation macro in the template, so the icon for the edit buttons had to be added in the controller script instead of the template files I was working through. The create account button was in the template so this was easier to implement the icon.

```
"<a href='{}' class='button'><i class=\"fas fa-lg fa-user-edit\"></i> Edit</a>".format(
    url_for("auth.account", id=item.id))
```

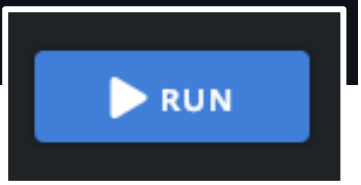
```
<a href="{{ url_for("staff.new_account") }}" class="button button-primary">
    <i class="fas fa-lg fa-user-plus"></i> Create
</a>
```



3.31.1.1.7. Automatic assignments generation (staff/automatic_assignments.jinja2)

This page, the automatic assignment control page, has two buttons on it; run and back. I have decided that I will only give the run button an icon as it is the primary action on the page and the back button shouldn't be used often. The run button can be given a standard icon to represent what it does, a play/run icon. Like all other buttons it will also have the large modifier.

```
<button type="submit" name="submit" class="button primary mt-4">
    <i class="fas fa-lg fa-play"></i> Run
</button>
```



3.31.1.1.8. Create a new account (staff/new_account.jinja2)

Much like the login page, this page is very sparse in its current form. To make it more inviting I will implement the same media layout and icons as the login page as well as adding the normal create account icon to the create button on the page. These changes should make this page for more inviting to the user and appealing to the eye.

However, with these changes made, the page felt unbalanced as the auth level did not have an icon. With an icon added to the auth level, the dropdown would become very small and so to implement an icon for the auth level I would also have to adjust the page layout, the column sizes.

The current column layout was two thirds for the left and one third for the right. Like most web framework grid systems, it was based on a width of 12. To make the page more balanced I replaced the two thirds with seven out of the total width of 12 and the right side with the remaining five.

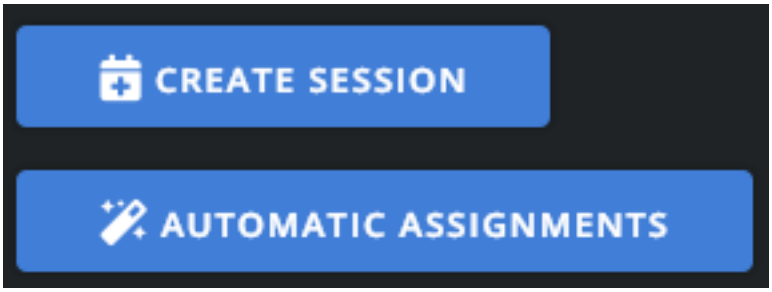
The icon I would use for the auth level should be one that represents something like an authentication card. I decided upon a clipboard with a checkmark after browsing the large collection of icons that FontAwesome offers. With this change made the page felt far more balanced and ready for consumer use.

3.31.1.1.9. Staff full rota view (staff/rota.jinja2)

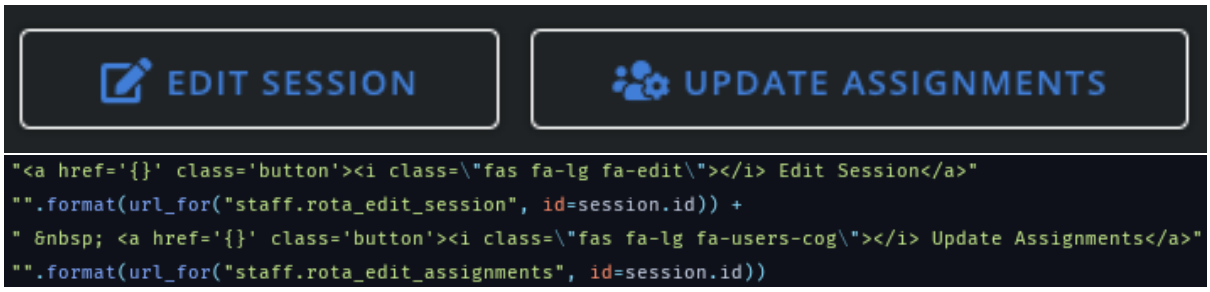
The full student rota view for staff features a multitude of buttons on it that can have icons added to them. At the top of the page there is a create session button as well as the button to access the automatic assignments functionality.

Create session is a button I've encountered before and so the icon for that is already set, a calendar with a plus in it. The automatic assignments button does not have an icon defined yet and so I will need to browse through the icons that FontAwesome

provides to find one that is suitable. After doing some searching I settled on using a magic wand icon to represent the magic that occurs when the automatic assignments feature is run.



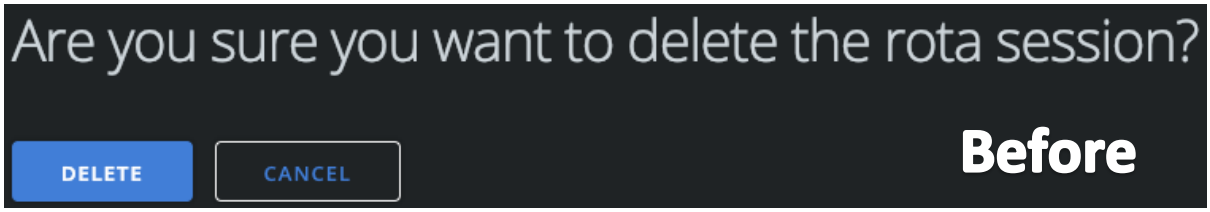
In the table on the page there are two buttons with each row, and edit session button and a button to update assignments for that session. The edit session button will be represented with a pencil edit icon, very standard in web design for an edit button. FontAwesome has the perfect icon for the update assignments button, it is called “users-cog”, two users stacked with a cog next to them which in my opinion represents the button’s action really well.

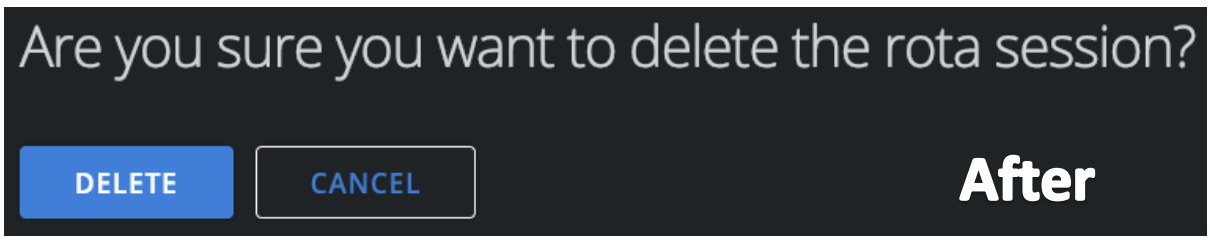



```
"<a href='{}' class='button'><i class=\"fas fa-lg fa-edit\"></i> Edit Session</a>"
"".format(url_for("staff.rota_edit_session", id=session.id)) +
"   <a href='{}' class='button'><i class=\"fas fa-lg fa-users-cog\"></i> Update Assignments</a>"
"".format(url_for("staff.rota_edit_assignments", id=session.id))
```

3.31.1.1.10. Delete rota session (staff/session_delete.jinja2)

The delete confirmation page for a rota session was very plain and contained two small buttons to confirm deletion or to cancel. These buttons were very small on the page in review and so the first thing I did for this page was write a modifier class “larger” to make the buttons display bigger when given this class.



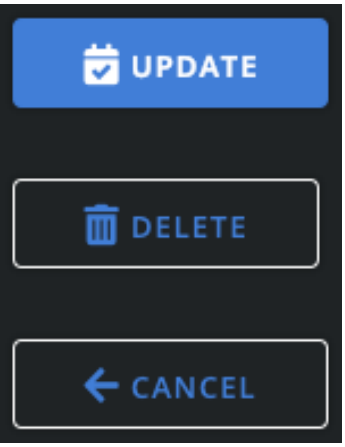


With this change made, only the icons need to be added now. To make it clear which icon does what, the delete button will be given a trash can icon and the cancel a simple arrow to represent going back. I decided to put more emphasis on the primary action, delete, I would not apply the large icon modification class to the back arrow.

3.31.1.1.11. Edit rota session (staff/session_edit.jinja2)

This page contains three buttons on it that can all be given icons. These are a button to confirm the edit, a button to delete the session and a cancel button. These delete and cancel buttons can make use of the same icons as on the delete confirmation page. The update button can be given a checkmark icon to signify exactly what it does, save the changes.

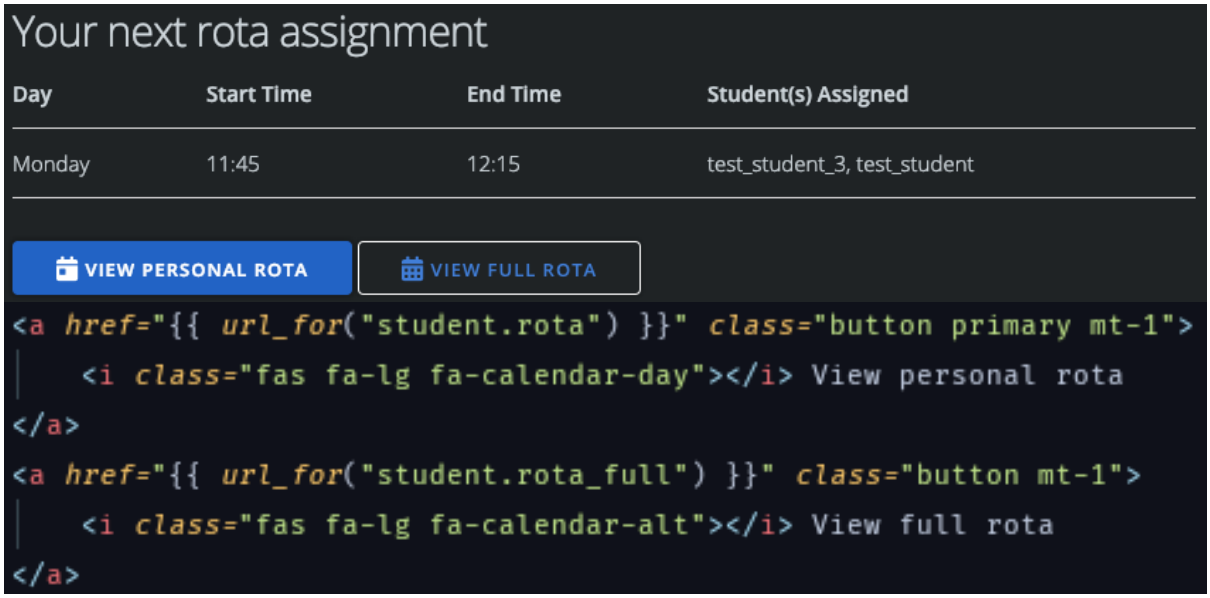
```
<button type="submit" name="submit" class="button primary mt-4">
    <i class="fas fa-lg fa-calendar-check"></i> {{ button_type }}
</button>
{% if id %}
    <br/>
    <a role="button" href="{{ url_for("staff.rota_delete_session", id=id) }}"
       class="button mt-4">
        <i class="fas fa-lg fa-trash-alt"></i> Delete
    </a>
{% endif %}
<br/>
<a role="button" href="{{ url_for("staff.rota") }}" class="button mt-4">
    <i class="fas fa-lg fa-arrow-left"></i> Cancel
</a>
```



3.31.1.1.12. Student portal (student/index.jinja2)

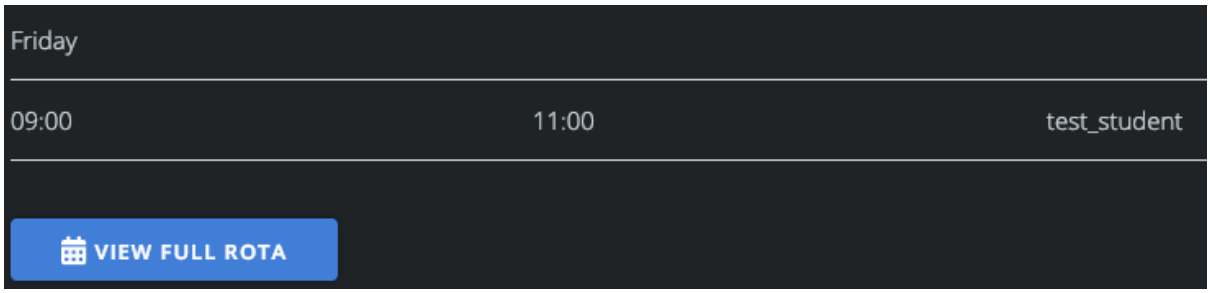
There are two buttons below the rota readout on the student portal, as well as an additional button in the availability sidebar. The unavailability button will share the same icon as it has in the navigation bar. The full rota button can also share the same icon as the one used in the navigation bar at the top of the page.

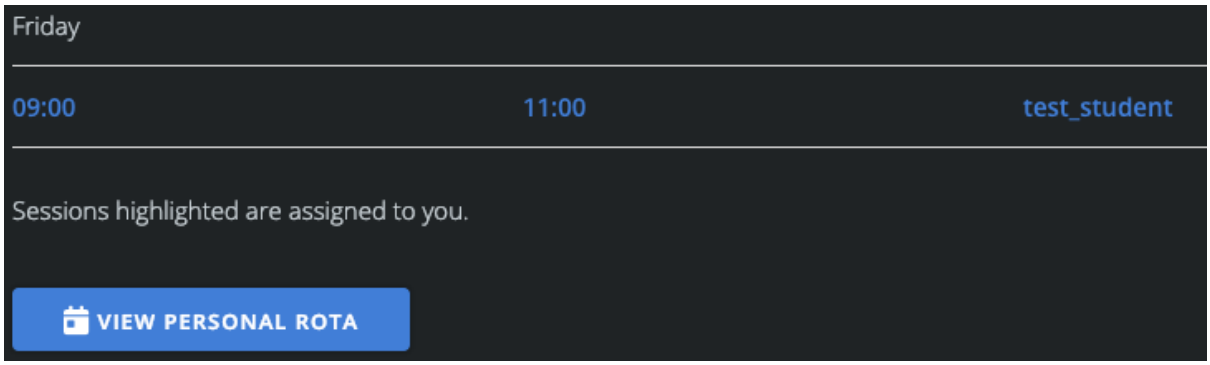
However, the personal rota button really needs a different icon from the full rota view. After looking at all the different calendar icons on FontAwesome, I decided upon the calendar-day icon, which indicates to the user a more specific rota/calendar view than the full calendar icon.



3.31.1.1.13. Student rota view (student/rota.jinja2)

The rota template is used for both the full student rota view as well as the personal rota view. Both of these have a single button at the bottom to view the alternate type of rota. Both of these options are the exact same as found at the bottom of the rota summary shown on the student portal and so the same icons can be used in this template as there.





Much like almost every other icon on a button, these were given the “fa-lg” modifier class to increase the size of the icon in comparison to the text, making the icons stand out more at first glance of the buttons.

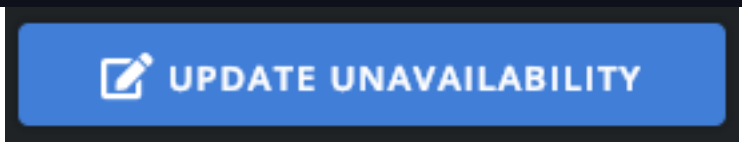
3.31.1.1.14. Student unavailability management (student/unavailability.jinja2)

This page provides the user with an overview of each session and if they have marked themselves as unavailable for it. With this, there is only one type of button on this page and that is the button for each session to update the user unavailability.

This button has not been seen before and so needs a new icon chosen for it. As it is essentially an edit button, I decided that it will use the classic edit icon used throughout the web, a pencil and a notepad.

Start Time	End Time	Unavailable	
Monday			
11:45	12:15	No	Currently assigned, unable to update.
Tuesday			
11:45	12:15	No	UPDATE UNAVAILABILITY
Wednesday			
12:00	21:00	No	Currently assigned, unable to update.
Friday			
09:00	11:00	No	Currently assigned, unable to update.

```
'<a class="button primary mbt-0" href="{}"><i class="fas fa-lg fa-edit"></i> Update unavailability</a>'
''.format(url_for('student.unavailability_edit', id=session.id))
```



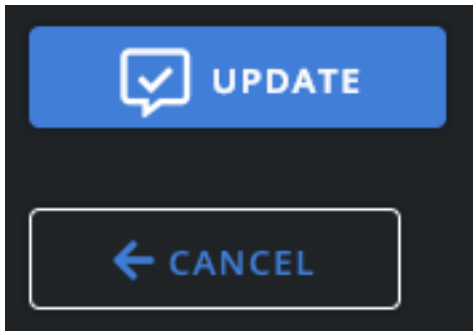
3.31.1.1.15. Edit student availability (student/unavailability_edit.jinja2)

The edit unavailability page is accessed by students using the button in the above view, it features two buttons: update and cancel. The icon I desired for the update button would be the comment icon used for the unavailability in the navigation bar but with a checkmark stacked on to the comment box. FontAwesome did not provide this icon in their package however they do provide a way to manually stack your own icons together.

By using their in-built stacking method, I could combine their outline comment box with a generic checkmark to create the custom icon that I wanted for this button. The cancel button will be given a back arrow like other cancel buttons throughout the rota website.

```
<span class="fa-stack fa-1x">
    <i class="fas fa-check fa-stack-1x"></i>
    <i class="far fa-comment-alt fa-stack-2x"></i>
</span>
Update
```

The resulting buttons were satisfactory and conveyed the meaning of their actions well to any user accessing the site.



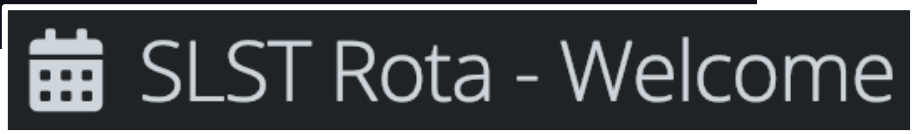
3.31.1.1.16. Default system homepage (index.jinja2)

This page in the system was hardly ever seen by me during the development of the system as I was always signed in as either a student or staff member. The page is only ever displayed when the user isn't logged in as the default page, simply showing a message saying "Hi, log in to get started".

This really isn't an attractive page for a user, which would be the first page they ever see of the system. To improve this I decided not only to implement icons but to redesign this page fully. It would contain a title, some general welcoming text describing the system and a nice call-to-action (CTA) button to sign in.

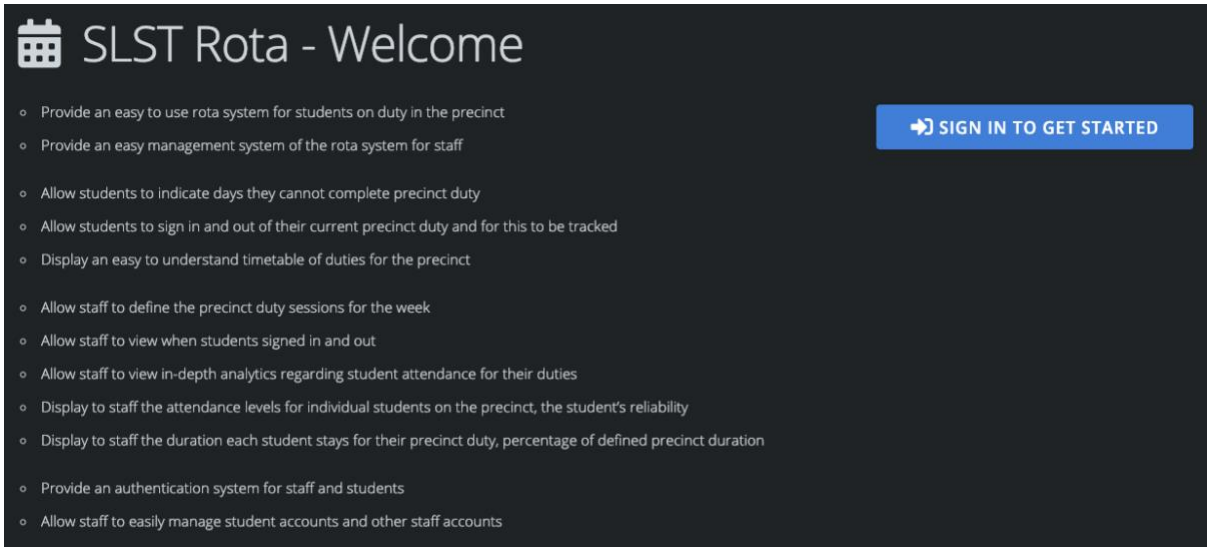
With the title, I will once again make use of my custom media object to display a large calendar icon next to the title, representing the rota management system as a whole. The CTA button would also be given an icon, the same as used in the navigation bar for logging in.

```
<div class="left">
    <h1><i class="fas fa-calendar-alt"></i></h1>
</div>
<div class="main">
    <h1>{{ app.name }} - Welcome</h1>
</div>
```




```
<a href="{{ url_for('auth.login') }}" class="button larger primary mt-1">
    <i class="fas fa-lg fa-sign-in-alt"></i> Sign in to get started
</a>
```

The text I decided to use for this welcome page, describing the system, was my initial set of objectives laid out in this documentation as it does provide a good overview of everything the system should do for both staff and student users of it. With this text in place as multiple lists in the HTML, I was satisfied that this page was completed and ready to go into production.



3.31.2.Sorting table lists

Many pages through the rota system website have lists in tables that would be better if they could be sorted. To do this I could implement a third-party library and most likely have to modify how my tables are generated significantly to make them work with any external library. The alternative option, which I decided would ultimately be easier would be for me to write my own sorting script and modify my tables slightly to support it.

3.31.2.1. Table rendering macro

I started by modifying the template macro used for rendering the existing tables to support me passing in a custom extra value with each field that would be the exact data that the script would eventually support. To ensure all my existing tables would continue to work, this method needed to be optional and backwards compatible. My choice of way to approach this was to check the field value and if it was not a string and iterable with a length of two or more, to assume the first value was the display value for the field and the second value the sort data. If the value was simply a string, like before, it would simply render this in the table.

```
{% if value is not string and value is iterable and value|length >= 2 %}
    <td data-sort="{{ value[1] }}">{{ value[0] }}</td>
{% else %}
    <td>{{ value }}</td>
{% endif %}
```

With this change made I could now modify one of the controllers that makes use of this macro to test whether this change works and adds the data attribute to the fields as expected. The controller I chose to use during development of the sorting was the all accounts table that staff can access. Modifying the value sent for the auth level field, making it a list and providing the raw auth level integer as the second value. I can then check the rendered table to ensure the data value is present.

```
["{} ({})".format(item.auth_label, item.auth_level), item.auth_level],
```

```
<td data-sort="1">
  <span class="label">Student</span>
  " (1)"
</td>
```

The data field was rendered as expected in the template. This was all the templates needed in terms of modification now. The rest of the sorting would all be done from a JavaScript script which will generate all the other data attributes needed and then handle the sorting.

3.31.2.2. The JavaScript sorting

The first step in the JavaScript side of the sorting system is to establish which columns in the table can be sorted. A column will be deemed as sortable if the header field for the column has text in it and every single field in the column has a data-sort attribute with data in it.

3.31.2.2.1. Headers

To achieve this I will first iterate over each header in the table selected and add a blank template of data to an array that I can then once again access later. This array will contain a simple Boolean that indicates if that column can be sorted, the id of the header element which will be needed later for icons and then a subarray that will contain the data needed for each field in the column.

