93601

936011

S

OUTSTANDING SCHOLARSHIP

NEW ZEALAND QUALIFICATIONS AUTHORITY
MANA TOHU MĀTAURANGA O AOTEAROA

QUALIFY FOR THE FUTURE WORLD
KIA NOHO TAKATŪ KI TŌ ĀMUA AO!

# Scholarship 2021
# Technology

# Scholarship Technology Report

## Supplementary Materials

- Video Tours
    - [redacted]
    - [redacted]
- GitHub
    - [redacted]
    - [redacted]
- Testing Documentation
    - See appendix at the end of the document

## Introduction

In Year 13, my school offers students the opportunity to earn a service award to encourage them to contribute to school life. These Service Awards are certificates that are received by Year 13 students at the Graduation Ceremony at the end of the year. There are different levels of service award available depending on the number of hours of service completed, with students having to provide evidence to staff in the form of a log of their completed hours to verify their award. This system was introduced at an assembly at the beginning of the year, where the only guidance given for this verification system was that students should record a log somewhere (no template was provided) and that they'd then need to transfer the data from this log to a Google Form at the end of the year. As a student, I found this irritating as there was no guidance given about what this form would look like and as such, what information I'd need to record. Many other students had the same complaint as I did.

Although I had found the system slightly irritating, I did not think that it had real potential as a project until I had a conversation with the teacher-in-charge of Debating at a Debating tournament. We were discussing my complaints with the system from a student perspective, when she noted that she also had complaints about it from an administrative perspective. Both herself and other teachers had received little guidance from senior staff about their role in the system, and many had taken it upon themselves to record the hours of students for them on their own spreadsheets or the likes, with the intention of then submitting these directly to senior staff at the end of the year. I noted that this was not what students had been told to do; they had been told that they should record their own log, not have teachers log them for them. She further explained that in previous years, she had numerous students tell her that the system was too confusing and laboursome and as such, hadn't bothered to submit their hours and missed out on an award. This conversation indicated that there were deep-rooted, systemic problems with the system that were affecting all stakeholders involved, and thus there was room to develop a better system.

It was alarming to me that students, who may have been completing the required service, were not bothering to get an award to recognise this at the end of the year. In my own experience, having done a significant amount of volunteer work, I'd found that this had been instrumental in helping my applications for jobs and other opportunities. In my eyes at least, having a service award that indicated the number of hours of service a student had done would be useful, as it would show prospective employers or the likes that a student not only had interests outside themselves, but is hard working. Therefore, it was apparent that there was a need for a new system to be developed.

## Investigation

Before attempting to diagnose any of the problems with the current system, I first wanted to define its purpose, value to students and key stakeholders (particularly the group of students it was not working for). This would ensure that I was analysing the system by investigating what was stopping it from achieving its intended purpose for its various stakeholders, thus allowing me to develop a system that would address these concerns and better meet its purpose

**The Levels of Service Award Available and their Requirements**

| Level | Requirements |
|---|---|
| Service for Graduation | 1 - 2 hours (required to receive the Graduation Diploma) |
| Silver Award | 20 - 30 hours of service |
| Gold Award | 30 - 40 hours of service |
| Platinum Award | 40+ hours of service |

## Purpose

Although from the school's perspective the purpose of the awards was to encourage students to complete service for the school, I was more interested in finding the purpose and value of the awards for students. I had noted earlier that, based on my anecdotal experience, the value of a service award to students was that it would assist them with finding employment or in their applications for other opportunities. To confirm this, I looked to the growing body of research surrounding volunteering and its benefits. A study by the Corporation for National and Community Service[1] in the United States verified my claim and found that volunteering is associated with a 27% higher odds of employment. The same study also found that volunteer work boosted the odds of employment by 51% for individuals who did not have a high school diploma or equivalent i.e. those who might not pass NCEA Level 3.

My research also revealed another benefit to volunteering, and by extension, earning a service award. An article by career coach Nancy Collamer[2], who writes for Forbes magazine, noted that the act of volunteering helps to boost morale in young people, making them feel both needed and productive. She explained that this sort of positive mindset is the most critical factor which helps unemployed people get into jobs.

Therefore, it was clear that the purpose of the service awards should be to provide students with documentation and boost their morale, thus supporting them in finding employment or other opportunities. Further to this, my research revealed that these impacts were even more true for students who would be going directly into employment, particularly if they were lower achieving academically, and thus this group would need to be a key stakeholder going forwards. For these students, a service award would be an important feature of a CV to help them differentiate themselves from other applicants.