At this stage I cannot check that every field in the column has data but I can ensure that the header has text in it by accessing the “textContent” attribute of the element, trimming it, fetching the length and double not-ing it to convert it to a Boolean which will be used in the blank data for the sortable flag.

```
// Get sortable headers
let sortable = [];
let header_counter = 0;
const headers = document.querySelectorAll( selectors: "table#" + id + " > thead > tr > th");
headers.forEach( callbackFn: (header) => {
    // Set id
    const header_id = id + "-h-" + header_counter.toString();
    header.setAttribute( qualifiedName: "id", header_id);
    header_counter++;
    // Save blank data (false immediately if no header text)
    sortable.push([!!header.textContent.trim().length, header_id, []]); // sortable, id, elements sort data
});
```

3.31.2.2.2. Rows

The next step to parse the table on the page is to iterate through each row in the table and then each field or column in that row. This is where I encountered my first error with this script. Iterating over the rows is done by doing “rows.forEach”. However, I then needed to access all the children of each row. In JavaScript elements have the children attribute which should contain a list of all children elements. I assumed that I would be able to use forEach on this much like the rows.

```
// Iterate over each row
let row_counter = 0;
const rows = document.querySelectorAll( selectors: "table#" + id + " > tbody > tr");
rows.forEach( callbackfn: (row) => {
    // Set row id
    const row_id = id + "-" + row_counter.toString();
    row.setAttribute( qualifiedName: "id", row_id);
    row_counter++;

    // Iterate over each child
    let column_counter = 0;
    row.children.forEach((child) => {
```

When heading into the browser to test this however, I was greeted with an error stating that forEach was not a function. Upon looking into this, it was due to the fact that the children attribute is actually what’s known as a HTMLCollection and not a normal array, so the forEach function was not present. Using the new ES6 standard, I could convert this to an array very easily by doing “Array.from(rows.children)” and then using forEach.

```
Uncaught TypeError: row.children.forEach is not a function
    at rows.forEach (table sort.js:80)
    at NodeList.forEach (<anonymous>)
    at process_table (table sort.js:72)
    at tables.forEach (table sort.js:191)
    at NodeList.forEach (<anonymous>)
```

```
Array.from(row.children).forEach( callbackfn: (child) => {
```

For each child fiend, I can then append its sort data or null to the relevant sortable column array. At this stage I can also disable sorting for a column if a field does not have sort data supplied, which will ensure that each column with sorting enabled has data for every single field in that column.

```
// Add sort data or null
sortable[column_counter][2].push([child.getAttribute( qualifiedName: "data-sort"), row_id]); // data, id
// Disable sort for column if doesn't have sort data
if (!child.hasAttribute( qualifiedName: "data-sort")) sortable[column_counter][0] = false;
```

Whilst iterating over the rows and fields I am also assigning IDs to every element so that I can easily access everything at a later date when I am sorting the elements and updating the HTML.

```
// Set header id
const header_id = id + "-h-" + header_counter.toString();
header.setAttribute( qualifiedName: "id", header_id);
```

```
// Set row id
const row_id = id + "-" + row_counter.toString();
row.setAttribute( qualifiedName: "id", row_id);
```

```
// Set row/column id
const column_id = row_id + "-" + column_counter.toString();
child.setAttribute( qualifiedName: "id", column_id);
```

3.31.2.2.3. Events and styles

With all the sortable data populated in the script, I could the iterate over this to register an event handler to sort the table when each header is clicked on. At this stage I would also verify that the element still exists, just as a safety measure so the event attachment never fails. At this stage, the event handler has no code inside it and is simply a blank function that takes the id for the target table and the index of the column to sort by.

```
// Get index
const header_id = header.getAttribute( qualifiedName: "id").split( separator: "-");
const index = parseInt(header_id[header_id.length - 1]);

// Set click event
header.onclick = (event) => {
    event.preventDefault();
    sort_table(id, index);
};
```

The other thing that I need to do to the headers at this stage is apply some styling to make it obvious that user can click to sort and to keep the UI clean when the user does click. To make the element appear clickable the cursor style can be modified, being set to display the pointer. To keep the UI clean, I will disable text selection so that double click to sort reversed won't select the text. This can be done by setting the user-select style to none.

```
// Set style
header.style.cursor = "pointer";
header.style.userSelect = "none";
```

3.31.2.2.4. Icons

Whilst the headers now show as clickable when you hover over them, it may not be clear to the user that they can actual click on them to sort without accidentally hovering over one of them. Additionally, if the table is sorted, there is currently no way to indicate how it is being sorted. To resolve this I decided I would implement simply icons next to the text for each header in the table. Initially when the table is parsed these will be little icons with an up and down arrow next to each header text.

When the table is then sorted by the user clicking, the icon for the header of the column that is being sorted will be changed to just an up arrow or just a down depending on it the sort is reversed or not (double click). To do this, I created a new function that accepted the table id and the active column index, which could be undefined during table parsing so there is no active sort on the table.

```
const apply_icons = (id, active) => {
    // Get relevant elms
    const table = document.getElementById(id);
    const headers = document.querySelectorAll( selectors: "table#" + id + " > thead > tr > th");
```

With the function created and the elements needed selected, I can now iterate over each header, remove any old icons in the header element and then create the new desired sort icon. To remove any old icons, which would be child elements, I can just set the textContent attribute of the header to the existing textContent. The new icon element can then be created with the base FontAwesome “fas” class and my margin modifier class to give it a gap on the left between it and the text.

```
// Create base icon
let icon = document.createElement( tagName: "i");
icon.classList.add("fas");
icon.classList.add("ml-2");
```

I can then check if the current header if the active one specified; If it is then I can check if the table is reversed or in a normal sort and then append the relevant up or down icon class to the icon element. If it isn’t the active sort header, then both the up and down arrow will be shown in the generic sort icon.

```
// Apply correct type
if (header_counter.toString() === active.toString()) {
    // Check reversed state
    const reverse = table.getAttribute( qualifiedName: "data-sort-reverse");
    if (reverse.toString() === "0") {
        icon.classList.add("fa-sort-up");
    } else {
        icon.classList.add("fa-sort-down");
    }
} else {
    icon.classList.add("fa-sort");
}
```

With the element created and the correct icon classes applied, it was then simply a case of appending the new child element, the icon, to the header element for it to be rendered on the web page. I can now head to the web page to check to see if the initial icons are appended when the table is parsed and initialised for sorting.

Username	Auth Level	Disabled	
test_student	Student (1)	No	EDIT
test_staff	Staff (2)	No	EDIT
disabled_staff	Staff (2)	Yes	EDIT
disabled_student	Student (1)	Yes	EDIT
test_student_2	Student (1)	No	EDIT
test_student_3	Student (1)	No	EDIT

The table rendered as expectd with the icons in the header next to each column. Upon hovering, the cursor styling also appears to the user knows they can click on the element.

3.31.2.2.5. Click event and sorting

The empty “sort_table” function now needs to be populated with the logic needed to sort the rows in the table and re-render them back to the web page for the user when they trigger the sorting through the click events bound to the header text for each column.

The first thing to do in this function, as it is an event handler, is to validate that both arguments given are what we expect and valid. These two arguments are the id of the table, which should be a string and the column index to sort by, which should be integer.

```
// Check given column and id
if (id === undefined || column === undefined || id === null || column === null) return; // null/undefined check
if (typeof id !== "string" || (typeof column != "string" && typeof column != "number")) return; // type check
if (!id.trim().length || !column.toString().trim().length) return; // empty check
```

With these initial checks done, more specific checks and lookups can be carried out to get all the data I need to be able to sort the table and re-render it on the page in the sorted order. This involves fetching the table element from the DOM as well as the sort data from the global variable “table_data”.

```
// Check table exists in data
if (!id in table_data) return;

// Check table exists in DOM
const table = document.getElementById(id);
if (!table) return;

// Get data
let data = table_data[id];
```

Now I have the sort data for that specific table I can check if the column requested is one that can be sorted. If the column requested can be sorted, I can then sort the data in the column using a custom sort function as the sort data is an array of two items, the data to sort and the id of the row the field came from, which will be needed in a bit to re-construct the table.

```
// Check can sort by column
if (column >= data.length) return;
if (!data[column][0]) return;

// Get column data and sort
data = data[column][2];
data.sort(sort);
```

```
const sort = (a, b) => {
    if (a[0] === b[0]) return 0;
    return (a[0] < b[0]) ? -1 : 1;
};
```

With the data sorted, I need to ensure that the data is sorted in the correct direction. If the table is being sorted by a column that is was not last sorted by, the data will always be in the default sort direction. If the table is being sorted by the same column as the last sort, I need to check the last direction and invert that to determine the direct of the data this time.

The sort direction is stored in an attribute of the table in the HTML, the “data-sort-reverse” attribute and the last column the table was sorted by is stored in the “data-sort-last” attribute. Using both of these allow me to determine the correct order and reverse the sort if needed, once set the attributes need to be updated to the new, current, sort.

```
// Set relevant table attrs and reverse if needed
if (table.getAttribute( qualifiedName: "data-sort-last") === column.toString()) {
    // Currently sorted by this column already, check direction
    if (table.getAttribute( qualifiedName: "data-sort-reverse") === "0") {
        // Previous sort was normal, so reverse
        data.reverse();
        table.setAttribute( qualifiedName: "data-sort-reverse",  value: "1");
    } else {
        // Previous sort was reverse, so normal
        table.setAttribute( qualifiedName: "data-sort-reverse",  value: "0");
    }
} else {
    // New column to sort by
    table.setAttribute( qualifiedName: "data-sort-last", column.toString());
    table.setAttribute( qualifiedName: "data-sort-reverse",  value: "0");
}
```

The icons at the top of the table in the headers were designed to indicate that the table could be sorted but also to indicate which column is currently be used to sort the table by. The “apply_icons” function has a second parameter of the column index for the column which is being used to sort currently, which I can make use of here to set the correct icons on the headers of the table.

```
// Do icons
apply_icons(id, column);
```

I can then generate the rows in the correct order for the table by iterating over the sorted data and making use of the second item in each array of the sorted data, the row id, to get the existing element and save it to a temporary array before I can then wipe the table body and write back all the rows in the correct, sorted order.

```
// Fetch rows in correct order
let new_rows = [];
data.forEach((item) => {
    // Get existing row by id and save
    new_rows.push(document.getElementById(item[1]));
});

// Get the table body and wipe all elements inside
const tbody = document.querySelector( selectors: "table#" + id + " > tbody");
while (tbody.firstChild) tbody.removeChild(tbody.firstChild);

// Append new sorted rows back to table
new_rows.forEach( callbackfn: (row) => {
    tbody.appendChild(row);
});
```

3.31.2.3. Testing

With this completed, the sorting script should be complete and I can test it using the initialisation loop at the bottom of the script that searches for every table in the current page and runs it through the initialisation function to attempt to make it sortable by the user.

```
// Initialise sorting
const tables = document.querySelectorAll( selectors: "table");
tables.forEach( callbackfn: (table) => {
    process_table(table);
})
```

Heading to the page being used for testing, the staff all accounts page, showed that the table had been parsed and the sort icons were shown above each column that was sortable. Clicking on one of these headers updated the table, sorting it correctly and updating the icon in the header for that column.

Pressing the same header again, as expected, sorted the table in the reserve order based on the column selected. This also updated the correct icon in the header to indicate the sort direction.

Username	Auth Level	Disabled	
test_student_3	Student (1)	No	EDIT
test_student_2	Student (1)	No	EDIT
test_student	Student (1)	No	EDIT
test_staff	Staff (2)	No	EDIT
disabled_student	Student (1)	Yes	EDIT
disabled_staff	Staff (2)	Yes	EDIT

At this point, I then clicked on a different header to check that the table can sort by any header marked as sortable and that it would correctly update the content of the table and the icons in the header of the table. Upon pressing the disabled header, the table updated and was sorted by the disabled value of the accounts as expected.

Username	Auth Level	Disabled	
disabled_staff	Staff (2)	Yes	EDIT
disabled_student	Student (1)	Yes	EDIT
test_student	Student (1)	No	EDIT
test_staff	Staff (2)	No	EDIT
test_student_2	Student (1)	No	EDIT
test_student_3	Student (1)	No	EDIT

3.31.2.4. Default sorting

I was now satisfied that the code was all working properly and so the next problem I was facing was that the data being rendered at the page load was not sorted and so I decided the best solution to this would be to adapt the table rendering macro and the JavaScript to support running a sort when the table is first initialised in the sorting script by the column specific in the render macro.

The first change needed was to adapt the render template to take a new parameter which would be the column index to attempt to sort the table be at initialisation. This parameter would be a string and would have a default value of a blank string if no column index was passed when the macro was called.

```
{% macro render_table(headings, data, initial_sort_column = "") %}
    <table class="u-full-width" data-sort-initial="{{ initial_sort_column }}">
```

With this change made, I now needed to detect this data attribute in the JavaScript and sort the table at initialisation by this value if it is present and valid. This is where the in-depth checks at the start of the "sort_table" come in very helpful as I don't now need to check the value in the initialisation as the function will filter out any invalid value for me. I do however need the id of the table which I can get by making the table process call return the id it generates which I can then pass into the initial sort call. I can also use the id to select the "data-sort-initial" attribute from the table.

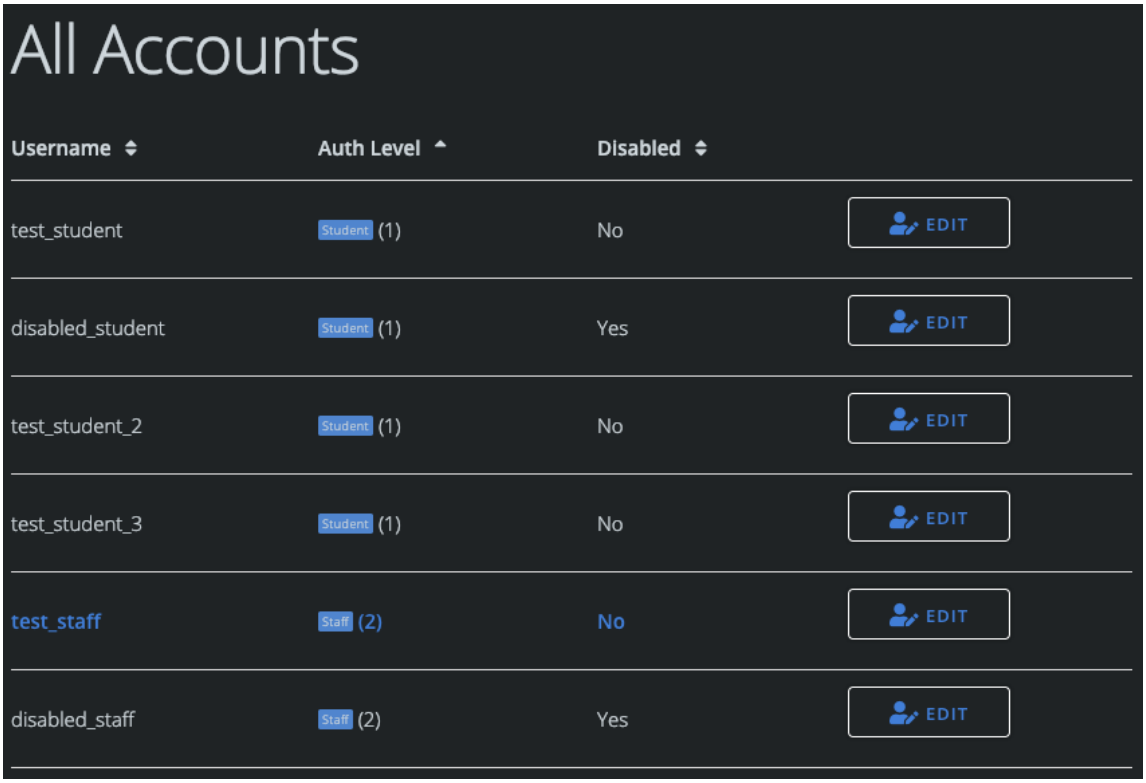
```
const id = process_table(table);
// Attempt initial sort
sort_table(id, document.getElementById(id).getAttribute( qualifiedName: "data-sort-initial"));
```

3.31.2.4.1. Testing

I can now pass this additional value into the table macro from the all accounts page that I have been using to test all the sorting script so far. I pass through the column index 1, which is the authentication level of each account. This should result in the account table being sorted by student and staff accounts which should at first view, make the table easier to understand than the default order from the database.

```
{{ render_table(["Username", "Auth Level", "Disabled", ""], accounts, 1) }}
```

Accessing the all accounts page rendered the accounts table as expected, with the entries sorted by their authentication level as desired. The icons in the header had also updated to indicate the initial sort which was also expected and as should occur. Edge cases don't need to be tested for this as the column index is only ever provided in the code and is not something the user in the system can modify, so it is expected that a correct value will always be provided to the macro and JavaScript.



3.31.2.5. Other tables

With the script and macro completed, I could now work through the whole system and implement this sorting in other tables that it would work on. Many of the tables were already sorted in the source code and could not be sorted by any other value such as the rota readouts and so this drastically reduced where I could implement this sorting system beyond the accounts table.

However, on the homepage for the staff there is a table which displayed an attendance overview for each student in a new row. This is a great candidate for implementing the sorting script as it can be sorted by the student username, their attendance percentage or their punctuality. To do this, the generation of the table data needs to be modified to add the raw sort data.

The raw data for the username is just the username, so Python list multiplication can be used to multiply the existing value and reduce code duplication. For the attendance, the raw percentage value for the attendance present can be used as the sort data. For the punctuality, I had the choice of either using the signing in punctuality data or the signing out. I opted to use the in data as I decided this is the more important statistic for judging how well a student performs on the assignment that when they leave.

```
[this_student.username] * 2,
[report.attendance_bar + "\n" + report.attendance, report.attendance_present],
[report.punctuality, report.punctuality_in_raw],
```

Loading up the page now and inspecting the code reveals that the sort data attributes have been generated correctly and so I can add the sorting script to the template to make the table sortable by the staff user.

```
▼<tr class="highlight">
    <td data-sort="test_student">test_student</td>
  ▶<td data-sort="39.53488372093023">…</td>
  ▶<td data-sort="1.0294117647058822">…</td>
  ▶<td>…</td>
  </tr>
```

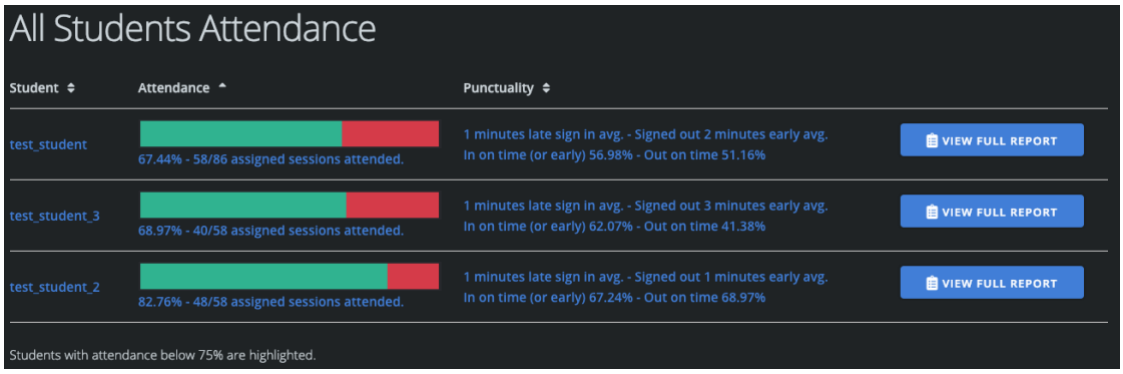
```
{% block extra_js %}
    <script src="/static/js/table_sort.js"></script>
```

With the script loaded on the page, the table was automatically parsed and make sortable with the icons appearing next to each header in the table to indicate this.

The final step was to modify the view template further to provide a default sort column index so that the entries in the table are in a sensible order when the staff user first accesses the page. For this, I decided to use the attendance column as this is the most important stat present in the table for the staff member.

```
{% from "app/macros/table.jinja2" import render_table with context %}
{{ render_table(["Student", "Attendance", "Punctuality", ""], attendance_table, 1) }}
```

With this small change made, the table now rendered correctly with the sortable columns set and the entries sorted by default when the page loads, making this table far more appealing to interact with for the end user on the rota system website.

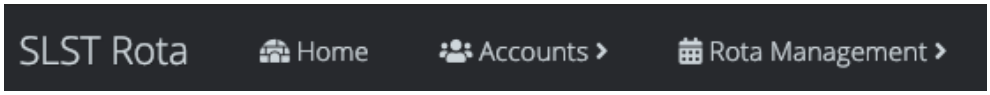


After looking through all pages in the system, I decided there was no other table where this script would be applicable.

3.31.3.Navigation bar

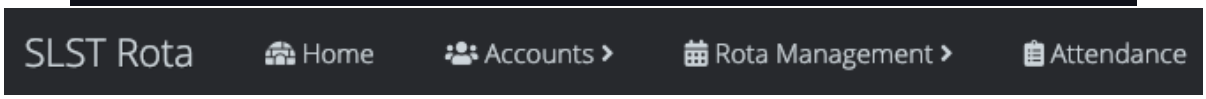
Whilst working through the system I noticed a minor issue on the staff panel that could lead to possible confusion for an end user. The attendance overview page had been made the homepage for the staff portal and so in the staff navigation bar, the home button took you to this page.

However, this could lead to confusion as to how to get to the attendance overview for a user who was new to the system and had not yet realised or learnt that attendance was actually the homepage.



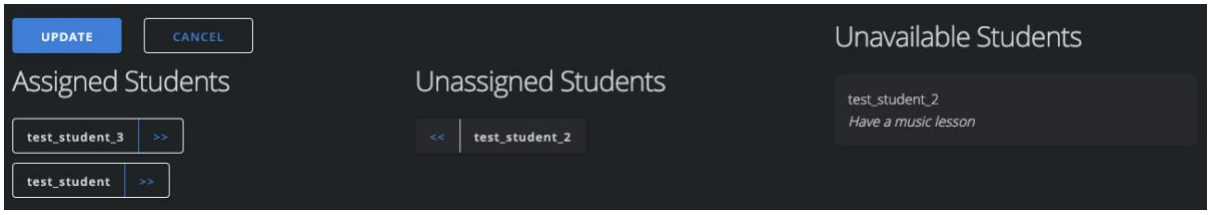
To resolve this issue, I decided to simply create an additional element in the navigation bar that would be a direct link to the attendance overview as well as still having the home button also in the navigation bar. Much like all other navigation bar items this would have an icon and the icon I decided to use would be the same as the one used for attendance on the student navigation bar, a clipboard icon.

```
<a class="item" href="{{ url_for('attendance.home') }}">
    <i class="fas fa-clipboard-list"></i> Attendance
</a>
```



3.31.4.Edit assignments

Another issue I noticed with the staff portal was in the edit assignments page for a rota session. On this page, the staff member has control to assign and un-assign students from the selected rota session. However, this page actually displays no information about the current rota session and so this could lead to easy confusion if a staff member forgets which session they were editing.

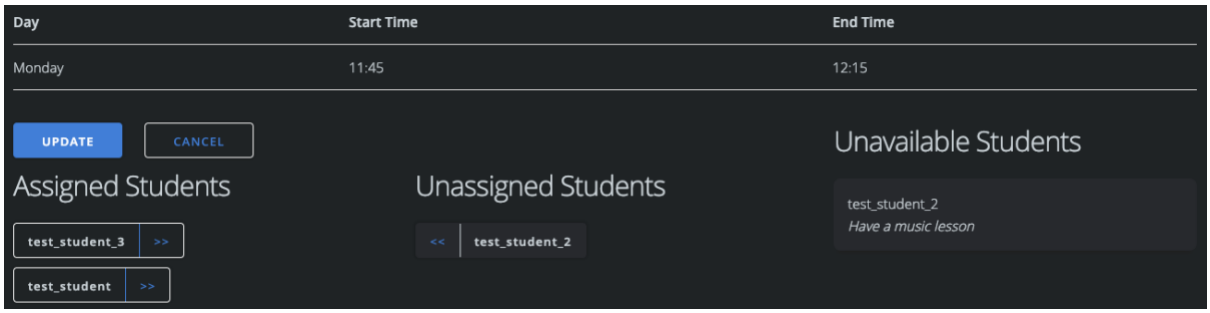


To combat this, I realised that I needed to display key information about the current rota session on this page so that the staff member could quickly and easily identify which session they were updating the assignments for.

The best way to display this data would be in a table similar to how all the sessions are displayed. To do this, I would need to pass the full session object into the Jinja template so that I could then generate the table manually in the template. The table data could then be generated and passed directly into the table rendering macro, which was comprised of the day of the session as well as the start and end times for it.

```
<div class="row">
    {% from "app/macros/table.jinja2" import render_table with context %}
    {{ render_table(["Day", "Start Time", "End Time"],
    [[False, [session.day_frmt, session.start_time_frmt, session.end_time_frmt]]]) }}
</div>
```

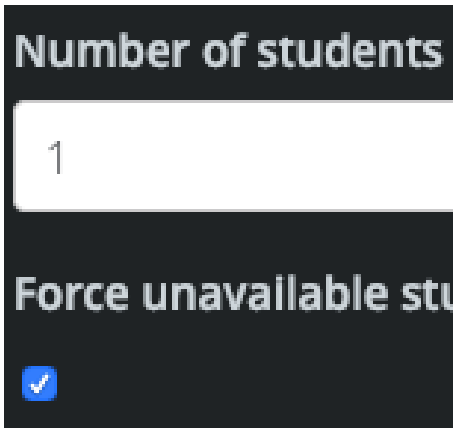
The update assignments page now showed key information about the current rota session to the staff member, making it far less ambiguous whilst on the page which session was being updated.



3.31.5.Checkbox styling

Certain forms in the rota system website make use of a checkbox as a way to interact with the form. This can be seen on the automatic assignments form where a checkbox is used to toggle the force assignment mode as well as on the student unavailability forms where it is used to allow the student to indicate they are unavailable.

In most modern browsers, checkboxes render as very small elements in comparison to the large and styled other inputs that I am using throughout the site's forms. This makes them hard to press for the user, where the page may be accessed through a touchscreen or mobile device in which the click accuracy is reduced significantly compared to using the site with a mouse.



3.31.5.1. HTML changes

To combat this I decided to do some custom styling of the checkbox which requires me to implement a custom HTML structure around the checkbox as the styling that can be done on just a checkbox element is very limiting.

To implement the structure, I would have to modify my form field rendering macro to use a custom HTML layout for checkbox fields instead of relying on the built-in method for creating the field HTML. To do this I need to check the type of the field passed into the macro and switch if a checkbox is detected, known as a "BooleanField" in the library I am using which is WTForms.

```
{% macro render_field(field, placeholder=None) %}
    <label for="{{ field.id }}">{{ placeholder }}</label>

    {% if field.type == "BooleanField" %}
```

With this switch in place I could then render the custom HTML field layout for the checkbox which is comprised of an outer container with tabindex and role attributes so it will behave like a normal input as well as an inner checkbox container which will then house the original checkbox input element and an empty span tag.

```
<div tabindex="0" class="checkbox" role="button">
    <div class="checkbox-inner">
        <input type="checkbox"/>
        <span></span>
    </div>
</div>
```

This structure now needs to be populated with identifying data so that the field will listen to the default value passed in through the WTForms field as well as using the id so that the backend python system can actually identify the input.

I will check the data attribute of the field to see if it should be checked by default and if so, add the checked attribute to the checkbox itself. If the field should not be checked, the unchecked class will be added to the outer container so that I can do custom styling. The id will also be set in both the id and name attributes of the field.

```
<div tabindex="0" class="checkbox{% if not field.data %} unchecked{% endif %}" role="button">
    <div class="checkbox-inner">
        <input type="checkbox" id="{{ field.id }}" name="{{ field.id }}" {% if field.data %}checked{% endif %}/>
        <span></span>
    </div>
</div>
```

3.31.5.1.1. Testing

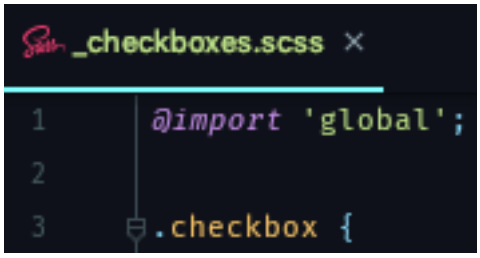
With these changes made to the macro I can now navigate to a page in the system that uses a checkbox to validate that the structure has rendered as expected in the source code of the page. At this time, it will not display anything custom on the page as the styling has not been completed but the underlying DOM structure should be present.

The page I will use for testing as I develop this custom checkbox is the automatic assignments confirmation page as the checkbox is a key part of the form here and very easy to work with whilst signed in as a staff member on the system. Inspecting the checkbox in the developer tools window reveals that the structure for the element has been generated as I expected which means I can now move onto the styling.

```
▼<div tabindex="0" class="checkbox disallowed" role="button">
  ▼<div class="checkbox-inner">
     <input type="checkbox" id="force" name="force">
     <span></span>
   </div>
 </div>
```

3.31.5.2. Styling changes

The styling for this checkbox structure has been taken from another project I recently created and developed, BotBlock.org, where checkboxes were frequently used and so required a custom style for them. I have taken the original styling from this and converted it into a nicer structure using SASS instead of normal CSS which allows for selector nesting.



The checkbox is given a border in the SASS with pseudo elements (:before and :after) being used with specific borders defined to create a tick and cross shape that will be displayed inside the border when the checkbox is checked and unchecked respectively.

When the checkbox is selected, it will also make use of the global primary theme colour for the site to fill in the background of the checkbox to indicate very clearly that it is currently checked. This will be complemented by the tick mark inside. When

unchecked, no background colour will be present in the checkbox and a dim cross mark will be displayed to make clear that it is unchecked.

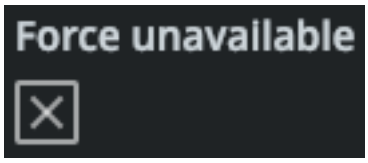
These are the designs from the previous site where the styles for these checkboxes have been taken from an adapted to work with the rota system site.

Unchecked:  Checked: 

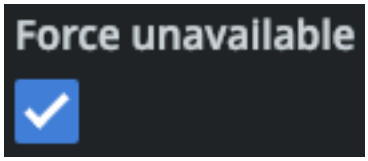
3.31.5.2.1. Testing

With all the styling written and compiled to the minified CSS file that the site uses, I could once again access the automatic assignments form that I was using for testing to see if the styling had applied correctly.

When initially accessing the page, I am expecting the field to be deselected and so it should show a grey bordered checkbox with a faint cross inside it and no background colour applied over the top of the page background.



The checkbox rendered correctly and as expected. To test the checked version, I need to go into developer tools and set the checkbox to be selected as well as removing the unchecked class from the outer container of the custom checkbox. This should then show the checked style of the element with a tick mark over a blue background, the primary colour used throughout the site.



3.31.5.3. JavaScript changes

With the styling completed and working correctly in the web page, the last part of the checkboxes was to make them work when clicked on by the user. This can be done through JavaScript by detecting the click event on the outer checkbox container and then toggling the unchecked class as well as the checked status of the hidden, original checkbox input.

The first step is to select all checkbox container elements in the page and the iterate over them, setting their onclick event to a custom function I have created, which is currently empty.

```
const checkboxes = document.getElementsByClassName( classNames: "checkbox");
checkboxes.forEach(function (elm) {
    elm.onclick = () => {
        handleClick(elm);
    };
});
```

3.31.5.3.1. Testing the selection

At this stage I wanted to check that the selection was working correctly and was picking up the checkboxes, so I loaded this script into the automatic assignments page in its current state and loaded up the page in my browser. The aim was to simply access the checkboxes element in the developer console to check its contents.

However, upon opening the console I was greeted by what was now a familiar 1 after having worked on the table sorting. “checkboxes.forEach is not a function”. It seems much like the querySelector call, the getElementsByClassName call also returns a HTMLCollection instead of an Array, which does not contain the forEach method. Fixing this was easy, much like before, making use of “Array.from” to convert the collection to an array which I could then use forEach on correctly.

```
Uncaught TypeError: checkboxes.forEach is not a function
    at checkboxes.js:13
    at checkboxes.js:18
```

```
Array.from(document.getElementsByClassName( classNames: "checkbox"));
```

With this fix made, I was able to check that the contents of the checkboxes array was as I expected, the single unchecked checkbox displayed on the page.

```
▼ [div.checkbox.unchecked] i
  ▶ 0: div.checkbox.unchecked
    length: 1
  ▶ __proto__: Array(0)
```

3.31.5.3.2. Event handling

I could now move onto writing the click event handler which just needs to toggle the unchecked class on the container and the checked state of the inner checkbox input. To determine the previous state of the checkbox, I would rely on the outer container and the presence of the unchecked class. If the unchecked class was present, I can remove it and set checked to true, if it isn’t present I can apply it and set checked to false.

```
const handleClick = (elm) => {
    if (elm.classList.contains("unchecked")) {
        elm.classList.remove( tokens: "unchecked");
        elm.getElementsByTagName("input")[0].checked = true;
    } else {
        elm.classList.add("unchecked");
        elm.getElementsByTagName("input")[0].checked = false;
    }
};
```

3.31.5.3.3. Final testing

With the script now completed, I could fully test the functionality of the checkbox. Instead of using the automatic assignments page I decided that using one of the unavailability pages would be better as I could then check that the checkbox would also load in the default state of being checked whereas on the automatic assignment page the checkbox will never load in a checked state.

Clicking on the checkbox from the unchecked state at load toggled it into the checked state as expected. Clicking again then toggled it back into the unchecked state so I now know that the JavaScript is updating the classes correctly. To ensure that it is updating the checked value, I clicked it once more to take it back to the checked state and then submitted the page so that it would be saved in the checked state.

Heading back into the same edit page now rendered the checkbox in the checked state by default and so I now knew that the checkbox was working fully and correctly.

3.31.6.Graph changes

On the rota system, there are two graphs that are generated at different points in the system. One is the attendance per session overview graph on the staff homepage and the other is the punctuality breakdown on the student report page. I had noticed minor issues with both that I felt should be resolved to tidy up the UI of the site and make it the best it can be.

3.31.6.1. Attendance per session

The attendance per session graph makes use of the raw calculated percentages for each session and so these have many decimal places in them. The tooltips for each bar in the chart are automatically generated by the library currently and so these make use of the full value passed into the graph.

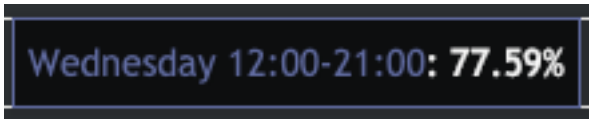
This is not what I'd like as the tooltips contain far too many decimal places and are also not suffixed as they should be with the percentage sign. They simply display the raw floating point value passed in for each bar.

To change this, I referred to the documentation for CanvasJS (https://canvasjs.com/docs/) where I found that you could define a custom tooltip value for each datapoint and still make use of the colour that was automatically assigned to each bar.

CanvasJS allows you to access its own variables inside the tooltip content and so I could make use of the original label and colour whilst costuming the data it used, making use of the rounding ability of Python's string formatting to round all the raw values to 2dp and then append the percentage sign for the tooltip.

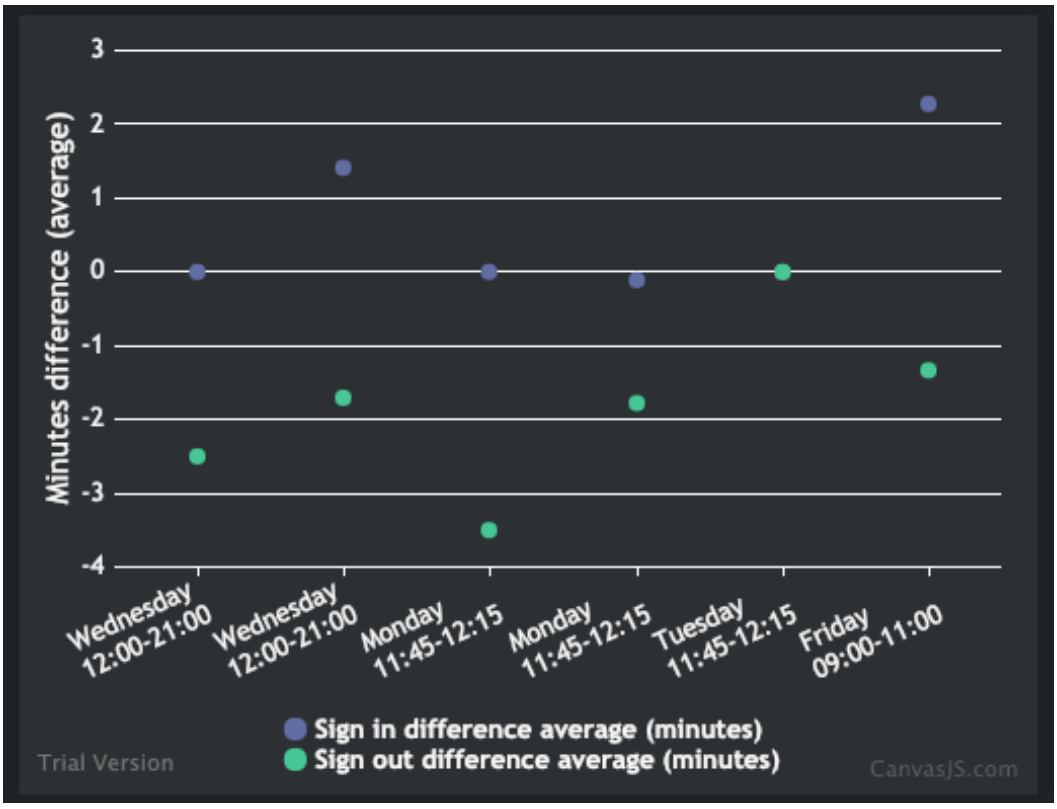
```
"label": "{0.day_frmt} {0.start_time_frmt}-{0.end_time_frmt}".format(data[0]),
"y": value,
"toolTipContent": "<span style='\"'color: {{color}};'\"'>{{label}}</span>: {:.2f}%".format(value)
```

This change would make use of the original label from before and retain the colour of it whilst providing the rounded value instead. To confirm this change had worked correctly I turned the system back on and loaded up the staff portal page. Hovering over one of the bars in the chart revealed that the tooltip had updated with this change and now displayed the rounded value as expected.



3.31.6.2. Punctuality graph

The issue I had noticed with the other graph was how small the data point markers were on the chart. I had considered making this a line graph to make the locations of each data point more obvious but decided against this due to the fact that the data wasn't done over time and was individual items of data for each session.

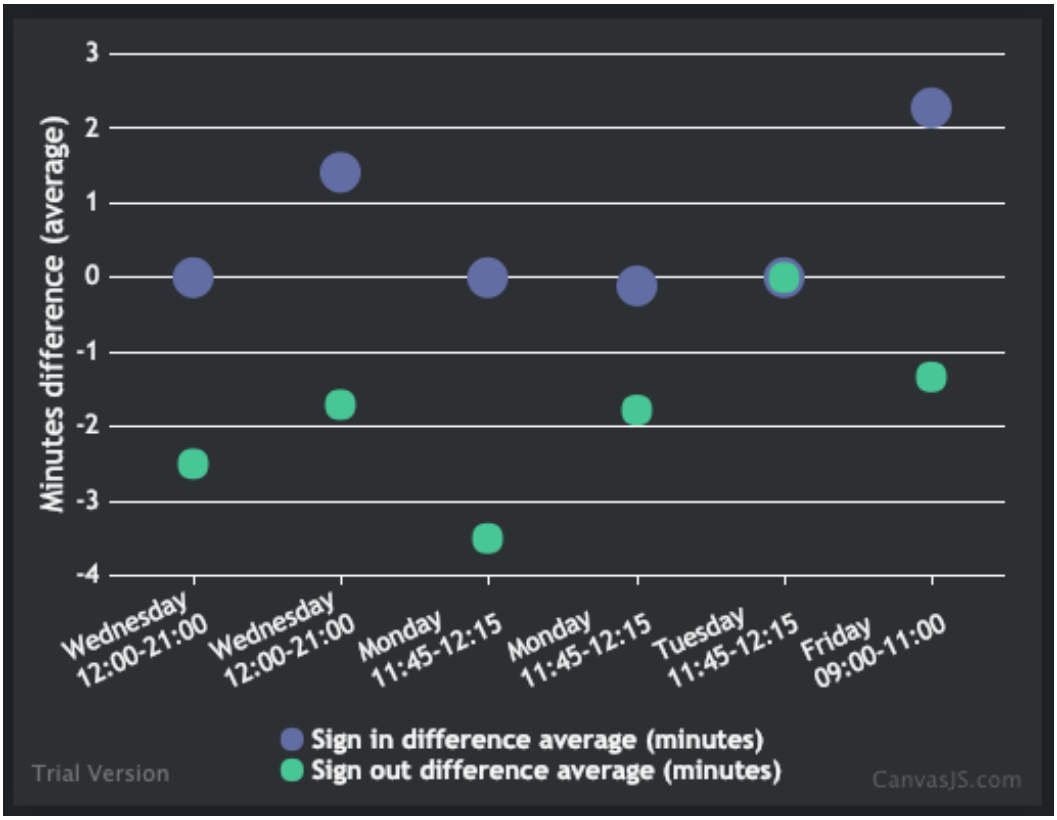


In some places, both data points would also fall on the same value and so observing both would be impossible which could lead to confusion about the data. To resolve this I decided to make both data points much larger but with one larger than the other so both were visible even if overlapping.

Referring once again to the CanvasJS documentation, I located the attribute I was looking for to achieve this, the “markerSize” property. This allows you to set the size of each marker in the data set to a custom size on the rendered graph. Using this I could specify the sizes for the data points in each data set and then pass that into the CanvasJS render call.

```
"y": assignment.in_diff_avg, "markerSize": 20})

"y": assignment.out_diff_avg, "markerSize": 15})
```

Reloading the page with these changes produced a great improvement to the graph with the data points being far more noticeable at quick glance and the difference in size between the two data sets means that when the points overlap both data points for that session (in and out) are visible.



4. Testing & Evaluation

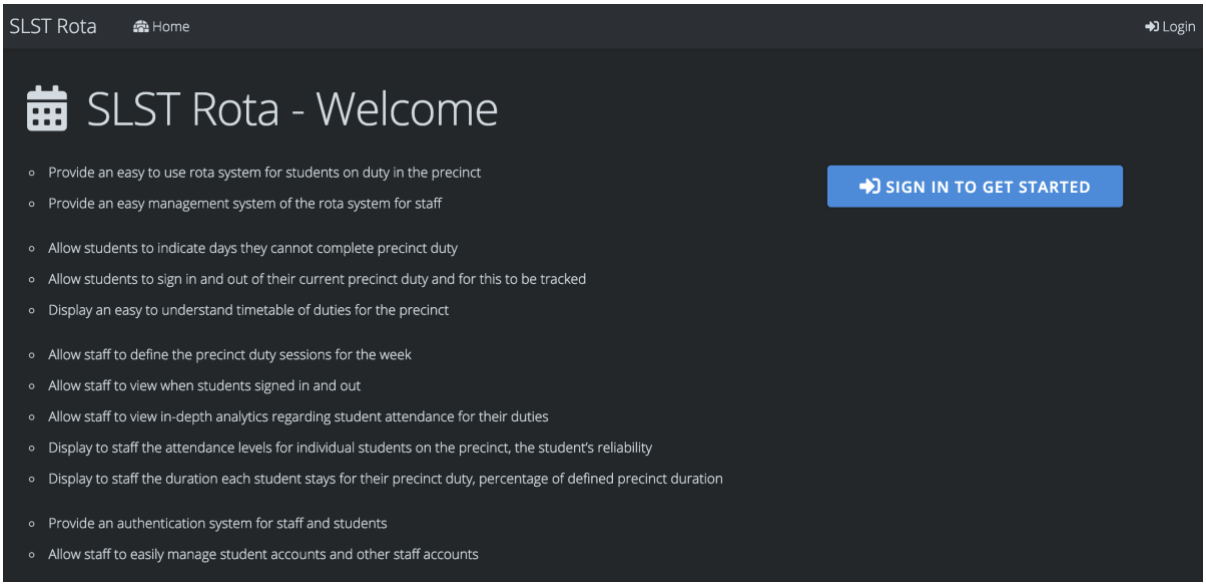
4.1. Testing to inform evaluation

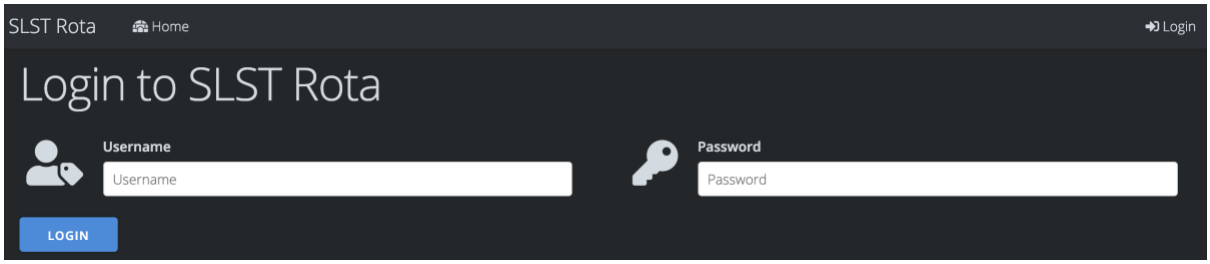
With the program fully developed it is no time to run a full test over the entire finished system before it can be released in a closed beta for end-user testing and feedback.

To do this with the most efficiency, I wiped all parts of the database except for one account so that the system would be starting from a new install and working from there to test each part of the system effectively. The only account in the system database is a staff account named “test_account” with the password “test”.

4.1.1. Logging in

The first part of the system to test is signing in correctly. To do this, the user will first reach the homepage where there is a big button show asking them to sign in. Clicking this takes the user to the log in form page as expected.



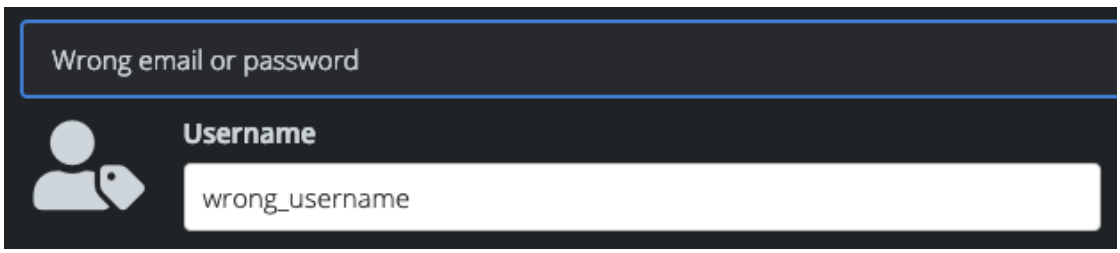


4.1.1.1. Bad data

The first test that can be carried out here is using bad data that is expected to fail at sign in. There are three different bits of bad data that we can use here and I will test each of these situations and record the result below along with the expected result.

Bad data type	Expected result	Actual result	Success
Correct username, bad password	Error message stating username or password incorrect	Error message stating username or password incorrect	√
Bad username, correct password	Error message stating username or password incorrect	Error message stating username or password incorrect	√
Bad username, bad password	Error message stating username or password incorrect	Error message stating username or password incorrect	√

The error message is displayed to the user in a large box above the form making it very obvious that it has occurred:



Following this testing it is clear that the login form can handle bad data correctly.

4.1.1.2. Extreme data

I can also test the form with extreme data, designed to cause errors and break the validation or form. Common extreme data would be making use of escape characters such as the ones listed in the table below.

Extreme data type	Expected result	Actual result	Success

Text ending with Python string escape character, a backslash '\'	Standard error of incorrect username or password.	Error message stating username or password incorrect	√
Text ending with SQL statement termination character, a semicolon ';'	Standard error of incorrect username or password.	Error message stating username or password incorrect	√

These tests indicate that the text passed into the form is being handled correctly by the system in a sanitised method, which means the form is safe from injection attacks.

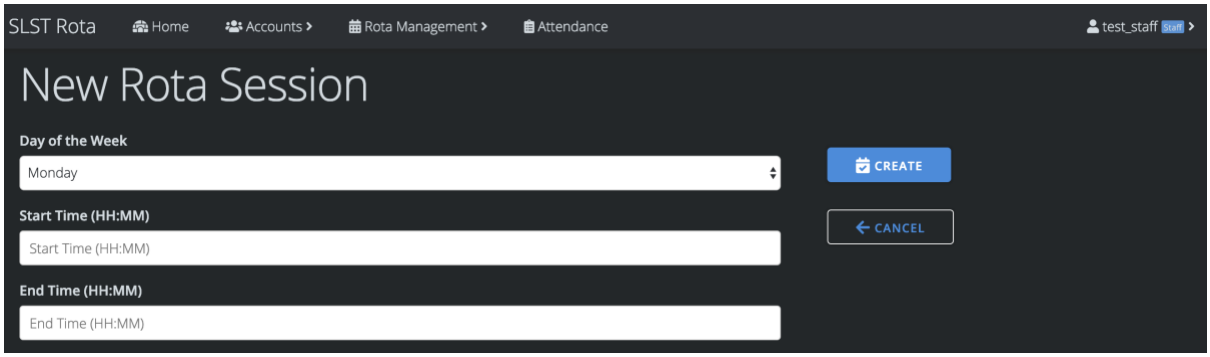
4.1.1.3. Good data

The last and most important test for the login form is that correct, "good" data works and allows the staff account, in this situation, to sign in and access the system without error.

Inputting the correct username and password combination and then pressing the login button works correctly and takes the staff member to the staff homepage as expected.

4.1.2. Creating new rota sessions

The next step in testing the system fully is to ensure that all the session creation works correctly as this will be an important step for a staff member.



4.1.2.1. Bad data

Due to the setup of the system, there is no way to pass in bad data for the day of the week. It is a predefined set of choices in a dropdown. In the final deployment, this

would be displayed on a touchscreen, locked down device so that a user could not modify the source to send a custom value.

The start and end times on this page however could have bad data submitted and so this must be tested. As both fields are validated using the same methods, I will use the start time for all testing.

Bad data type	Expected result	Actual result	Success
Incorrect separator “13.00”	Error message stating to use the correct format	“start_time: Please enter a start time and ensure it is in the correct format.”	√
Missing minutes “13”	Error message stating to use the correct format	“start_time: Please enter a start time and ensure it is in the correct format.”	√
Missing separator, space “13 00”	Error message stating to use the correct format	“start_time: Please enter a start time and ensure it is in the correct format.”	√
Missing separator, nothing “1300”	Error message stating to use the correct format	“start_time: Please enter a start time and ensure it is in the correct format.”	√
Negative hours “-13:00”	Error message stating to use the correct format	“start_time: Please enter a start time and ensure it is in the correct format.”	√

The other possible type of bad data that could be submitted to this system is a situation where the start time is greater than or equal to the end time, which would obviously be impossible and so the system must be checked to ensure it considers this as part of the form validation.

Bad data type	Expected result	Actual result	Success
Start time equals end time	Error message stating start time must be before end time	“Start time must be before end time”	√
Start time greater than end time	Error message stating start time must be before end time	“Start time must be before end time”	√

4.1.2.2. Extreme data

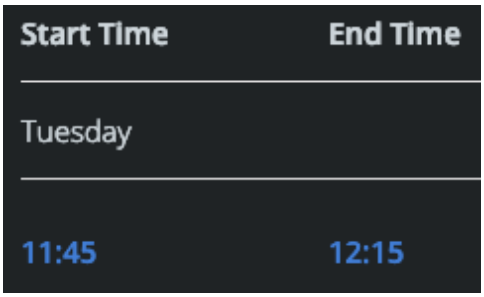
Like the previous form, the login form, the same extreme data test suite can be carried out to ensure that injection attacks will not work on the system.