While researching, I also briefly looked into general barriers to volunteer work. As I was looking at volunteering within a school setting, this wasn't entirely relevant, however the findings were of interest as they indicated that volunteering within school presented a unique opportunity. One of my own observations from my volunteer work outside of school was that it tended to exclude certain demographics; you didn't have time to volunteer if you needed to work or spend time with family. The UK Government 2018-2019 Community Life Survey[3] highlighted this, finding that the three biggest barriers to volunteer work were work commitments (49%), other activities taking up spare time (35%) and having to look after family (23%). The same survey also noted that

[1] https://www.nationalservice.gov/sites/default/files/upload/employment_research_report.pdf

[2] https://www.forbes.com/sites/nextavenue/2013/06/24/proof-that-volunteering-pays-off-for-job-hunters/?sh=70a13a19753a

[3] https://www.gov.uk/government/statistics/community-life-survey-2018-19

people would be more likely to volunteer if they could be flexible with their time commitment to volunteering (50%) and how they were able to give their time e.g. online from home (40%). It is unfortunate that the young people who are likely to benefit most from a record of volunteer service (those who go directly into employment after school) are also those who are more likely to face the aforementioned systemic barriers (e.g. needing to work part-time, looking after family).

With this in mind, the in-school service awards represented a unique opportunity to make the playing field more level; students already had to be at school, so any service they completed during this time or before/after school would not intrude on the rest of their time as much as other volunteer work would.

### Profile of Award-Receiving Students and Purpose Confirmation

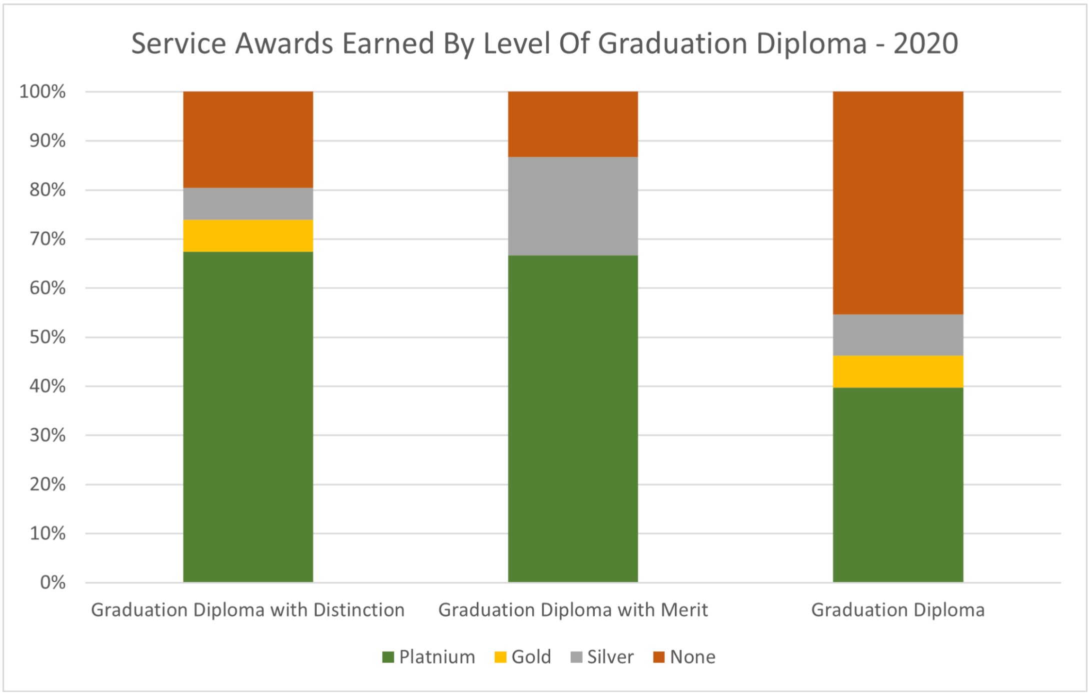
Having now identified that the students which stood to benefit the most from a volunteer service award were those who would go directly into employment after school, I needed to check if these students were actually receiving awards. This would verify that there were issues with the current system, thus legitimising improving the system as a project.