Extreme data type	Expected result	Actual result	Success
Text ending with Python string escape character, a backslash '\'	Error message stating to use the correct format	"start_time: Please enter a start time and ensure it is in the correct format."	√
Text ending with SQL statement termination character, a semicolon ';'	Error message stating to use the correct format	"start_time: Please enter a start time and ensure it is in the correct format."	√

4.1.2.3. Good data

Knowing that the form can handle any bad data a user may throw at it, I can now test to ensure that it will create a session in the rota when given valid data.

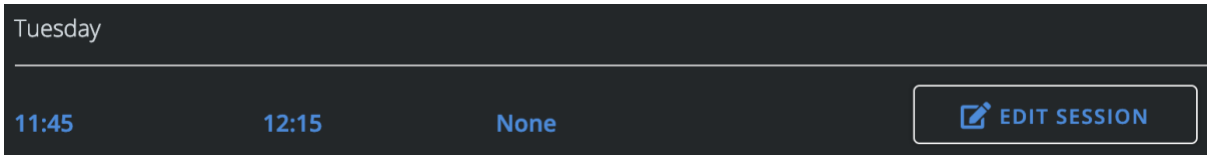
To test that it works correctly, I will create a session on Tuesday from 11:45am till 12:15pm. The inputs will be as follows: "Tuesday" selected from the dropdown, 11:45 for the start time and 12:15 for the end time. Pressing the create button redirects the user back to the full rota view where the new session is now created and present.



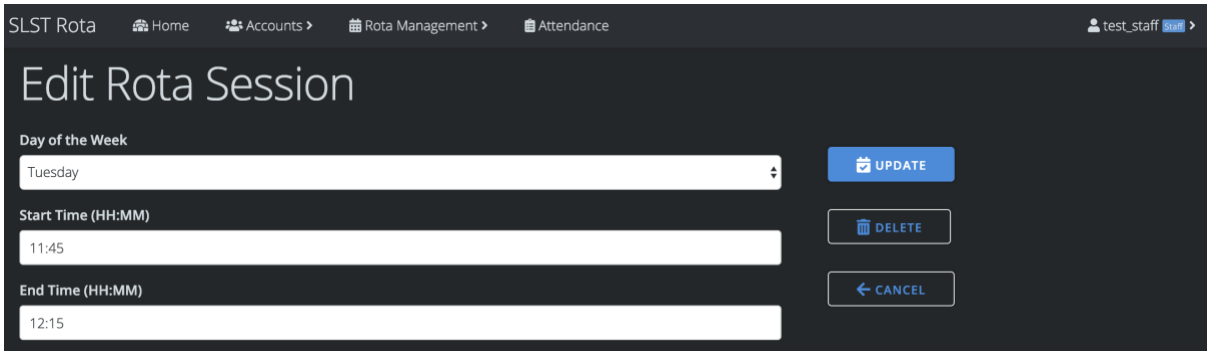
4.1.3. Editing a rota session

The session edit and new rota session both use the same form from WTForms and route in the Flask application, so there is no need to test this again for correct data handling.

It is important however to test that it will pre-fill the data of the current session selected correctly.



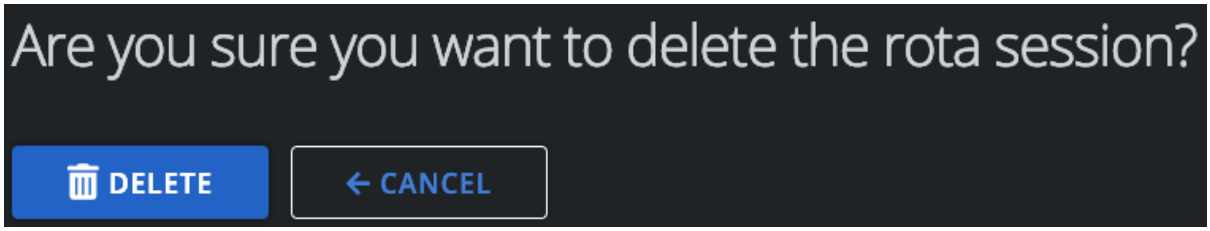
Pressing edit on the first session in the rota, the one just created reveals that it does correctly pre-fill the data for the staff user to edit. √



4.1.4. Deleting a rota session

The next action that I can test is the ability to delete a session. This is simple to test as there is no user input beyond the delete button being pressed, however I will need to confirm that the session was indeed deleted and marked as such in the database to be satisfied that it is functioning correctly.

Pressing the delete button on the edit rota session page takes the user to a confirmation page for the deletion with large buttons to delete or cancel. These should both be tested for correct functionality.



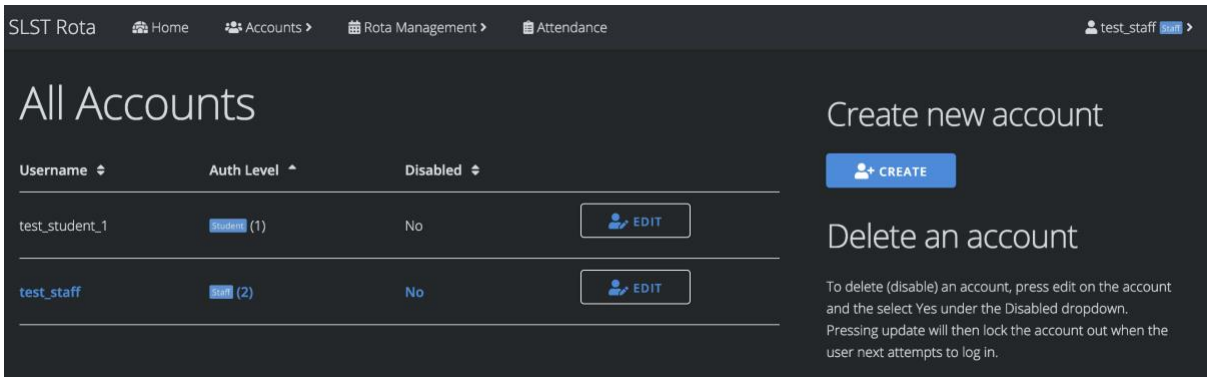
Button type	Expected result	Actual result	Success
Cancel	Redirects user to the edit page	User taken back to the edit page for the session	√
Delete	Marks session as deleted in DB and redirects user to rota overview	User is redirected to rota view. Inspecting DB confirms session	√

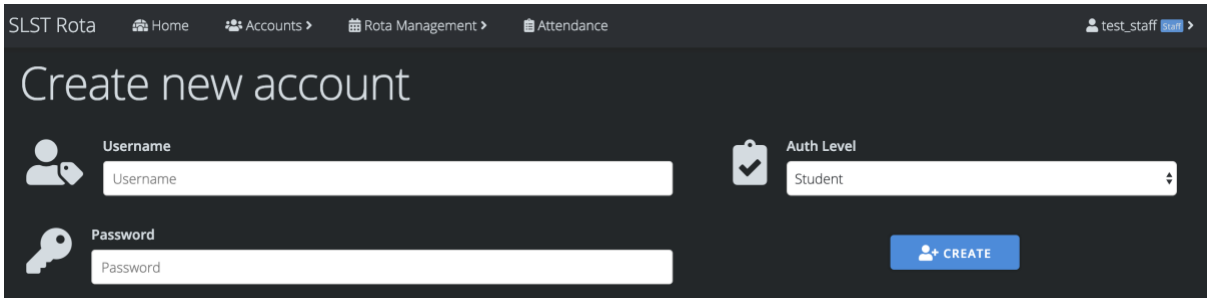
		was marked as archived.	

Both buttons did what was expected of them, with delete marking the session as archived in the database. This is done instead of fully deleting so that attendance records can still reference a now deleted session etc.

4.1.5. Creating an account

With the rota sessions created and tested, the next thing a staff member may need to do on a new system is to create accounts for the students and other staff that are responsible for the precinct.





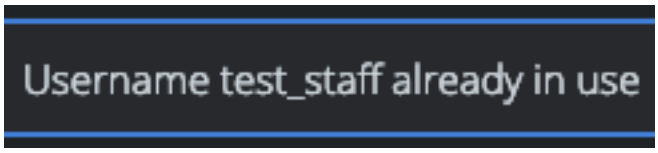
The create account form has three fields; username, password & authentication level. Username and password are string inputs, with authentication level being a pre-defined dropdown choice.

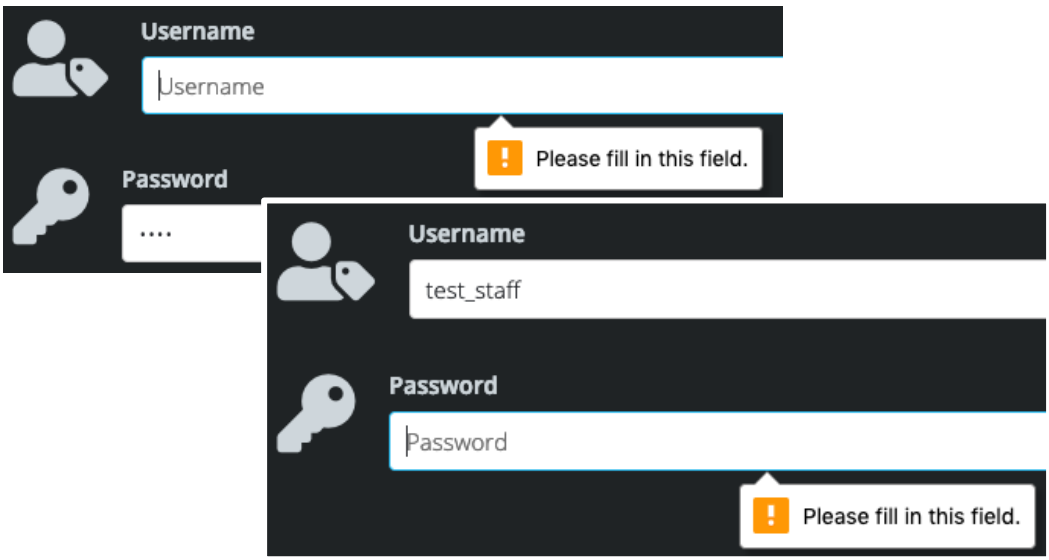
4.1.5.1. Bad data

There are four major bad data cases for the new account form. A username that is already in existence in the system or missing fields (username, password or username & password).

Bad data type	Expected result	Actual result	Success
Existing username “test_staff”	Error stating username is already being used	“Username test_staff already in use”	√

No username, password present	Client side warning that field is required	“Please fill in this field” browser warning	√
No username, no password	Client side warning that fields are required	“Please fill in this field” browser warning	√
Username present, no password	Client side warning that field is required	“Please fill in this field” browser warning	√



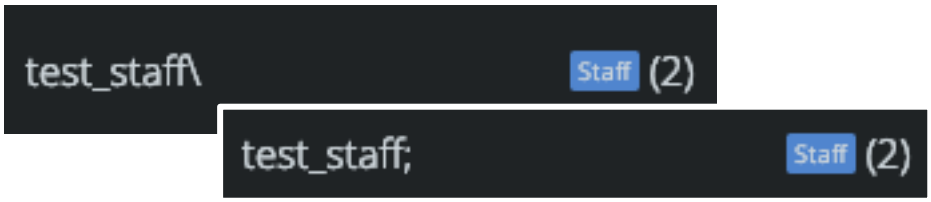


4.1.5.2. Extreme data

As the inputs for this form are also string inputs, the now standard injection test suite can be used once again to ensure that neither field are susceptible to injection attacks.

Extreme data type	Expected result	Actual result	Success
Username field – Text ending with Python string escape character, a backslash ‘\’	Account created with ‘\’ present in username, escaped as needed internally.	User redirected to accounts overview page – account created with ‘\’ present	√
Username field – Text ending with SQL statement	Account created with ‘;’ present in username, escaped	User redirected to accounts overview page – account	√

termination character, a semicolon ‘;’	as needed internally.	created with ‘;’ present	
Password field – Text ending with Python string escape character, a backslash ‘\’	Account created with ‘\’ present in the password, escaped as needed internally.	User redirected to accounts overview page with no error – Signing in to account, password has ‘\’ present.	√
Password field – Text ending with SQL statement termination character, a semicolon ‘;’	Account created with ‘;’ present in the password, escaped as needed internally.	User redirected to accounts overview page with no error – Signing in to account, password has ‘;’ present.	√

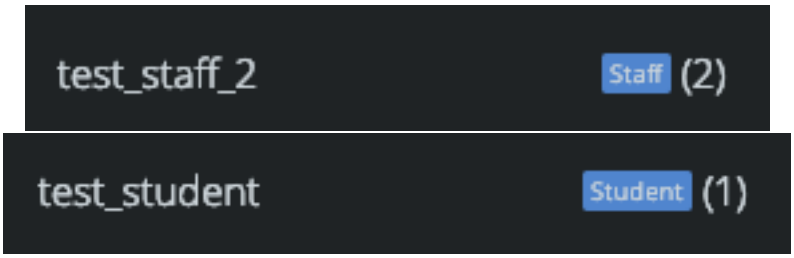


4.1.5.3. Good data

The last part to check of creating an account is that when a user does input good data, that it works correctly to create the accounts.

The system supports two types of account that can be created, a student account which is used for the precinct monitors to sign in and out of their assigned rota sessions.
In addition, there are also staff accounts for managing the rota and assignments. These both need to be tested as part of the account creation form.

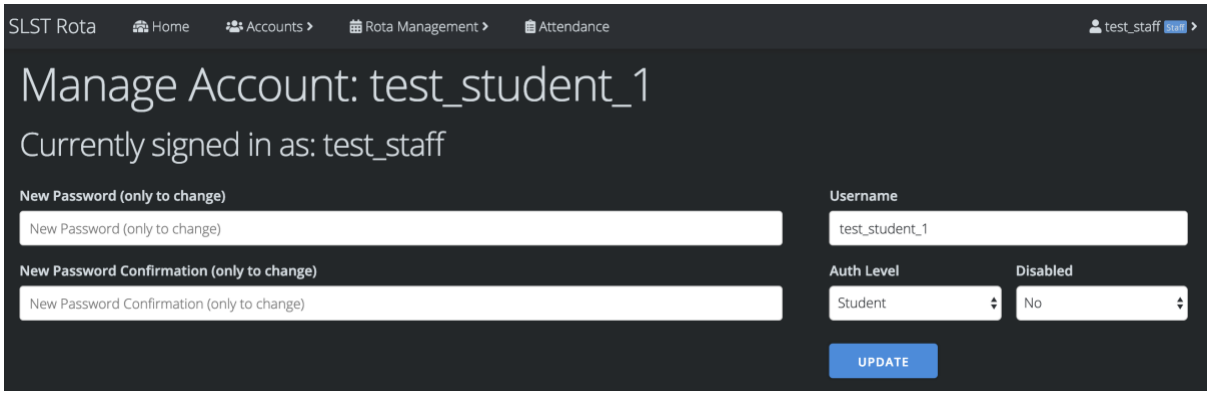
Account type	Expected result	Actual result	Success
“test_staff2” – Authentication level “Staff”	Redirects user to accounts overview, “test_staff2” created as staff account	User taken back to the accounts page, “test_staff2” is listed as a staff account	√
“test_student” – Authentication level “Student”	Redirects user to accounts overview, “test_student” created as student account	User taken back to the accounts page, “test_student” is listed as a student account	√



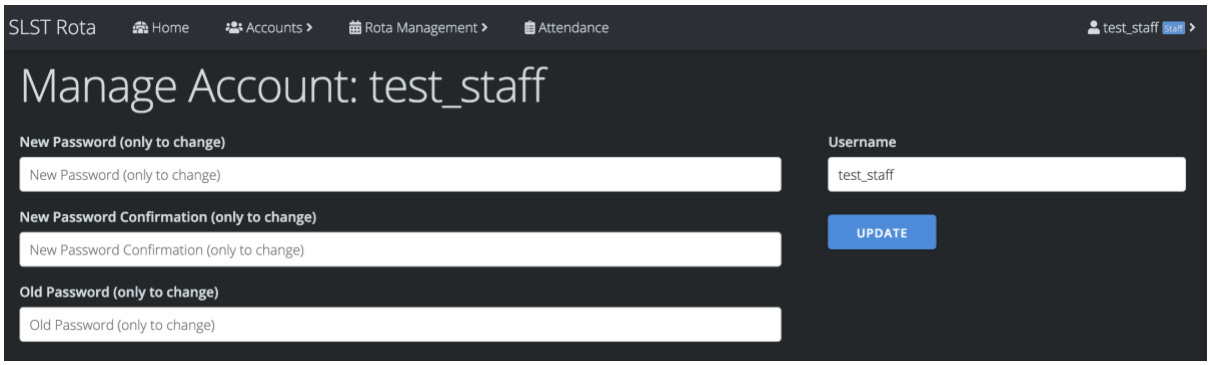
4.1.6. Editing an account

Another key ability of the account management system is the ability for staff members to edit accounts.

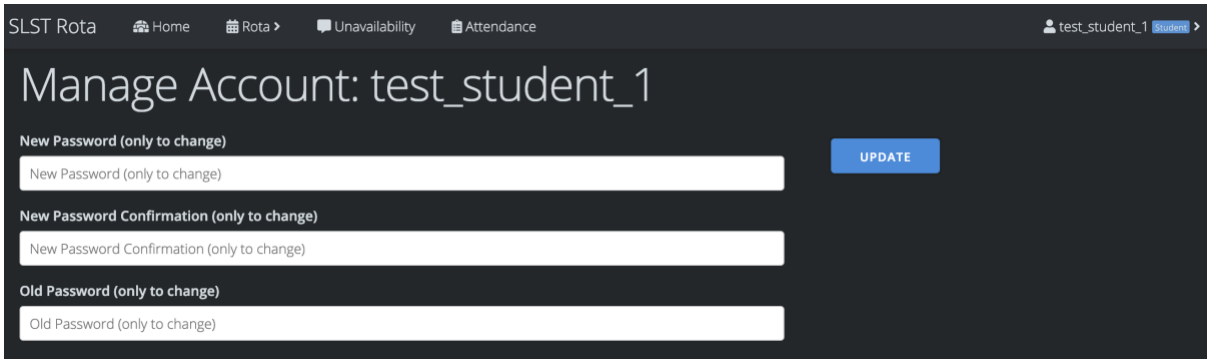
When a staff member edits another account, they have the ability to change the password, change the username, change the authentication level and disable an account.



When a staff member edits their own account however, they also have to enter their old password when changing it to a new one.



A student can also edit their own account and have the same form as when a staff member edits their account for changing their password but a student cannot change their username, authentication level or disable their own account.



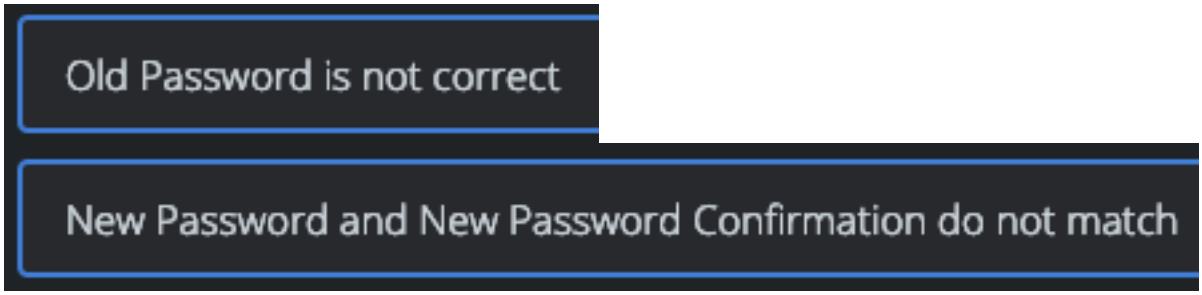
Therefore, I will need to test the following situations:

- Staff member editing their own account
- Staff member editing another account
- Student editing their own account

4.1.6.1. Staff member – Own account

Situation	Expected result	Actual result	Success
Password change – New password & confirmation match, old password is incorrect	Error message that old password does not match the one in the database	Message shown that old password is wrong	√
Password change – New password does not match confirmation, old password correct	Error message that new and confirmation do not match	Message shown that new and confirmation do not match	√
Password change – New password does not match confirmation, old password incorrect	Error message that new and confirmation do not match	Message shown that new and confirmation do not match	√
Password change – New password & confirmation match, old password is correct	Password is changed, user shown message confirming change	Message shown confirming password is changed	√
Username change – New username already in use	Error stating the new username is already being used	Message shown that username is already in use	√

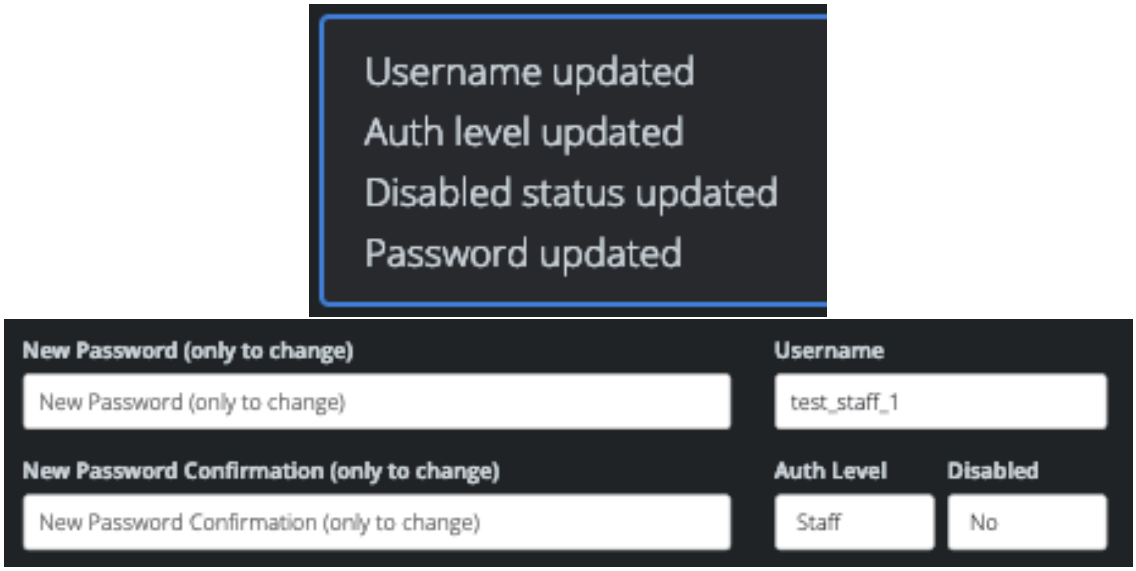
Username change – Blank username input	Error stating username is required	Message shown asking for a username	√
Username change – New username is valid	Username is changed, user shown message confirming change	Message shown confirming username is changed	√



4.1.6.2. Staff member – Another account

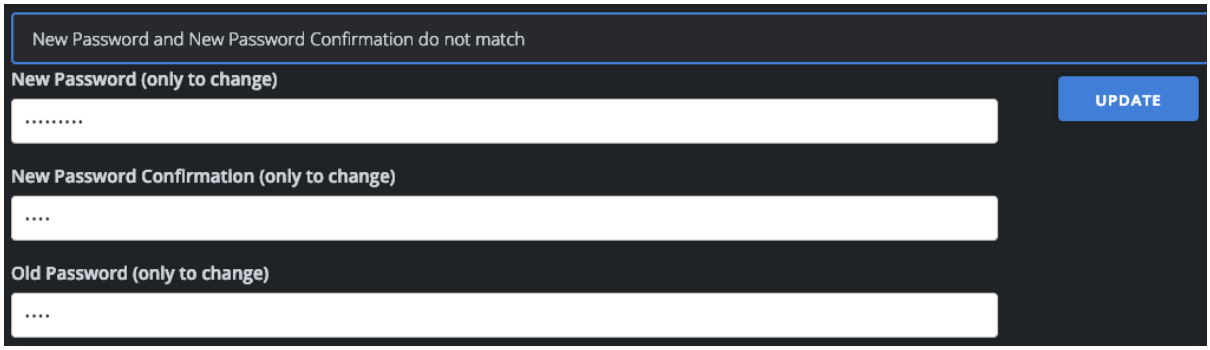
Situation	**Expected result**	**Actual result**	**Success**
Password change – New password does not match confirmation	Error message that new and confirmation do not match	Message shown that new and confirmation do not match	√
Password change – New password & confirmation match	Password is changed, user shown message confirming change	Message shown confirming password is changed	√
Username change – New username already in use	Error stating the new username is already being used	Message shown that username is already in use	√
Username change – Blank username input	Error stating username is required	Message shown asking for a username	√
Username change – New username is valid	Username is changed, user shown message confirming change	Message shown confirming username is changed	√
Authentication change – Student account to staff account	Authentication level changed, user shown confirmation message	Message shown confirming level is changed	√

Disabled status change – Active changed to disabled	Disabled status changed, user shown confirmation message	Message shown confirming status changed	√



4.1.6.3. Student account – Own account

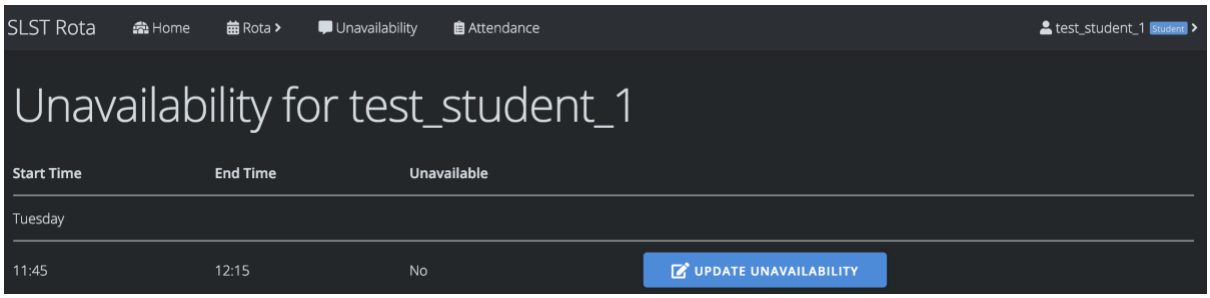
Situation	Expected result	Actual result	Success
Password change – New password & confirmation match, old password is incorrect	Error message that old password does not match the one in the database	Message shown that old password is wrong	√
Password change – New password does not match confirmation, old password correct	Error message that new and confirmation do not match	Message shown that new and confirmation do not match	√
Password change – New password does not match confirmation, old password incorrect	Error message that new and confirmation do not match	Message shown that new and confirmation do not match	√
Password change – New password & confirmation match, old password is correct	Password is changed, user shown message confirming change	Message shown confirming password is changed	√

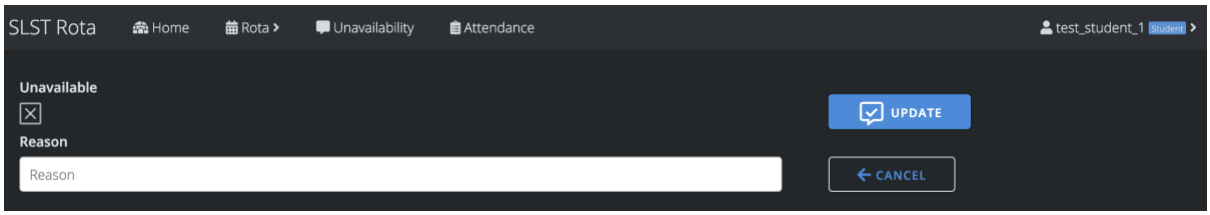


With all this testing completed, I am satisfied that the account control forms are all functioning correctly and displaying the correct messages to the user for whatever situation they are in.

4.1.7. Updating unavailability

With student accounts created and the rota session tested, the next thing to test is the ability for students to update their unavailability for each session in the rota. This needs to be tested before the assignment testing as once a student is assigned they cannot change their unavailability for that session.





4.1.7.1. Extreme data

As the reason input for this form is also a string input, the standard injection testing can be used once again to ensure it isn't susceptible to injection attacks.

Extreme data type	Expected result	Actual result	Success
Text ending with Python string escape character, a backslash '\'	Reason set with '\' present, escaped as needed internally.	User redirected to availability overview page – unavailability set correctly	√

Text ending with SQL statement termination character, a semicolon ';'	Reason set with ';' present, escaped as needed internally.	User redirected to availability overview page – unavailability set correctly	√

4.1.7.2. Bad data

The only bad data possible in this form is for the student to not enter a reason when marking themselves as unavailable. This is handled by checks on the backend and so this needs to be tested.

Situation	Expected result	Actual result	Success
No reason, user set as available	Successful submission, user redirect to overview	User redirected to availability overview page	√
No reason, user set to be unavailable	Error shown to user as reason is required when unavailable	"Please enter a reason for marking yourself as unavailable" shown to user	√

4.1.7.3. Good data

The only situation not tested yet that would be considered a good data condition is when the user sets a reason for their unavailability and then marks themselves as unavailable using the checkbox.

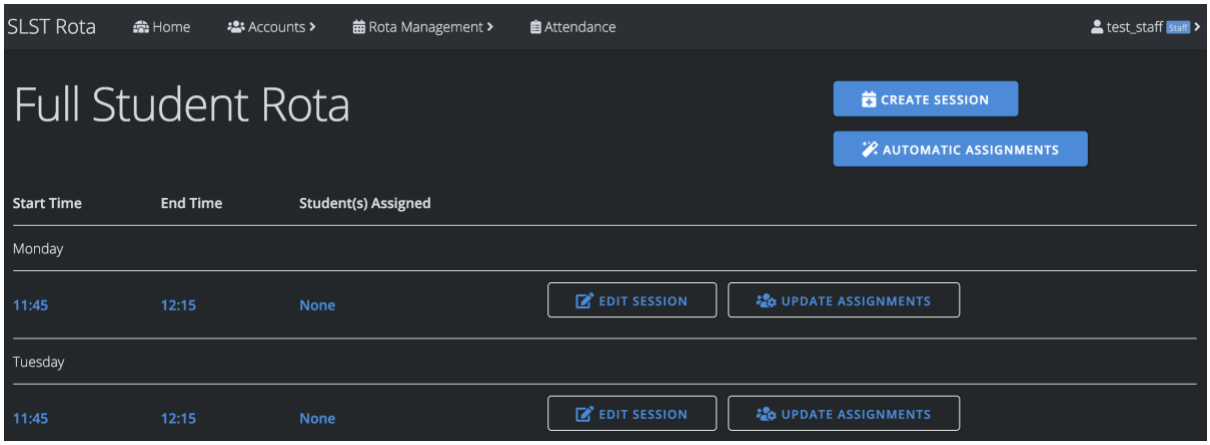
Upon the user setting their reason, ticking the unavailable box and pressing update they are taken back to the unavailability overview page and unavailable for the session in question is now set to "Yes".

4.1.8. Updating assignments

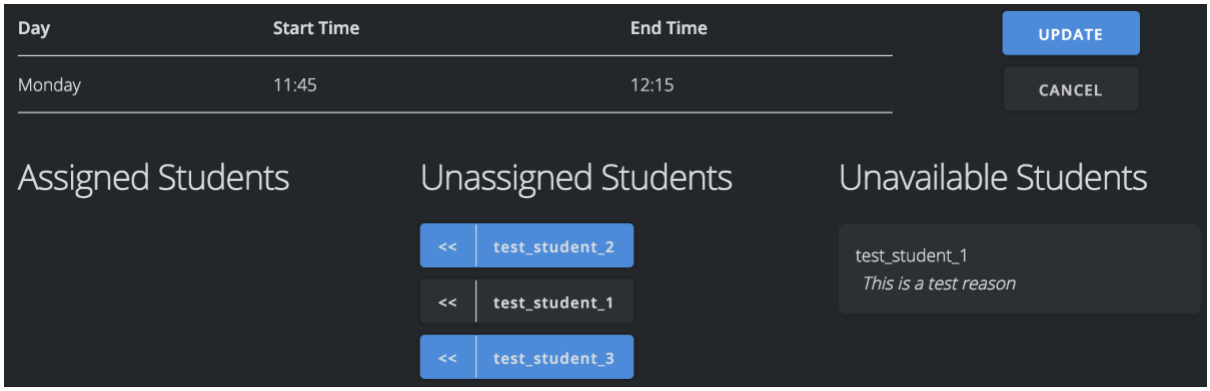
With the unavailability set for the test student account and the sessions created, I am now in a perfect situation to begin testing the system for staff to update student assignments to sessions in the precinct rota.

4.1.8.1. Main page

Upon pressing the update assignments button on the first session in the staff rota view, the staff user is redirected to the correct page for updating the students assigned.



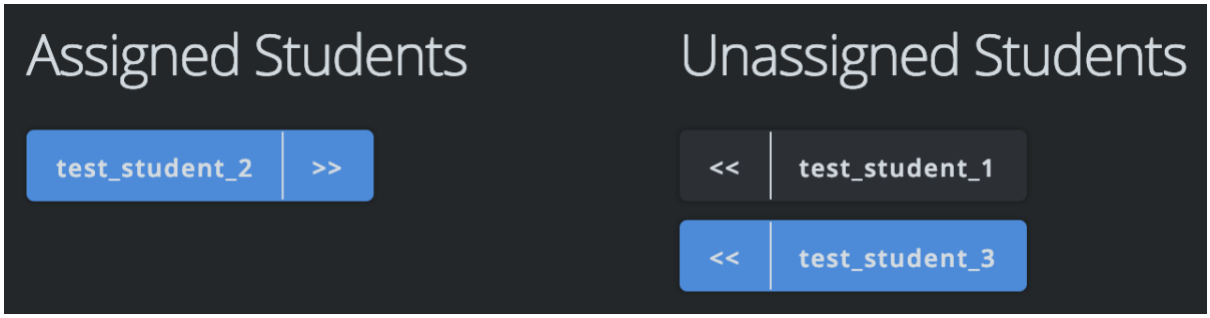
The page correctly shows the session information at the top, along with all active student accounts in the system. It also provides a list for the staff member of students who have marked themselves as unavailable and their reason for this.



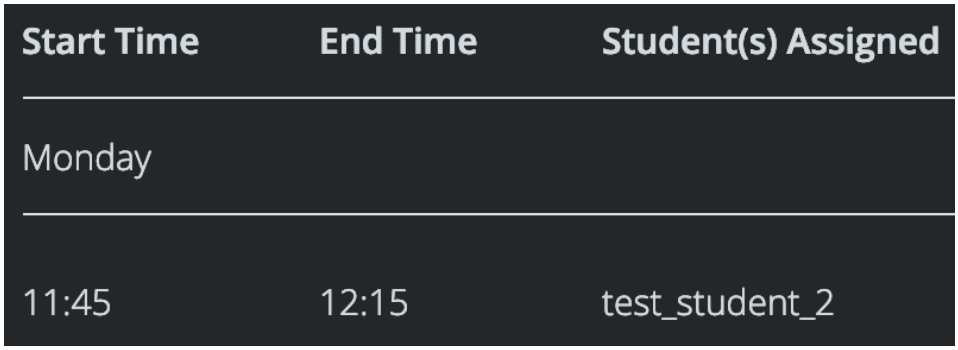
This is a great success as it shows that the system can recognise the unavailability set by the user and display this to the staff member as well as marking the student with the dark grey button instead of the white outlined buttons that indicate the user is available.

4.1.8.2. Setting an assignment

To test that the system and this page in particular can set an assignment correctly, I will click the arrows next to test_student_2 to move that student to the assigned students list and then press the update button to save this in the system.

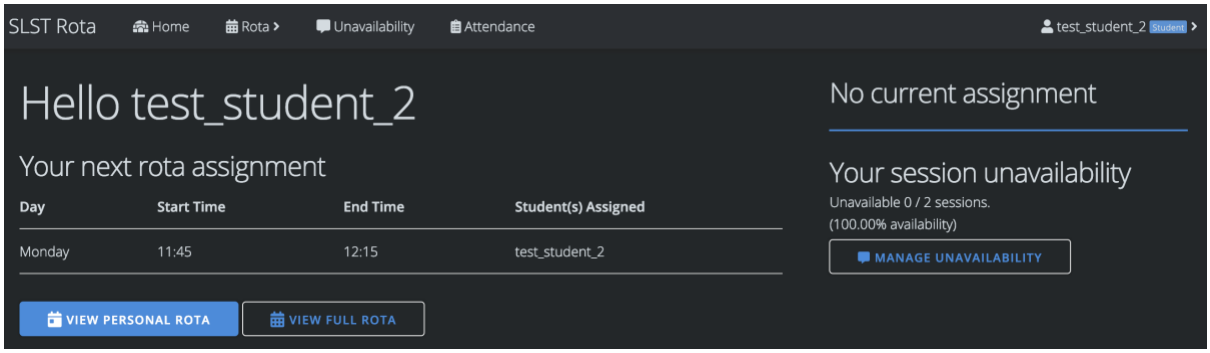


When the update button is pressed, the staff user is redirected back to the rota overview page and in the table, test_student_2 is now listed as assigned for the first session in the rota, the one I have been using for testing.



Start Time	End Time	Student(s) Assigned
Monday		
11:45	12:15	test_student_2

Signing in to the rota system as test_student_2 confirmed that the assignment had been created correctly as this session was immediately displayed to the user as their next session in the rota.



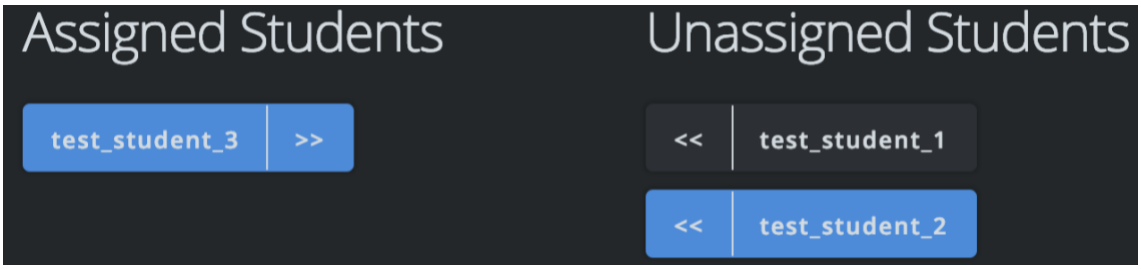
This means the system successfully can update assignments of rota sessions to students.

4.1.8.3. Removing an assignment

It is also important that I test the ability for a staff user to remove a student from an assignment on a session.

To test this, I went back to the assignment edit page and this time moved test_student_2 back to the unassigned list and then instead moved test_student_3 to the assigned list. This is a good test as it tests to ensure that a student can be

unassigned correctly but also that the system can assign a new student at the same time.



Pressing update takes the staff user back to the overview page once again and now only test_student_3 is listed in the table, indicating that the assignment updates were carried out correctly.

Signing in to the student 2 & 3 accounts confirms that test_student_2 has now been unassigned and that test_student_3 was assigned to the session correctly.

The full rota view also confirms that the students are assigned as expected.

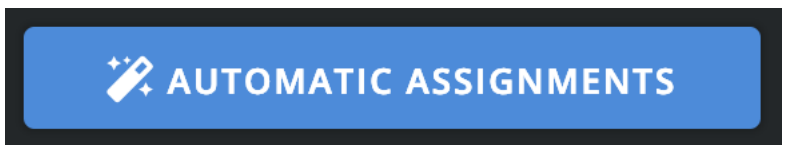
This is a successful test of the assignment editing system that shows it is all functioning as it should. √

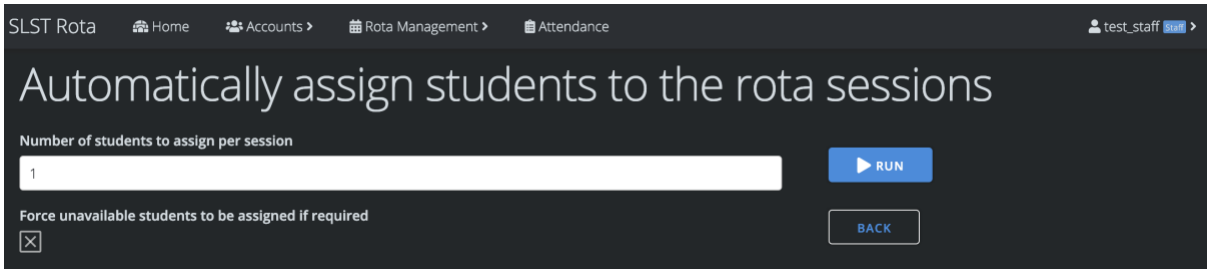
4.1.9. Automatic assignments

The other option for staff users to assign students in the automatic function of the system that made it so computationally solvable as a solution.

To test the automatic assignment functionality, a situation needs to be created where the automatic assignment output can be predicted and tested.

In the test situation, there are three student accounts in the system. The only unavailability set by the students is for the first session where test_student_1 is unavailable. There are four sessions defined in the rota for this test.





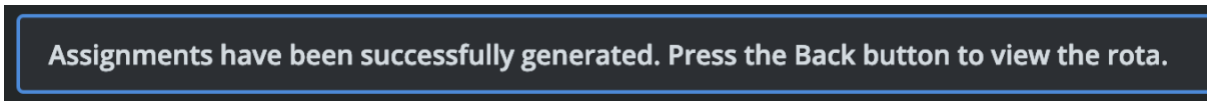
For the first session, test_student_1 is unavailable and so the system will pick the next user with fewest assignments so far which would be both test_student_2 and test_student_3. In this situation, it would then rely on alphabetical sorting to the pick the next user and so test_student_2 should be assigned to the first session.

For the second session, test_student_1 and test_student_3 have the fewest sessions and so following the alphabetical fall-back for this, test_student_1 should be assigned.

The third session should have test_student_3 assigned as they would now be the only student with no assignments.

In the fourth session, all three student users now have one assignment and so it will fall-back to the alphabetical assignment and so test_student_1 should be assigned.

When the run button was pressed on the automatic assignments page, with students set to 1 and force disabled, the staff user was shown a message confirming that the automatic assignments had been carried out and that they could press back to view the rota.



With this, the results of the test could be collected from the full rota view and recorded in the table below, confirming that the automatic assignments feature is working correctly with the resource allocation algorithm.

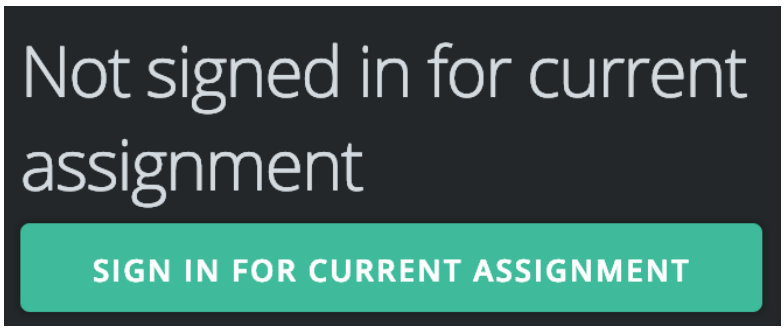
Session	Expected user	Actual user	Success
Monday 11:45	test_student_2	test_student_2	√
Monday 13:30	test_student_1	test_student_1	√
Tuesday 11:45	test_student_3	test_student_3	√
Tuesday 13:30	test_student_1	test_student_1	√

4.1.10.Signing in/out

To test the ability for a student to sign in and out correctly, I will create a test session and assign all three students to it. This will give me three accounts that I can test different scenarios from for signing in and out of the rota system assignments.

4.1.10.1. Signing in

Whilst being logged in as a student on their homepage, once the rota session begins a large sign in button is shown in green for the user. They can then press this to sign in for the session.



Upon pressing the button, the page reloads and the button has then changed to a sign out button, indicating that the user has signed in successfully. √

This test did however highlight a flaw whereby a user cannot sign in early for a session as the button simply isn't shown.

4.1.10.2. Allowing a student to sign in early

To allow a student to sign in early, the current function that finds the next assignment for a student will need to return an extra, new, value stating if the session is soon in addition to the current session flag already returned. The flag will be set to true if the assignment starts in five minutes or fewer.

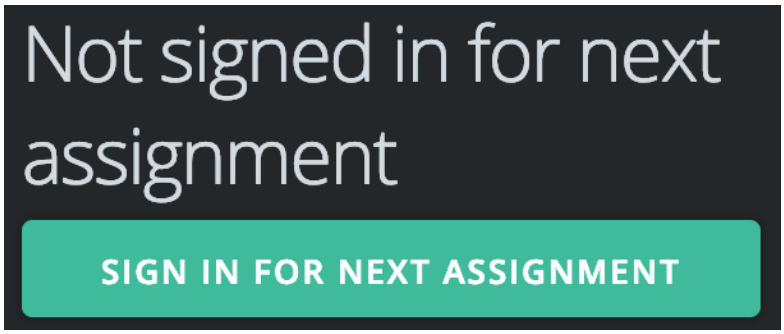
```
# Check if the next session is soon
session_is_soon = False
if not session_is_current \
        and next_session.day == datetime.today().weekday() \
        and next_session.start_time - 5 <= Utils.minutes_now():
    session_is_soon = True

return next_session, session_is_current, session_is_soon
```

Using this new soon flag, the student index template can be updated to show the sign in button if a session is coming up soon for the student logged in.

4.1.10.3. Signing in early

With the extra development done to allow a student to sign in early for a session, this now needs to be tested. To do this I created a new session that is in the near future and assigned a student to it.

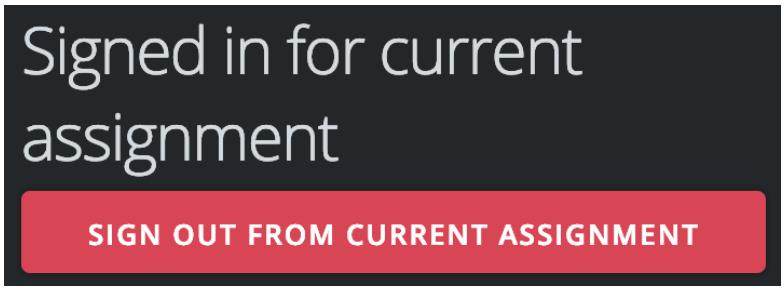
Logging in to the student account with less than five minutes until the session starts, the student is greeted by the sign in button encouraging the user to sign in for their next session.



Much like a normal sign in, pressing the button reloads the page and then displays the sign out button to the user indicating that they have signed in successfully, early in this situation. √

4.1.10.4. Signing out early

Once a student user is signed in for their session, they are immediately shown the sign out button which allows them to sign out from their current rota assignment at any time during the session.



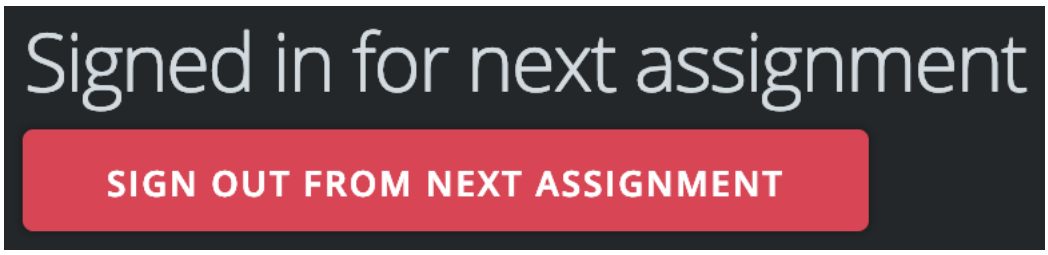
When the user presses the button, the page reloads once again and upon loading, the button is now removed and replaced by text stating that the user has already signed out from their current assignment.

This indicates that the user successfully signed out early. √

4.1.10.5. Signing out automatically

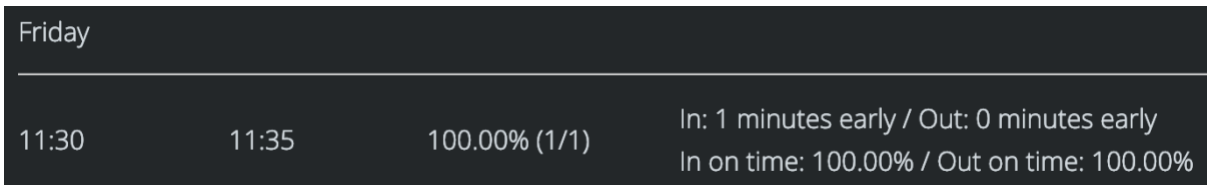
At the end of the assignment session, if the user has not signed out early, the system should automatically sign them out.

To test this, I assigned one of the student accounts to a session and signed in to that account and marked the account as present for the assignment.



I will not press the sign out button and instead wait until the session ends to allow it to automatically sign the account out from the assignment.

With the account signed out as the session has ended, I can log back into the staff account to view the attendance report for the user to confirm that it shows them signing out on time.



This entry in the attendance breakdown for the test_student_1 account shows that the account was automatically signed out at the end of the session. √

4.1.11.Attendance report

To test the full attendance report for both the student and staff accounts I will need to be able to generate a large volume of attendance data that can be processed and

displayed. To do this, instead of manually signing accounts in and out for a week, I will write a test script to generate fake attendance data.

4.1.11.1. Generating test data

To generate the test data for attendance, I will loop over each active student account in the system and from there create new attendance records.

For each student, I can generate a full attendance report for the account which will then allow me to access the assigned session breakdown for that user.

```
students = User.query.filter_by(auth_level=1, disabled=0).all()
# Work over each student in the system
for this_student in students:
    # Get report to find all assignments
    report = StudentReport(this_student.id)
    for assignment in report.breakdown:
```

For each assignment, I can now access the generated array of attendance records for each assignment in the student report. For each one, if the student has not attended it in the system, I can begin the process of creating test data for that attendance record.

```
# Loop over each attendance record in assignment
for att in assignment.attendance:
    # If has attendance already ignore
    if not att[1]:
        # Decide if should have attended (2/3)
        if random.randint(0, 2) != 0:
```

If the random integer check decides that the user should have an attendance record generated, it will create a new database session and then generate a new attendance object from the current assignment session.