I wanted to get my hands on the data (overall grades, after-school intentions, ethnicity, gender etc.) of the students who had gotten service awards in previous years, so I went and visited the Senior Division Principle, Mr R[redacted] who manages the administration of the awards. He said that he would have a go at getting me the data from the school database administrator, although he noted that there could be some privacy concerns with this and, at the very least, the data would need to be scrubbed of any personally-identifying information.

While meeting with Mr R[redacted] I also ran the purpose of the service awards I was using to frame my project past him. He said that the main purpose from the school's perspective was to encourage students to complete service, but agreed with what I had defined for the purpose of my project, and was surprised to see just how useful volunteer work was in finding employment or the like. He also noted that the service awards were able to legitimise the work of students if they did want to use it on something like a CV. Rather than just saying that they helped coach a sports team or were a student support prefect, they could say that they were awarded a platinum service award in recognition of their work coaching a sports team and as a student support prefect. This recognition would highlight the importance and value of their work.

After a week, I went back to Mr R[redacted] to see if there had been any progress with getting the data. Unfortunately, there were some hurdles to get over and he said that it might take a few weeks as the database administrator was quite busy.

I wanted to get moving with my project but I still wanted to get a profile of the students who were receiving service awards, so I instead consulted the prize list from last year's Graduation Ceremony. Our school gives out both Service Awards and Graduation Diplomas at this event, with the level of diploma received depending on the student's academic achievement that year. Using the data accessible on our school website, I matched each student's level of graduation diploma with their service award, producing the following graph.



It was clear that high achieving students, those who would receive their Graduation Diploma with Distinction or Merit, were significantly more likely to receive a service award than their peers who received the regular Graduation Diploma. These peers were also the students who were more likely to be heading directly into employment than those who received a higher level of Graduation Diploma and, as aforementioned, benefit more from a service award. Consequently, it became clear that our service award system was not working for all students, particularly those who needed them most and therefore there was room for it to be improved. Ideally, it would have been good to have some more information about the demographic of students that fitted into this group, but this was enough to validate my concerns with the system and so I continued my investigation.

## Stakeholders

To conduct a thorough investigation into the service award system, I need to define the various stakeholders that were important to it. These would also be the stakeholders I would need to continue consulting with throughout the project to gather feedback and guide my development of a new system.

1. Students
   There were two subgroups of students that were important to define.
   a. Students heading directly into employment
      These were the students that stood to benefit the most from gaining a service award. As such, they were the most important stakeholder in my project.
   b. Higher-achieving students
      While these students did not stand to gain as much from the new system, it was still important that my project worked for all students.
2. System Administrator (Mr R[redacted]
   Being the teacher-in-charge of the service award program, Mr R[redacted] would be my primary staff stakeholder and be able to give me guidance around what would work for staff, as well as for him when administering the system. As I was a Head Prefect, I also met weekly with Mr R[redacted] anyway, which gave me ample opportunity to get feedback from him.
3. Staff (Ms H[redacted]/Mr G[redacted])
   As the TICs of Debating and Sustainability Council respectively, these teachers would be able to give me a staff-based opinion of the system. Ms H[redacted] had prompted my initial concerns about the system, and Mr Gibson was one of the teachers who was recording service hours on behalf of students.
4. Technical Stakeholder (Mr D[redacted]
   Mr D[redacted] my digital technologies teacher, would be my technical stakeholder, providing me with guidance on best practices and addressing my technical concerns throughout the project.

## Stakeholder Engagement

Having now defined the purpose of my project, who it needed to work for and who I needed to talk to, I could finally begin conversations with stakeholders to figure out where the current system was going wrong and why it wasn't working for all students.

There were three broad areas that could have been creating barriers to students gaining a service award. These were:

1. Access to and understanding of volunteer opportunities

Did students actually get chances to volunteer? Did students and staff understand what counted as service and were correctly recognising this?

2. <u>The perceived value of service awards</u>
   Did students care about getting an award and understand the value of one?
3. <u>Administrative barriers associated with the recording and submitting of service hours</u>
   Was a confusing system putting off students or were they managing to soldier through it?

My initial conversations with students I knew and staff had suggested that the key problems with the current system lay in its administration, but this did not preclude other areas from also having problems. My intentions with my project at this point were to create some sort of website, but I was unsure exactly what this would look like and whether it would be focused on helping students find volunteering opportunities within school, tracking their progress or something else entirely. I also wanted to pick up any other grievances which students and staff were having to fix along the way.