The date for the attendance record can then be overwritten so that it appears as though it is from the original date and not the day the data was generated on.

```
# Create attendance from session
session = db_session()
att_entry = Attendance.from_session(this_student.id, assignment.assignment.session)

# Override today's date with attendance date
att_entry.date = att[0]
```

The in time for the attendance record can then be created. This can be shifted with a random integer between -5 and 15 minutes a quarter of the time and will act as the human element of the signing in to simulate some students being early or late. To make it more realistic and so it isn't precisely on a minute, an extra shift between 1 and -1 will be added to the in time, but as a random float instead of an integer.

This can then be saved to the attendance record, as well as the original in time from the session data, which is used for comparing the student in time to the intended in time. This is saved to the attendance record directly to allow for the session times to modified after and not affect the attendance of previous students' records.

```
# Set the in_time, make early/late for 1/4
in_offset = 0
if random.randint(0, 3) == 0:
    in_offset = random.randint(-5, min(15, int(
        assignment.assignment.session.end_time - assignment.assignment.session.start_time)))
in_offset += random.random() * 2 - 1  # float between -1 and 1
att_entry.in_time = Utils.minutes_date(att[0],
                                       assignment.assignment.session.start_time + in_offset)
att_entry.in_time_org = Utils.minutes_date(att[0], assignment.assignment.session.start_time)
```

Similar logic can then be applied to the out time generation, but this time the -1 to 1 random float will not be implemented as most students would be automatically signed out by the system. The quarter chance will still be used however, accounting for the human element of signing out early. This will be between -15 and 0 as a student cannot sign out after the end of the session.

```
# Set the out_time, make early sign out for 1/4
out_offset = 0
if random.randint(0, 3) == 0:
    out_offset = random.randint(-min(15, int(
        assignment.assignment.session.end_time - assignment.assignment.session.start_time)), 0)
att_entry.out_time = Utils.minutes_date(att[0],
                                        assignment.assignment.session.end_time + out_offset)
att_entry.out_time_org = Utils.minutes_date(att[0], assignment.assignment.session.end_time)
```

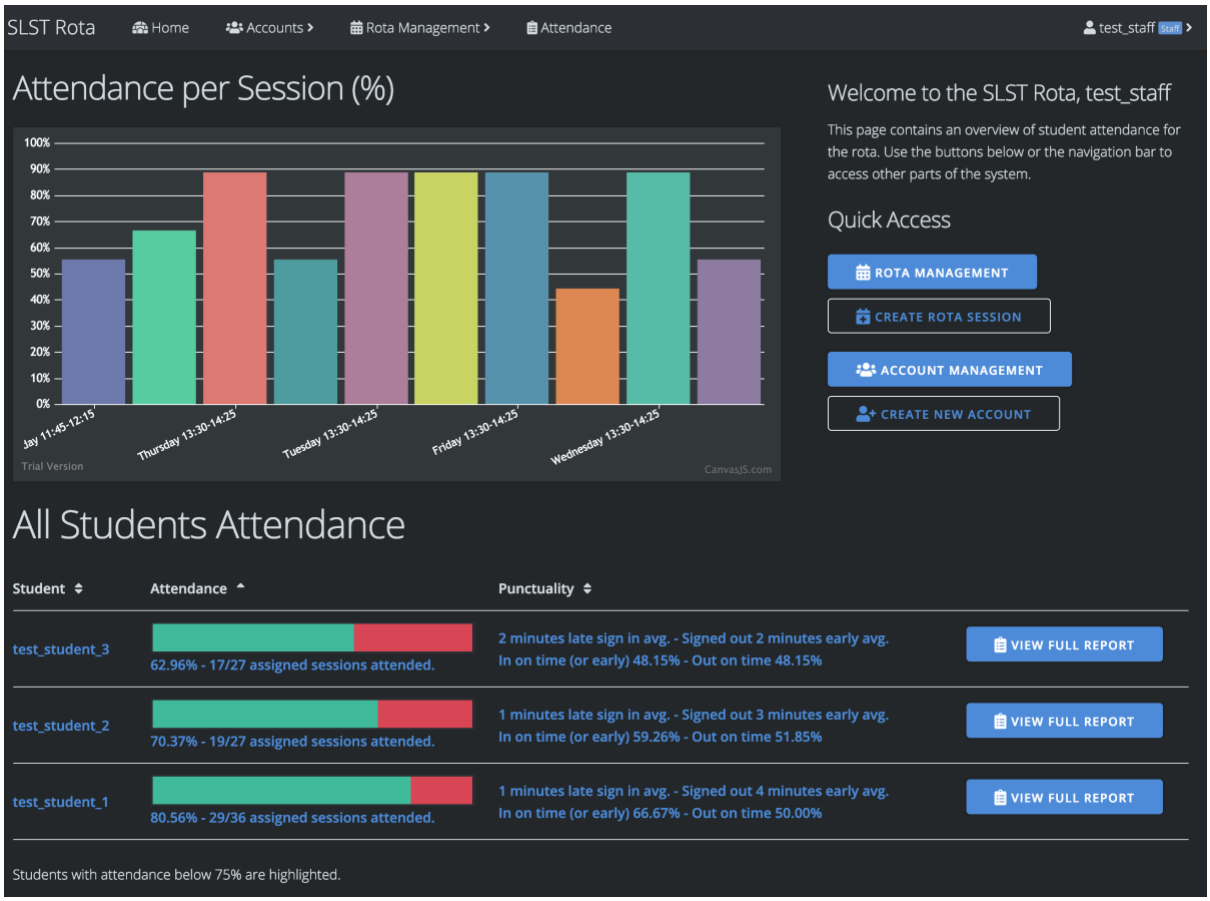
This fully generated record can now be saved to the database for the system.

```
# Save
session.add(att_entry)
session.commit()
```

4.1.11.2. Testing test data generation

With the test data generation route written, I can now test it by automatically assigning all the students in the system to the rota layout defined. This however would mean there is no time for any historical attendance records and so to allow me to test, I modified the database directly to shift back the creation timestamp of each assignment to a week ago.

Heading to the attendance test data generation route immediately redirected me back to the attendance homepage which is a good sign, nothing errored. Once the page had loaded, it was evident that the system had correctly generated all the test data as the overview graph had been populated as had the student overview table below.



This shows me that the attendance report overview page is functioning as intended, as is the test data generation. √

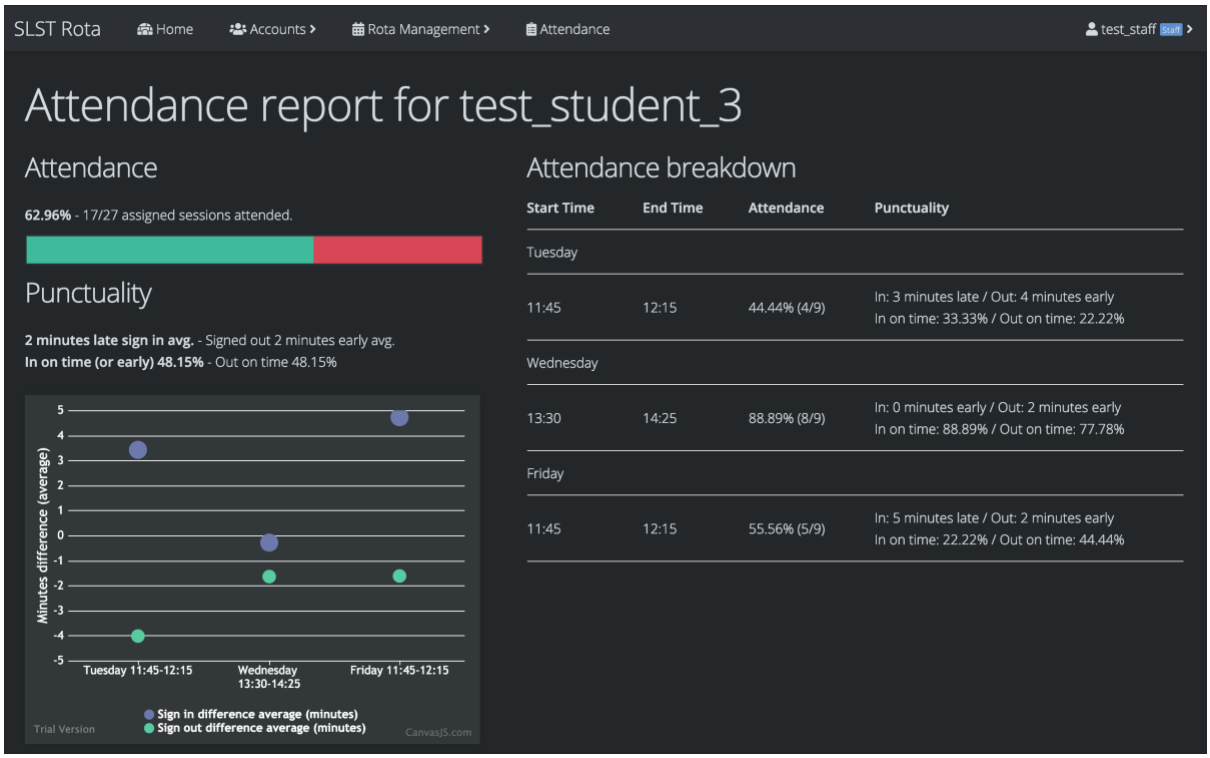
4.1.11.3. Student attendance report

Pressing the “View Full Report” button for the test_student_3 account allows me to ensure that the student attendance report page is also functioning correctly now that there is attendance data populated in the system.

The button correctly takes me, the staff user, to the student breakdown. On the page the attendance bar graph is shown as a quick indicator of how many sessions the student has missed versus how many they have attended.

Below that is the punctuality scatter graph which is also correctly showing the average early/late offset for the student signing in and out of each session they attended.

On the right, the main content of the page is the attendance breakdown table for the student. This provides more information about the attendance of the student for each session in the rota that they are assigned to. This allows for the staff user to isolate specific sessions where the student may always be late or absent for example.



The page rendered all the data correctly, with the graphs both showing the data they should be showing and the breakdown table making the attendance per session obvious to the staff user.

This is a successful test of the student attendance report page. √

4.2. Stakeholder review

With my own full testing completed of the system, I was happy to give it to my select stakeholders so that they could both provide further feedback on the system and find any flaws that could be improved upon.

To ensure an unbiased test with good feedback, I simply gave the system to both Nick and Tom, asking them to try it out and give me feedback on it. I gave no pointers on how to best use the system or any instructions on it.

4.2.1. Nick Balaam

Nick is the staff member that acts as the team leader for the stage team and the stage precinct. His feedback, which I cut down to the crucial issues, was as follows:

When using the system, and setting up the rota layout with all the rota sessions, there appeared to be no checks for overlapping sessions as I was able to create one session from 11am until 1pm and then a second session from 12pm until 2pm on the same day. When I then went to assign my students to it, it allowed me to assign the same student to both.

This is a flaw that I need to investigate and resolve as overlapping sessions should not be permitted in the system. The rota is designed to operate only for the stage precinct and so no two sessions should ever overlap.

When I was creating the accounts for the students but also the other staff who I wanted to give access to, it showed a little number next to the "authentication level" badge that indicates if the account is a student or staff member. This number seems to have no relevance at all and I just found it pointless and confusing to have there.

The auth level number is something that can easily be removed and I will do this to provide a cleaner user experience for those using the rota system.

Once the students had been using the system for a while to test it as you had requested we do, I started looking at the attendance reports that your system provides. They are amazing as they show to me exactly who has been slacking (none of the boys in this case) but also who has always been on time or even early in some cases. With this though, a small thing with the punctuality graph annoyed me – If a student signs in early for a session, this is shown in a blob that is below zero on the graph. However, if a student signs out early (this would be a bad thing) it is also shown as a negative blob on the graph which was a bit confusing.

The simple fix for this to improve the user experience is to invert the singing out offset as it can never be above zero currently, so that signing in late and out early are

show on the same half of the punctuality graph, making it more obvious at a glance how the student is doing.

However, Matt, the system you have produced is ultimately a very useful program that the stage team can certainly look at using going forward to make organising the precinct easier as the team continues to grow in size. I thank you for all your efforts in making this and with your continued work on the stage helping events.

4.2.1.1. Fixing the issues raised

With the feedback from Nick gathered, I could now work on resolving these issues to ensure that my final product is the best that it can be.

4.2.1.1.1. Overlapping sessions

The first issue I need to address is the ability for sessions to be created or edited that overlap existing ones. To do this, I need to create a check function that can be run in the server side validation to ensure that the new or modified session does not clash.

This function should take in the new session and iterate over all other sessions in the database. It then needs to see if the new session's start time sits within an existing session, whether the new end time sits within an existing session or if the new session completely encompasses an old session. The function can then return a Boolean to indicate if the new session is okay and it can also return the overlapped session if there is an issue so the user can be given information about which session is causing a clash.

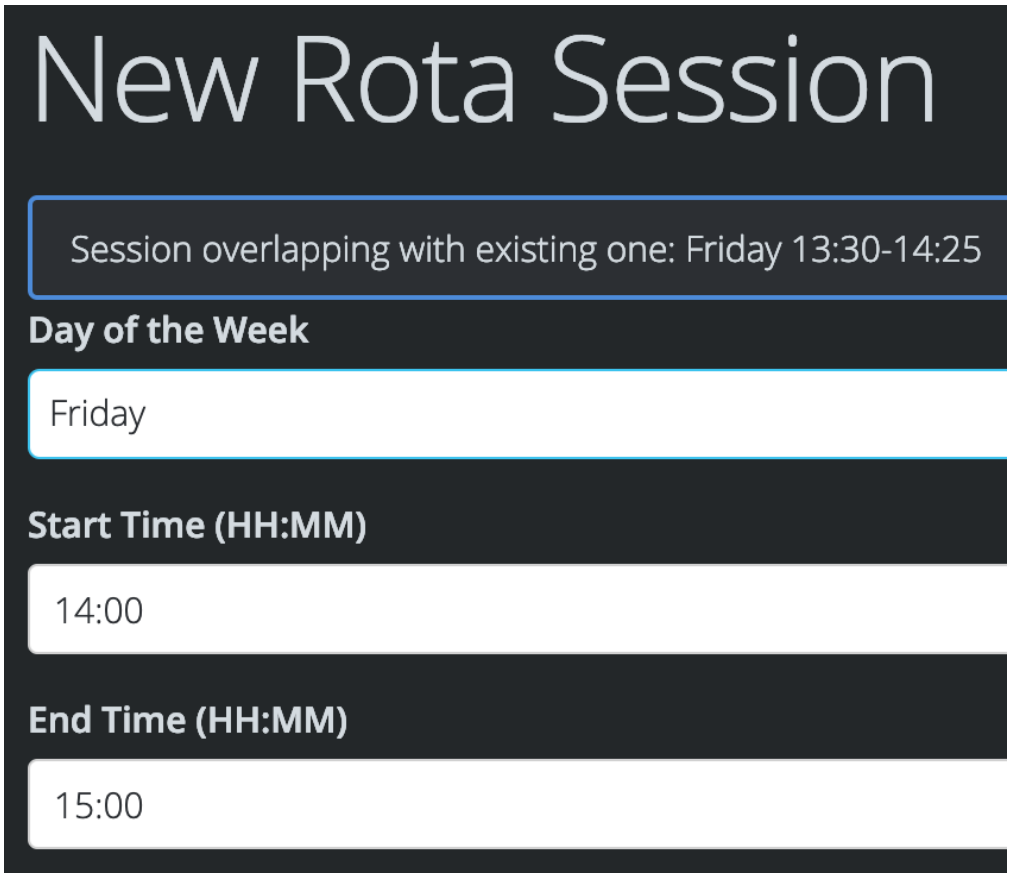
```
# Check if a new session overlaps an existing one
def check_session_overlap(session) -> Tuple[bool, Union[Session, None]]:
    sessions = Session.query.filter_by(day=session.day).order_by(Session.day.asc(), Session.start_time.asc()).all()
    for sess in sessions:
        # Skip current
        if session.id == sess.id:
            continue
        # Check start overlap
        if sess.start_time < session.start_time < sess.end_time:
            return False, sess
        # Check end overlap
        if sess.start_time < session.end_time < sess.end_time:
            return False, sess
        # Check in between
        if sess.start_time > session.start_time and sess.end_time < session.end_time:
            return False, sess
    return True, None
```

With the check function written, I could now implement it as a validation routine in the server side route for the adding of a new session. I will do the new session first instead of editing as there is no chance of the session clashing with itself.

With the check implemented I could then test the functionality in the system, creating a new session and setting its start time so that it would intersect with an existing session.

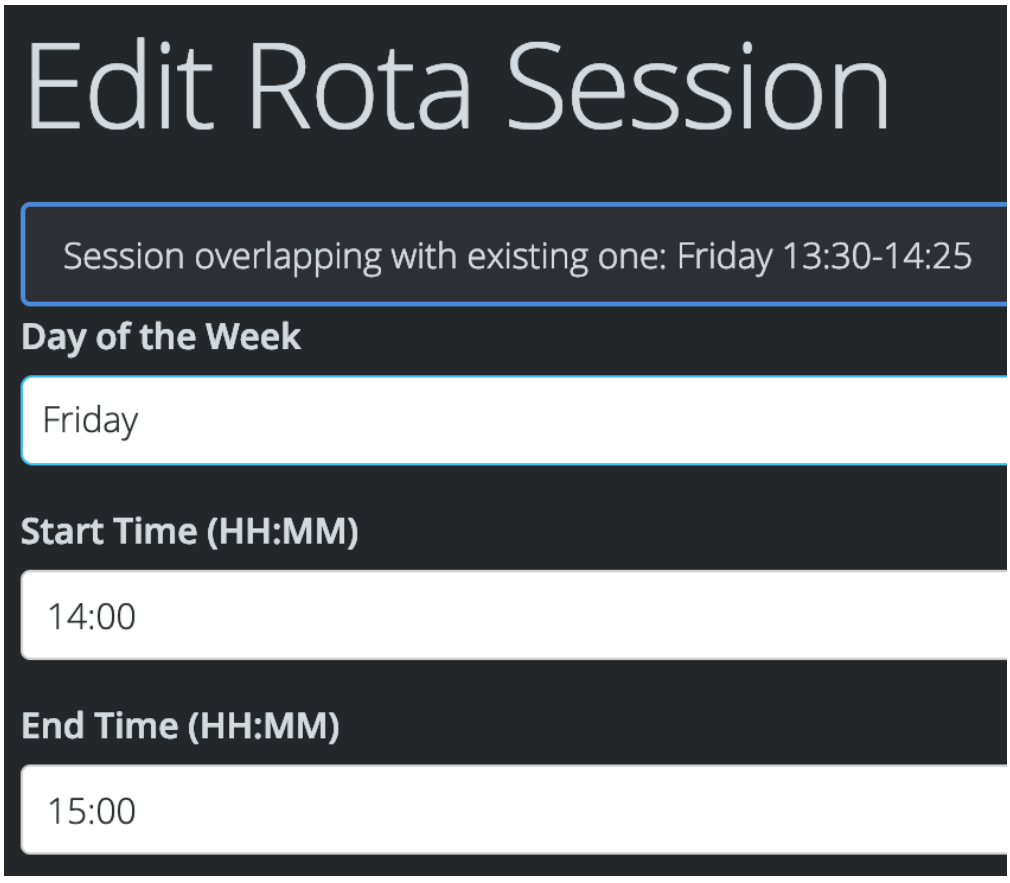
```
session = Session(form.day.data, start_time, end_time)

check = check_session_overlap(session)
if check[0]:
    dbsession = db_session()
    dbsession.add(session)
    dbsession.commit()
    return redirect(url_for('staff.rota'))

else:
    flash('Session overlapping with existing one: '
          '{0.day_frmt} {0.start_time_frmt}-{0.end_time_frmt}'.format(check[1]))
```

Upon pressing the create button the system ran the check correctly and returned the error to the user on the system website so that they can see the information about the session they have clashed with.



I can now implement this into the edit session route to ensure that a session also cannot be edited into a situation where it clashes with an existing one in the system. The code is almost the same but I need to ensure in the event of the check failing, the database session transaction is rolled back so that the bad changes are not committed to the system's database.

```
check = check_session_overlap(session)
if check[0]:
    dbsession.commit()
    return redirect(url_for('staff.rota'))

else:
    dbsession.rollback()
    flash('Session overlapping with existing one: '
          '{0.day_frmt} {0.start_time_frmt}-{0.end_time_frmt}'.format(check[1]))
```



With this done, I am happy that the system will no longer permit staff users to create overlapping sessions in the system.

4.2.1.1.2. Removing the auth level number

The change to remove the auth level number that is displayed on the accounts overview table for staff members is an easy and quick one to complete. In the Flask route that generates the data for this page, I simply needed to modify the format string for the auth level label in the table row, removing the number representation and just leaving the auth level label.

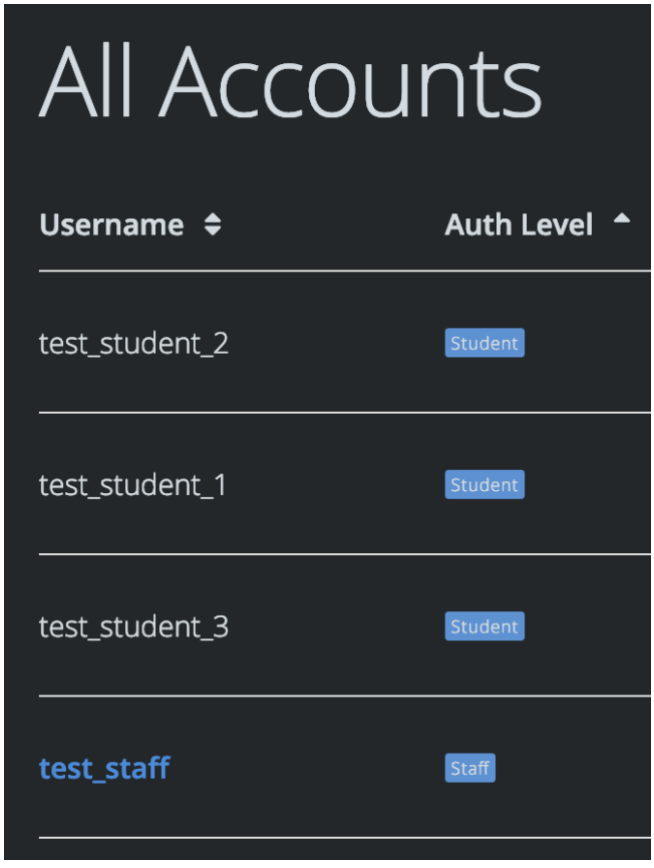
Before:

```
["{} ({})".format(item.auth_label, item.auth_level), item.auth_level],
```

After:

```
[item.auth_label, item.auth_level],
```

The second value in the list here is the value that the table will be able to be sorted by, which will still be the numeric representation of the auth level, but this will no longer be displayed to the user. Instead, only the authentication level label will be shown.



4.2.1.1.3. Fixing punctuality graph

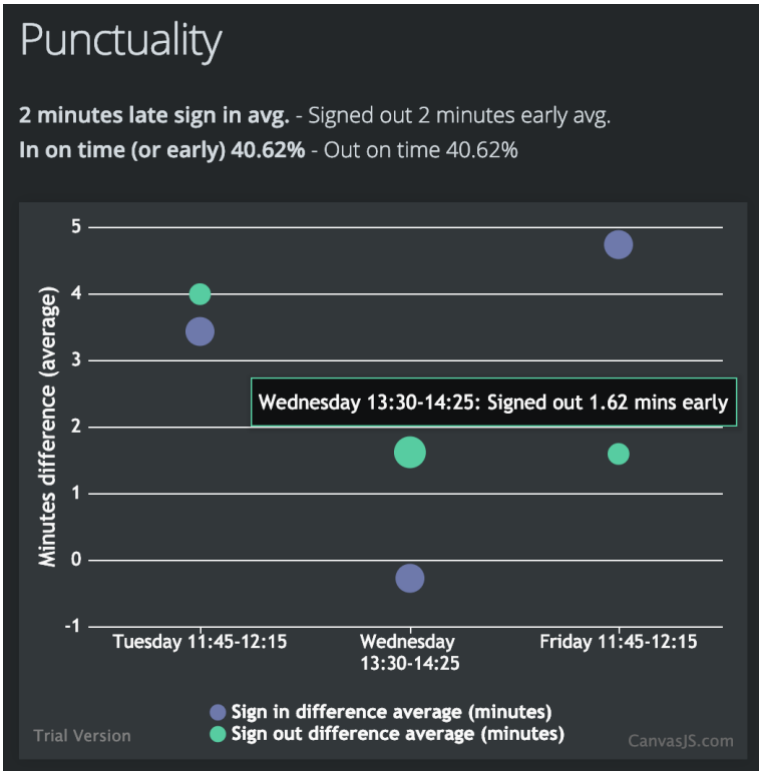
To fix the issue with the punctuality graph where the signing out early values are on the wrong “side” of the graph, I just need to go to where the values are generated for the graph and simply invert every value for signing out.

```
for assignment in report.breakdown:
    label = "{0.day_frmt} {0.start_time_frmt}-{0.end_time_frmt}".format(assignment.assignment.session)
    # In time diff
    graph[0]["dataPoints"].append({"label": label, "y": assignment.in_diff_avg, "markerSize": 20})
    # Out time diff
    graph[1]["dataPoints"].append({"label": label, "y": assignment.out_diff_avg, "markerSize": 15})
```

There are the lines that generated the markers on the graph, so I just need to add a minus sign in front of the out_diff_avg value to invert it, which should mean that the markers for singing out early are then shown on the same side of the graph as signing in late.

To further improve the usability of the graph, I decided to add custom tooltips to each marker which will further explain the data when hovered over.

```
for assignment in report.breakdown:
    label = "{0.day_frmt} {0.start_time_frmt}-{0.end_time_frmt}".format(assignment.assignment.session)
    # In time diff
    graph[0]["dataPoints"].append({"label": label, "y": assignment.in_diff_avg, "markerSize": 20,
                                   "toolTipContent": "{{label}}: Signed in {:,.2f} mins {}".format(
                                       abs(assignment.in_diff_avg),
                                       "late" if assignment.in_diff_avg > 0 else "early")})
    # Out time diff
    graph[1]["dataPoints"].append({"label": label, "y": -assignment.out_diff_avg, "markerSize": 15,
                                   "toolTipContent": "{{label}}: Signed out {:,.2f} mins {}".format(
                                       abs(assignment.out_diff_avg),
                                       "late" if assignment.out_diff_avg > 0 else "early")})
```



4.2.2. Tom Brownridge

With all the changes completed based on the feedback of Nick, I could now more onto the review provided by Tom who is one of the current precinct monitors for the stage.

I really like this precinct system for us to use. It makes organising when we are here so much easier as we can just check the rota to see who is meant to be. It also means that Mr Balaam knows exactly who was here doing their job and when because we all sign in and out for the precinct duties.

The system was embraced well by Tom and the other precinct monitors who got stuck straight in with making use of it during the test period, giving it a full run to ensure there were no flaws with it and that it did function as intended.

However, the white table outlines and the white buttons are really quite bright compared to the background of the site which is the nice dark grey. It would be great is these could be fixed so they don't hurt my eyes so much. We are, after all, a stage team who are used to the dark.

This is something that can be easily fixed with some styling changes and should improve the use experience for those one the system's website, giving a more consistent dark appearance throughout the site.

Also, the table headings are very hard to notice compared with all the other content in the full rota view, it would be great if these had some sort of background maybe so that they were more noticeable as divisions in the table.

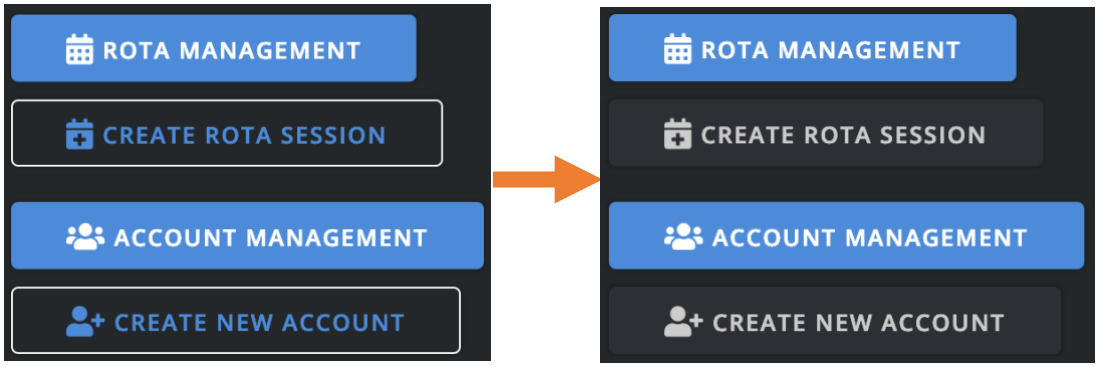
Again, this is something that should be able to be addressed with some simple modifications to the site styling and a small change to the table rendering script to allow this to be applied correctly.

Overall though the system is really great for us all to use when doing our precinct duties with the simple rota and buttons we can press to sign in at the start. The automatic sign out system and the way we can view our own attendance records is very useful as it means we don't have to worry at the end of a duty and we can keep track of our own performance to ensure we are doing well. It is a very good system.

4.2.2.1. Fixing issues mentioned

4.2.2.1.1. White outline button styling

To do this, I first tackled the issue of the buttons, simply replacing the current secondary button styling with a flatter design that was more consistent with the primary button design.

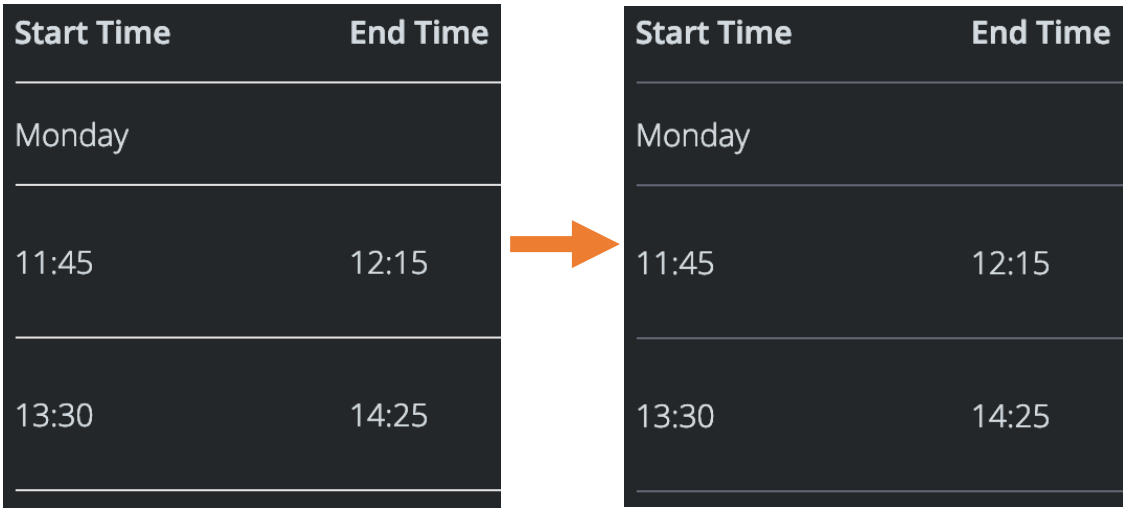


4.2.2.1.2. White table borders

The other issue raised that needs to be addressed is the current design of the table borders that are currently bright white, which understandably may hurt users eyes on the dark background of the site.

```
table {
  tr {
    th, td {
      border-color: #667;
```

This can be fixed with a simple bit of styling to change the border colour of table rows and table headings.



4.2.2.1.3. Table subheading backgrounds

The last issue raised by Tom was that the headings in the tables didn't have a background are so were hard to notice over the other content in the tables.

```
&.subheading {
  background: lighten($black, 10%);

  th, td {
    &:first-of-type {
      padding-left: 12px;
    }
  }
}
```

To fix this, I can create a new subheading class for table rows that will have a background colour set for it.

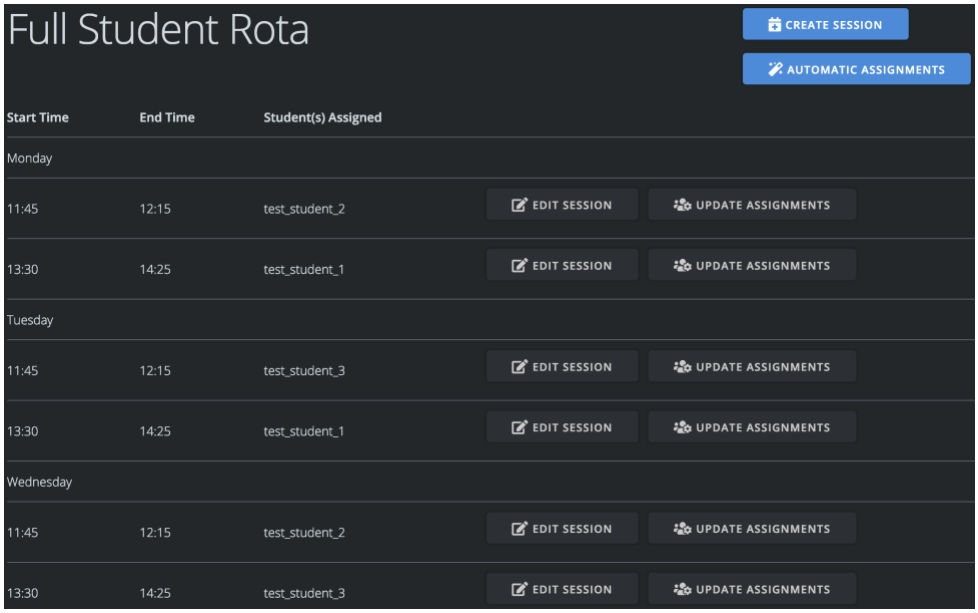
I will also add some padding to the first element in the row to ensure that the text in it has sufficient padding with the background.

To apply this new class to the table, I will need to update the table macro to accept a new value that will trigger the subheading class to be applied. I will use a new third element in each call to the macro which will be a Boolean value to trigger the class being added.

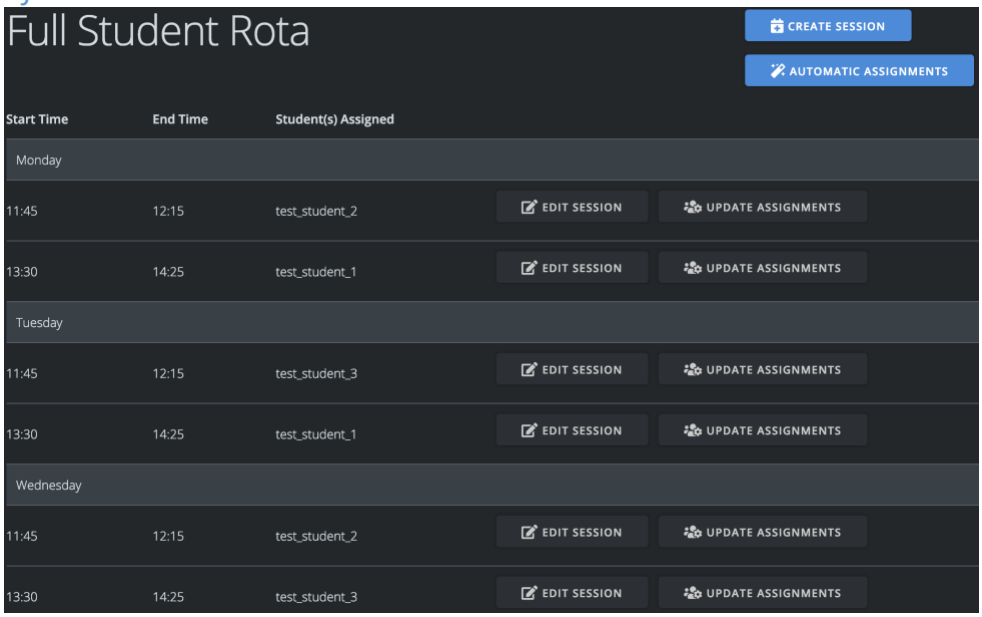
```
<tr class="{% if item[0] %}highlight {% endif %}{% if item[2] %}subheading {% endif %}">
```

With this change done I could now head to the full rota view and pass in the new extra Boolean value for the day headings which when rendered on the table look amazing.

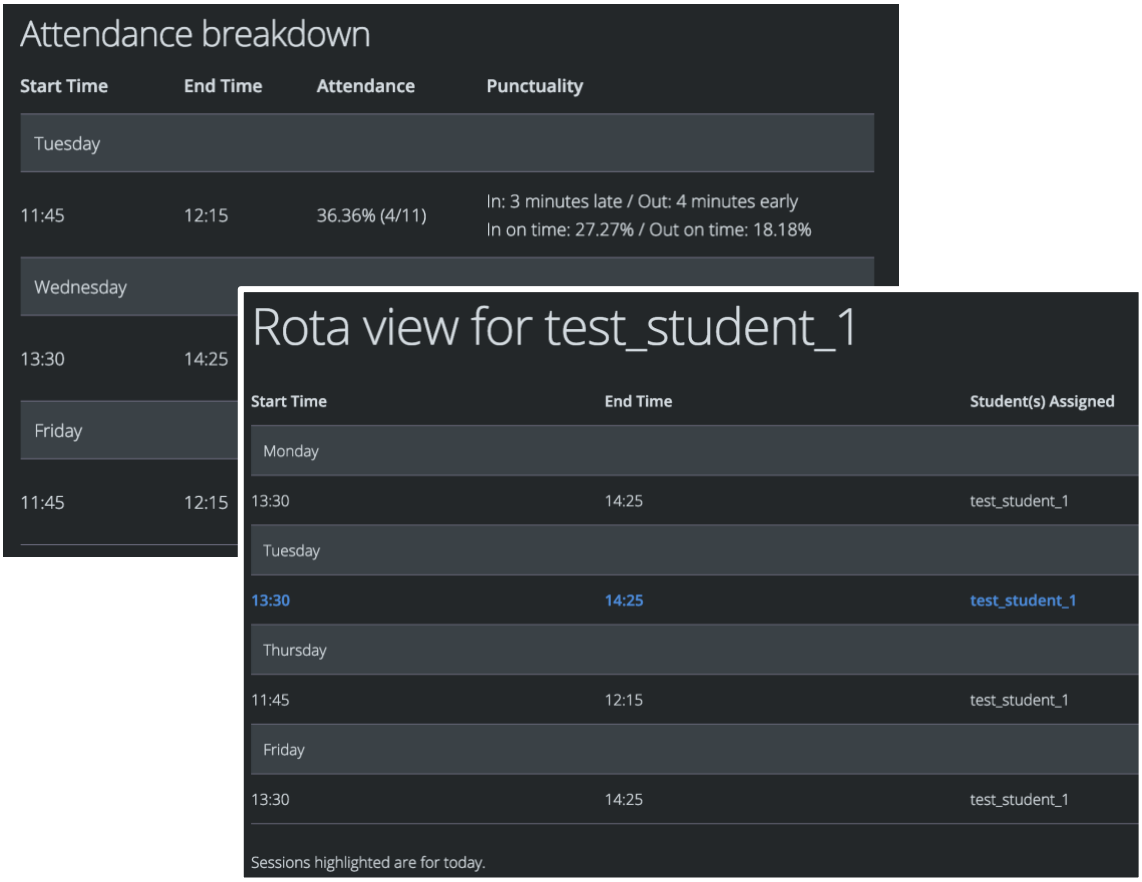
Before:



After:



This change can also be applied to every other table where subheadings are used.



With these changes completed I was happy that the buttons and tables were now much more aligned to the overall colour scheme of the site, being much easier on any eyes.

4.3. Evaluation of the solution

Working through the initial list of objectives for the system will provide a good way for me to evaluate how well the final solution performs in reference to what the intention of the program was before the development had been done.

I will work through each initial objective and provide comments/evidence on how well that objective was achieved in the final solution which should provide a good evaluation of the program.

4.3.1. Provide an easy to use rota system for students on duty in the precinct

The program provides a clean and simple to understand rota table preview for all students that log into the system as well as provide an intuitive user interface when they need to interact with the system to sign in and out from assigned duties as well as viewing their own attendance.

I can determine that this objective has been met successfully based on the stakeholder review from Tom where he indicated that he was satisfied with the final system once the minor colour scheme changes to some elements were made.

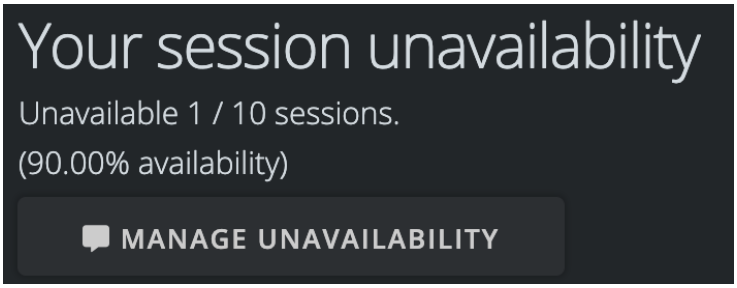
4.3.2. Provide an easy management system of the rota system for staff

The staff panel on the system provides staff with a clear view of the rota layout as well as control over the students and their assignments to each rota session. Further, the website provides clear but in-depth analytics about the performance of each student in the precinct allowing for easy management by the staff of the precinct and its rota.

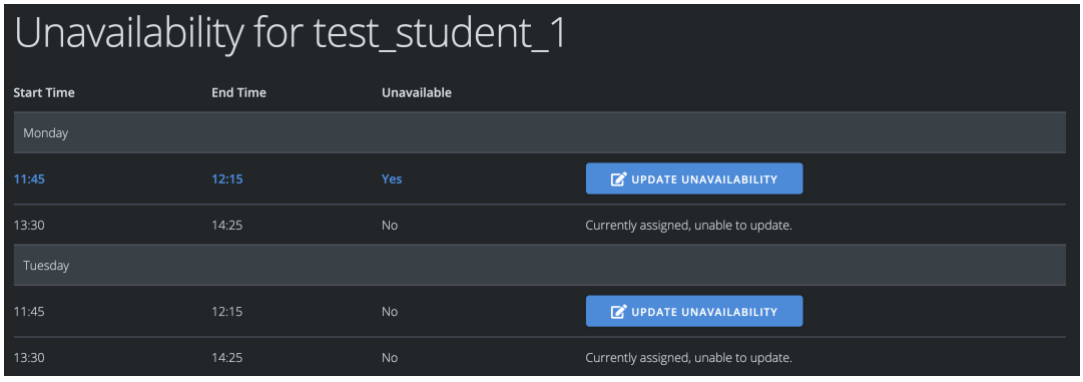
Based on the stakeholder review from Nick Balaam, I can confirm that the software solution for the precinct rota met this objective as the comments provided indicated that Nick found the system easy to use and with the features it needed to manage the precinct effectively.

4.3.3. Allow students to indicate days they cannot complete precinct duty

As a student logged into the system, the option to manage when they are unavailable is provided directly on the homepage. This ensures that students can easily indicate to staff when they are unable to monitor a precinct session.



Pressing the manage unavailability button takes the student directly to their unavailability overview page where they are provided with a breakdown of every rota session that is active in the system, allowing them to update their unavailability for each one.



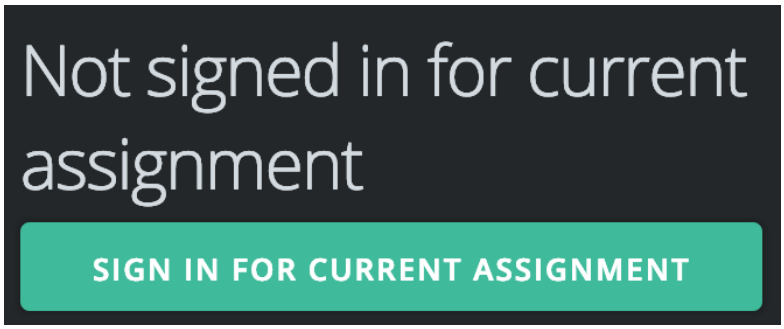
The update unavailability button takes the student to a simple form page where they can mark themselves as unavailable and provide a reason for their unavailability so that it is clearly communicated with the staff members on the system.

Based on these features, I am happy that the objective of allowing the student to indicate their unavailability has been met with success.

4.3.4. Allow students to sign in and out of their current precinct duty and for this to be tracked

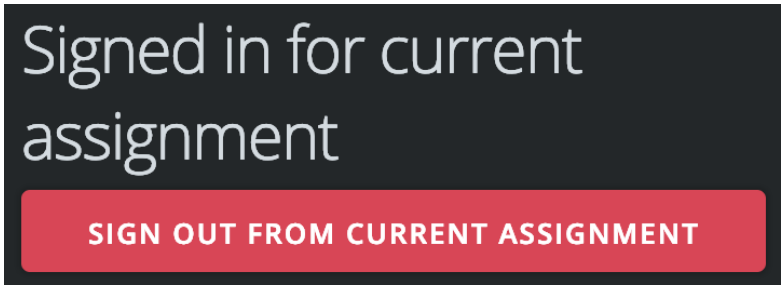
The key feature of the system is that students can sign in and out for the duties, allowing the system to effectively track their attendance and provide this in reports.

When a student is within five minutes of an assigned session starting, they will be shown a sign in button. This will continue to show on their homepage until the student presses it or the session ends.



Once the button is pressed, the student is marked in the system as attending the session. This is recorded in the database so that staff can view the attendance report of every student.

Once the student is signed in, the button is replaced with one that allows them to sign out of their session early if they need to. This is also displayed in the same location on their homepage as where the sign in button was.



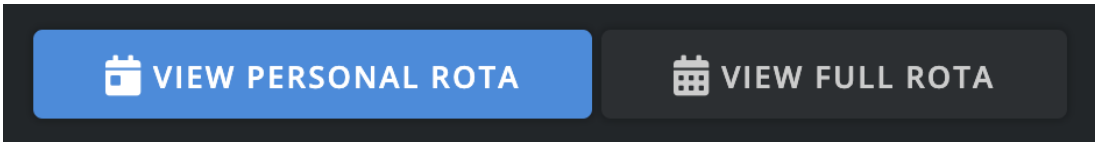
Pressing the button allows the student to sign out from their session before it ends, which is also recorded in the database and displayed in the student attendance reports as leaving early.

If the student does not press the button, they are automatically signed out at the end of the session and this is also recorded into the database for the system.

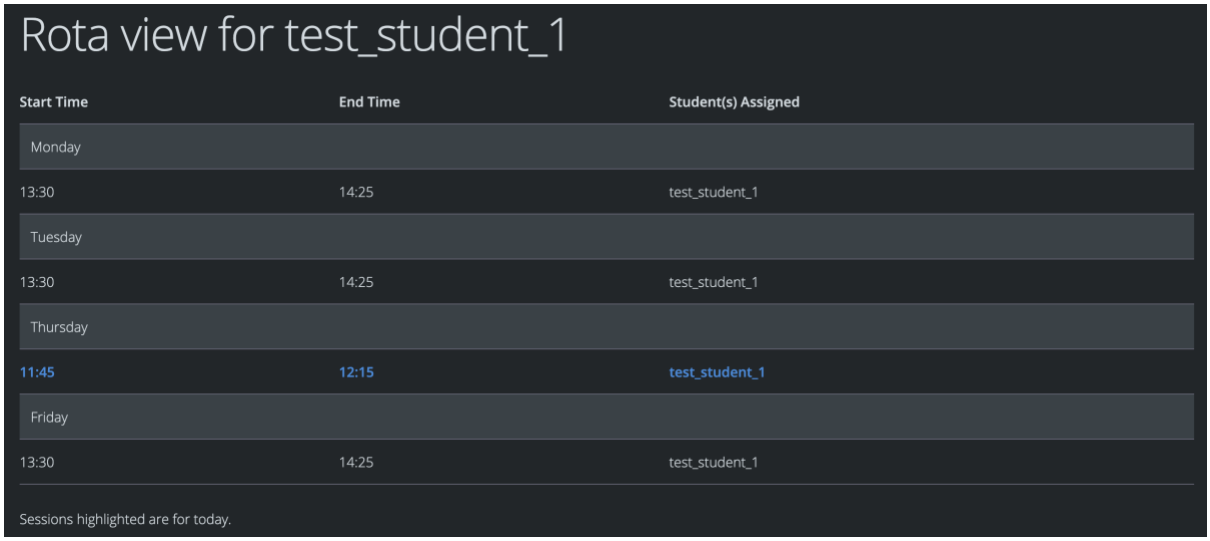
The system provides a clear and simple way for students to sign themselves in and out of assigned sessions, meeting the objective effectively.

4.3.5. Display an easy to understand timetable of duties for the precinct

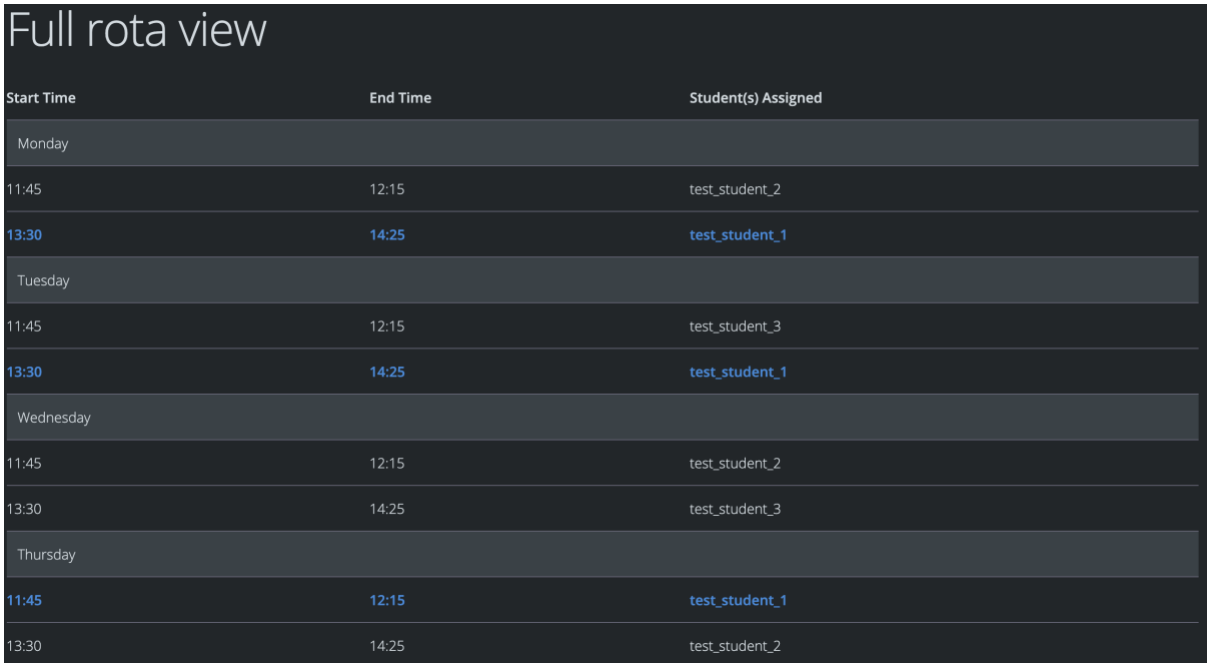
The student portal provides easy to access to two different rota views for the precinct, both providing important information to the user about the duties.



The first is the personal rota view, which as the name suggests, provides the user with a simple rota that is personalised to only the sessions they are assigned to. This is generally the best view for the student to use as it provides a clear breakdown of the start and end time for each session they are assigned to.



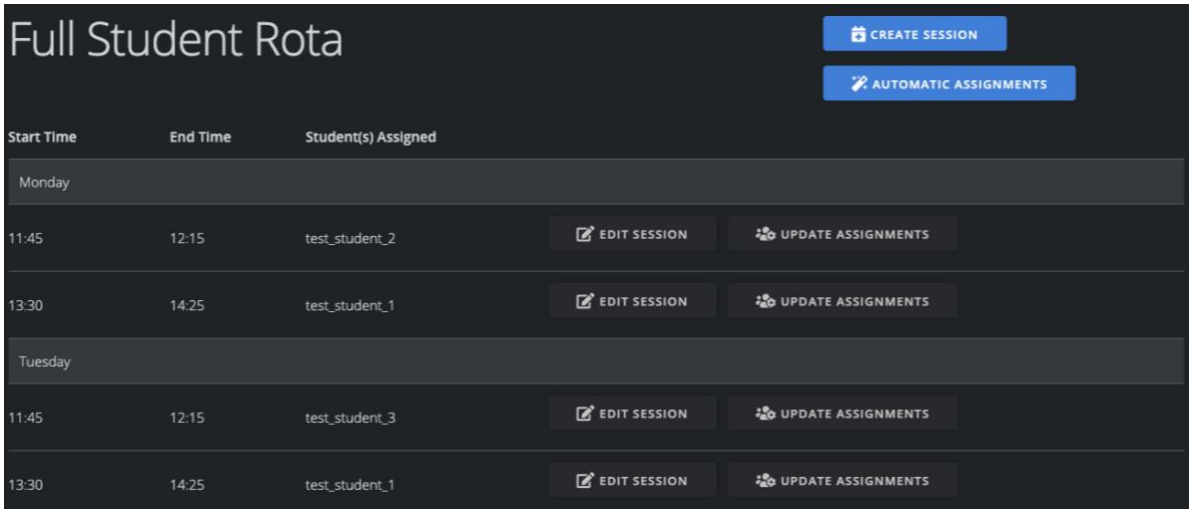
I feel this meets my initial objective well of giving an easy to understand rota for the student. However, in addition to this view, the system also provides the student with a further full rota view to ensure that all information they could need for the precinct is available to them.



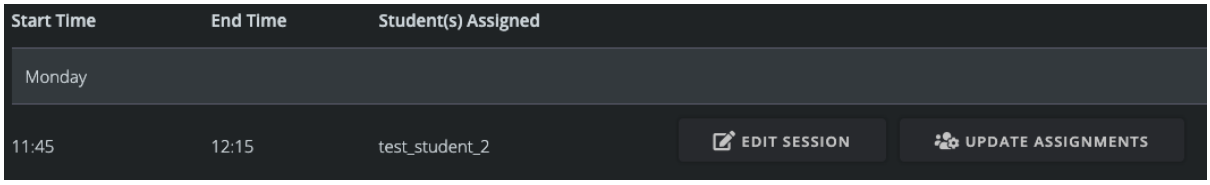
With these two rota views providing all the information available in the system about the precinct timetable, I believe this objective has been successfully met by my computational solution to the problem.

4.3.6. Allow staff to define the precinct duty sessions for the week

I believe this objective is competently met by the software solution as it provides an advanced table interface that allows staff to view the current sessions created in the system whilst also being able to create new ones and edit existing ones.



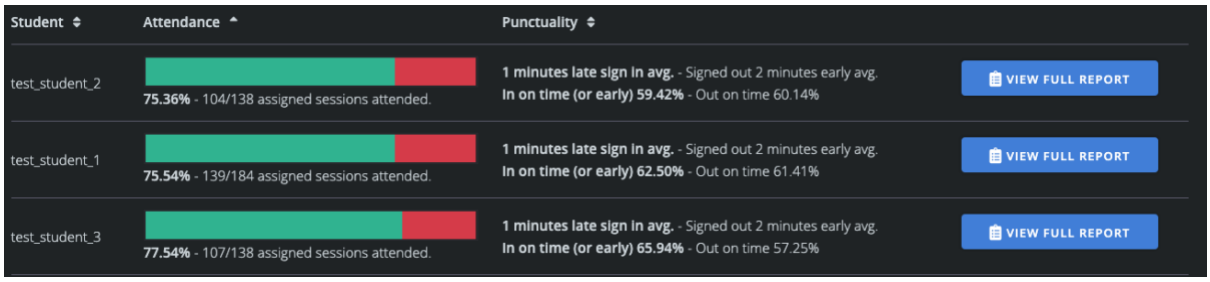
Further, this solution also seamlessly integrates the buttons that allow the staff to control the students that are assigned to the session alongside providing the key information about the currently assigned students for each rota session displayed.



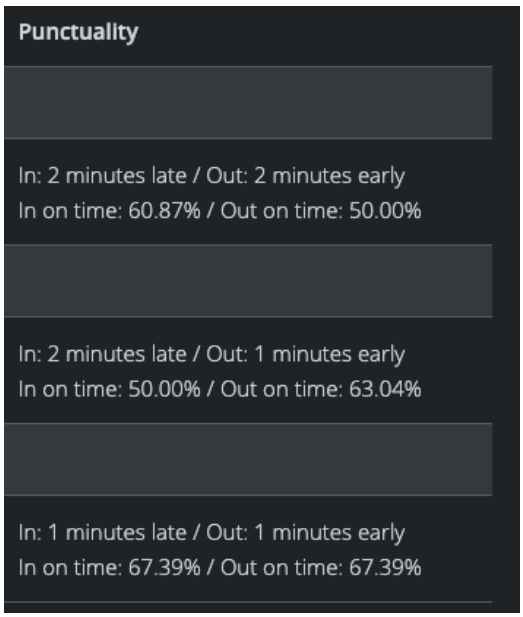
Overall, this objective has been achieved by the solution created.

4.3.7. Allow staff to view when students signed in and out

From the staff homepage, they are shown an overview table of all students in the system and their attendance information. Part of this information is the average sign in/out times and the punctuality ratings for each one.



Further, when the staff member clicks on the button provided to view the full report for a student, a better breakdown is show to the staff. This provides the same key information about the signing in and out punctuality but broken down into a per session table.



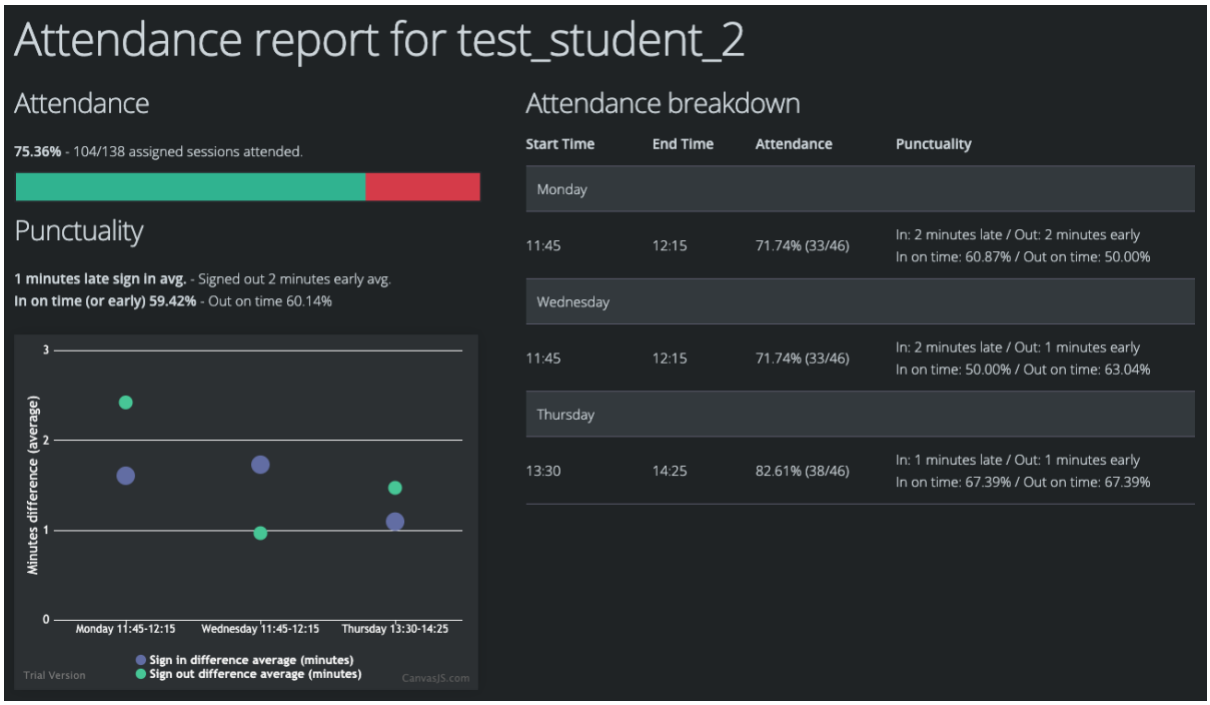
If I were to work on this section again, I think it would very beneficial to have a system in place where-by the staff member could view the raw sign in/out timings for every single attendance record of that student.

This would provide even more detail to the staff member about anomalous days that throw off the student's average punctuality.

However, I do believe that the current system still manages to meet this objective as the system does provide the key information on each student's signing in/out punctuality.

4.3.8. Allow staff to view in-depth analytics regarding student attendance for their duties

As mentioned in the previous objective evaluation, staff can access dedicated reports on the attendance of each student. Within these reports, analytics are provided highlighting to the staff member sessions where the student may not be performing as well, whilst also providing key information on the overall attendance of the student.



This page provides two bits of key useful data on the attendance performance of the student, through the breakdown table that highlights the punctuality and attendance of the student on a per-session basis, as well as the punctuality graph which gives the staff member a good visual representation of how late the student is to sign in for sessions as well as how early they sign out on average from each session.

4.3.8.1. Further development ideas

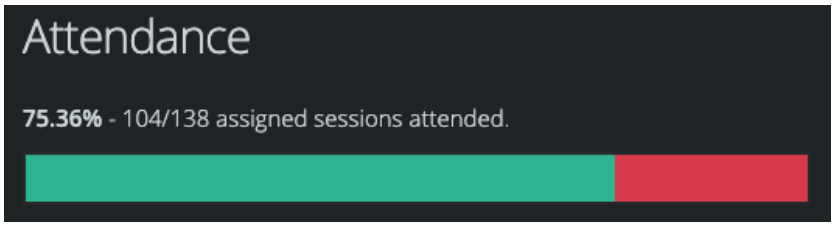
If I could continue development of this section, I would like to be able to provide further analytical information based on the data the system has available such as better insights into the attendance per day, so that staff could isolate any anomalies as well as manually adjust any bad data where students may have accidentally signed out early for example.

However, the current solution does provide the key information needed by staff to manage the precinct effectively and locate any students that may not be performing their duties fully. This is backed up well by the review from one of the stakeholders for the system, Nick Balaam, who is the staff member responsible for the precinct:

“[Student reports] are amazing as they show to me exactly who has been slacking (none of the boys in this case) but also who has always been on time or even early in some cases.”

4.3.9. Display to staff the attendance levels for individual students on the precinct, the student’s reliability

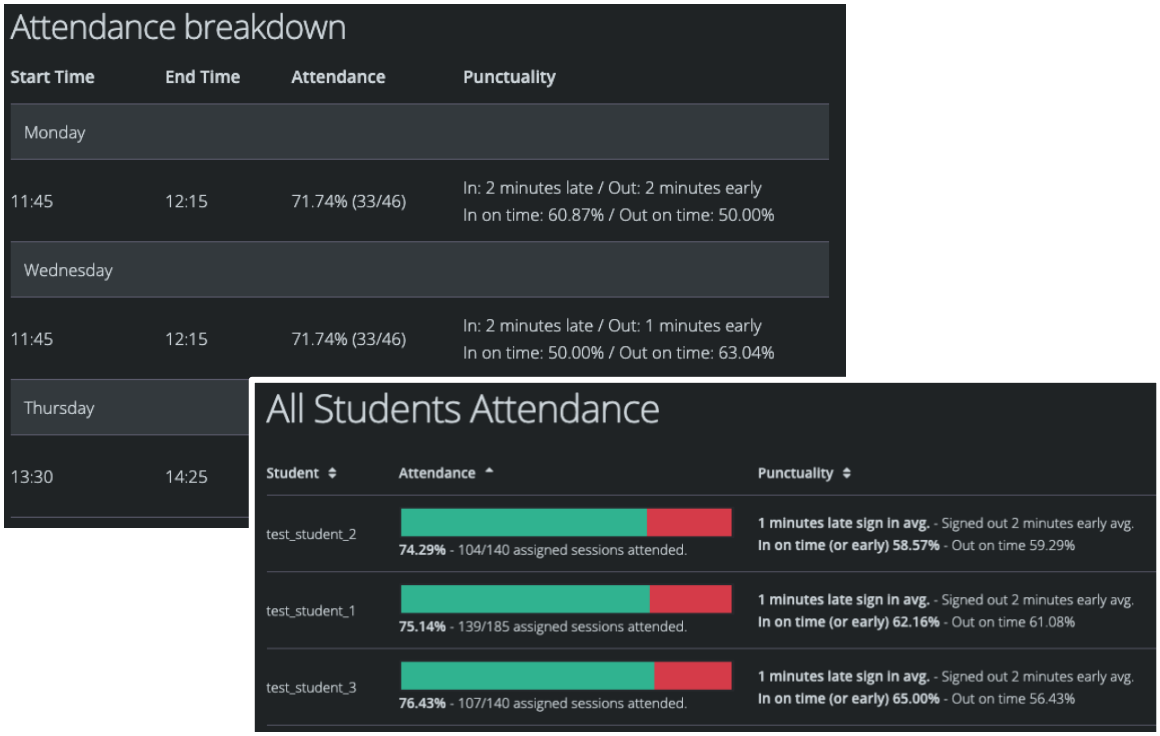
On the student report page, the page provides an easy to understand bar graph in the top-left, which indicates to the staff member at a glance the overall performance of the student by home many sessions assigned that they have attended.



This bar graph is also shown on the attendance overview of the staff homepage and so the information displayed in it is easily accessible to staff members from multiple key locations within the system’s website. I believe therefore that this objective has been met effectively by the solution developed.

4.3.10. Display to staff the duration each student stays for their precinct duty, percentage of defined precinct duration

As described in previous sections of the evaluation, the system provides a good array of data to the staff members about the students’ attendance and punctuality. Included in this is information about how late students are to sign in for each assigned session as well as how early they are to sign out on average.

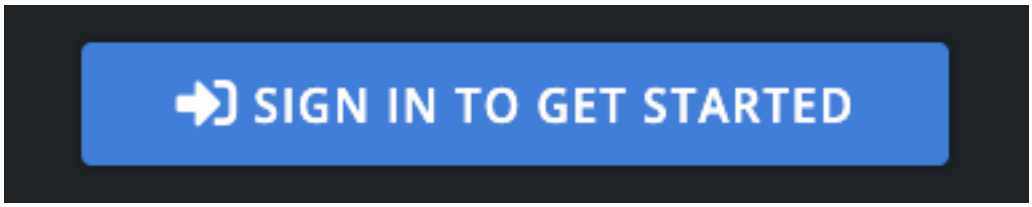


This information is displayed both on the homepage of the staff portal with the student overview where general averages for in/out are displayed for each student, as well as on the individual student report page where the in/out percentages are shown per session that is assigned to the student.

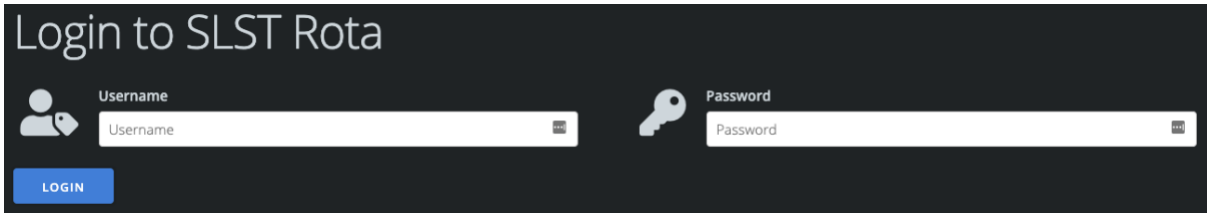
Based on this, I believe the computational solution created allows for this objective to be met effectively for the staff members using the system.

4.3.11.Provide an authentication system for staff and students

On the initial page a user reaches when accessing the system for the first time, they are greeted by a large button encouraging them to sign in to access the system.



This button takes the user to a single sign in form that works for both student and staff accounts registered in the system. The form requires both the account username and password to sign in, as is standard with login forms throughout the web.



In addition to this basic authentication system used throughout the website, once signed in both student and staff accounts have the ability to update their own passwords should they need to. This is done through a simple form under the account tab in the navigation bar, requiring the current password and the new password with confirmation for the new password.

This provides the system with a competent authentication system that is easy to use for all accounts.

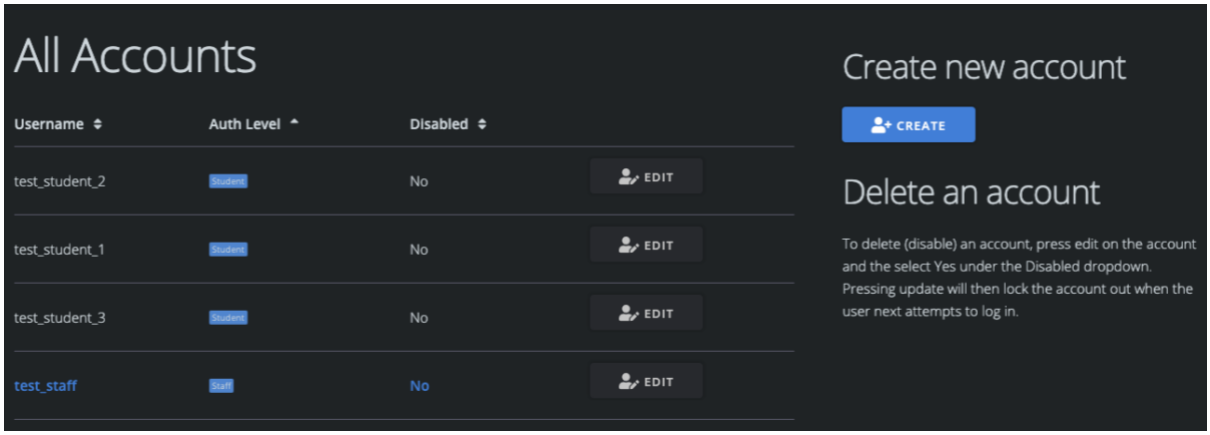
4.3.11.1. Further development ideas

However, a further feature to develop could be allowing forgotten passwords to be reset by email, which was an original intention for this system. Unfortunately, in the time I had this was not feasible due to needing a proper server to be able to send mail correctly. This should be easily to implement with the proper setup however, as the system would simply need to store account emails and then be able to generate reset tokens with an expiry date when a reset is requested.

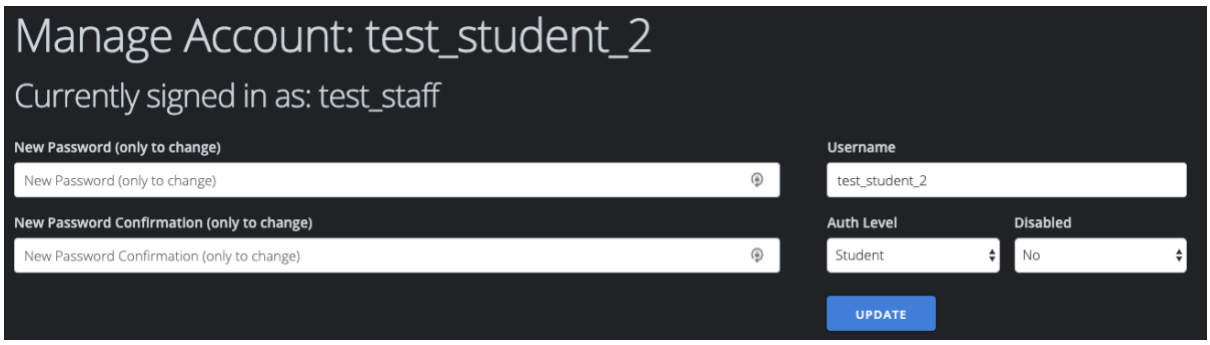
This objective has however still been met by the final solution as it provides the rota site with an easy-to-use login system.

4.3.12. Allow staff to easily manage student accounts and other staff accounts

From the staff portal homepage, the accounts link in the navigation bar will take them to the main account overview page where they can manage all accounts in the system, both students and staff.



This page shows key information about every account, whether it is disabled and what type of account it is, as well as providing a button to edit the account. This button takes the staff member to a page that allows them to update all properties of the selected account.



This page allows for the password on an account to be changed without knowing the previous one, useful if a student forgets their password so that a staff member can then reset it. Further, the page also allows for the account username to be changed as well as switching it between staff & student.

To “delete” an account, the disabled drop down can be used which stops the account being able to be used to be signed in as well as removing it from any sessions if a student account.

The account overview page also has a button to create a new account which takes the staff member to a form similar to the account edit page but without information pre-filled nor the option to disable it immediately.

With all these features, the rota system provides clear options for staff members to be able to easily manage existing accounts as well as being able to create new ones with ease. Therefor I am of the opinion that the system has effectively met the objective here.

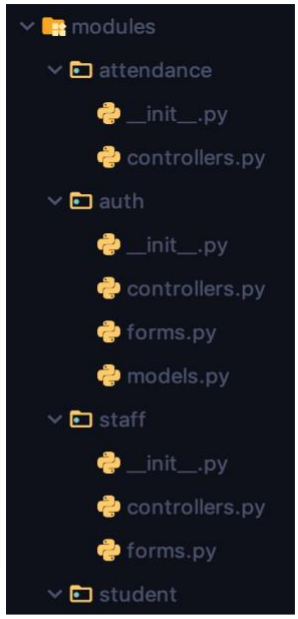
4.3.13.Final thoughts

Overall, the computational solution produced for the initial problem of rota/precinct management is far better than any current system can provide with the bespoke features designed to meet the needs of the staff and students responsible for the precinct area.

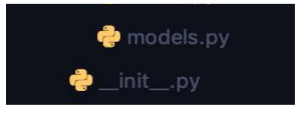
4.3.13.1. Maintenance of final solution

The program was designed with a clear structure in mind that splits apart all main sections of the website into dedicated directories and controllers. This makes the program far easier to maintain as each website route is easy to locate through the base route sections and comments in the controller files.

```python
# Login
@auth.route('/login/', methods=['GET', '
def login():...


# Logout
@auth.route('/logout/', methods=['GET'])
def logout():...


# Manage Account
@auth.route('/account/', methods=['GET',
@auth.route('/account/<int:id>', methods
def account(id: int = None):...
```




```python
# Student overview
@attendance.route('/student/<int:student_id>', methods=['GET'])
def student(student_id: int):
```



The only part of the software that may lead to maintenance issues in the future is the table macro, which contains multiple “bodge” solutions to allow extra styling to be applied to rows of a table. Whilst this works fine now, adding anything extra to this macro may be complicated and could cause issues.

```
{% for item in data %}
    <tr class="{% if item[0] %}highlight {% endif %}{% if item[2] %}subheading {% endif %}">
        {% for value in item[1] %}
            {% if value is not string and value is iterable and value|length >= 2 %}
                <td data-sort="{{ value[1] }}">{{ value[0] }}</td>
            {% else %}
                <td>{{ value }}</td>
            {% endif %}
        {% endfor %}
    </tr>
{% endfor %}
```

4.3.13.1.1. Further development to resolve maintenance issues

To mitigate the issue raised with the table macro, this could be redeveloped with a cleaner solution to passing extra styling into certain rows of the table. This would be complicated to achieve due to the many places in the software solution that call to the table macro, but would make it far easier for any further changes that may be made to the tables.

However, the software solution has managed to achieve all the initial objectives and provides a solid product that can be used by them going forward.