To engage with stakeholders, I conducted interviews with ten students with a variety of after-school intentions, as well as the staff I had laid out in my stakeholder setup. I then separated out the content from these conversations into the three areas I had defined earlier.

***<u>1. Access to and understanding of volunteer opportunities</u>***

Before looking at the rest of the system, it was important to check that students actually had chances to volunteer and complete service for the school; if students couldn't volunteer, then they were never actually going to get an award. Further to this, I also needed to check whether students understood what work of theirs counted as volunteer work. This was important as if they did not see their work as volunteer work, then they would never record it and get the award they deserved.

Irrespective of their plans for when they finished school or whether they wanted to get a service award, all the students I interviewed thought that they had good opportunities to gain service hours. These opportunities were wide-ranging; some were gaining their service hours from leadership roles, while others were from academic tutoring or coaching sports teams. A number of the students noted that they didn't really care about getting the award, but that they were completing the required work anyway and would get the award regardless.

Students were also offered a number of one-off opportunities to gain service hours. A quick glance through my emails revealed opportunities to help out with athletics day, the school environment group, careers office and other odd jobs for deans.

Senior management had put significant effort into explaining what qualified as service to the school, mainly by illustrating the difference between service and participation. This was brought up regularly in assemblies and in emails at the start of the year that were sent out to the Year 13 Year group. Despite this, there was still a large focus on service as coming from regular, official roles, such as being a prefect or peer tutor.

This was highlighted by Mr D[redacted] who felt that the focus on service hours as coming from traditional and well-defined roles was causing the service of some students to go unrecognised. He explained that this was because much of this work would fall outside of what was generally considered to be volunteer work, such as having a leadership role, tutoring or being on a council. Instead, their work would be in undefined or unofficial roles like helping to run sports teams or cultural groups that they were part of, assisting siblings and friends with work or doing odd jobs for teachers at schools. He had numerous examples of students who he had seen undertaking these kinds of work which had been going unrecognised.

From my conversations within this area, I drew the following conclusions:

- Students had good and varied opportunities to earn service awards, with a mix of ongoing and one-off events that covered a wide range of styles and abilities
- A side-effect of the heavy promotion of these opportunities, which all needed to be signed up for, was that students completing service in less official capacities did not always see their work as counting as service and thus were not recording it.

Therefore, the new system would need to ensure that it placed equal value on all times of volunteer work and ensure that service that fell outside of its typical types was recognised.

## ***2. Perceived value of service awards***

Now that I knew that students had adequate opportunities to earn hours, I needed to check if students actually wanted to earn them.

There was a notable difference in the service award aspirations between the students intending to go into further tertiary study (100% of whom intended to get an award) and those who were not (33% of whom intended to get an award). This supported my observations for the Graduation Diploma vs. Service Award data analysed earlier (see

Figure 2). Of those who intended to get an award, 75% said it was because they thought it would look good on their CV, while 25% indicated that they did not particularly care about it, but would be getting one anyway as they were already doing the work. Those who did not intend to get a service award indicated that this was because they did not see value in it. Notably, these students indicated that they still thought that they had good opportunities to gain service hours, but they all had complaints about the difficulty of recording and submitting them to the school - put simply, it seemed like a hassle for little reward to them.

My research had indicated that there was in fact value for these students in getting an award, which was clearly not being communicated to students effectively. This highlighted that there was a need for further elaboration on the value of service awards from Senior Division staff.

Alternatively, a new system might also benefit from finding other ways of encouraging students to complete and log service (perhaps by adding competitive elements or the likes).

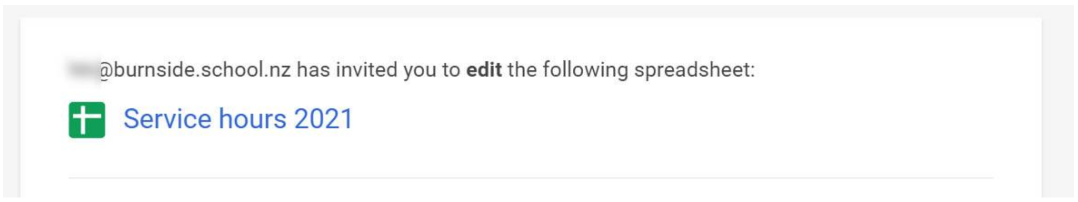
***<u>3. Administrative barriers associated with the recording and submitting of service hours</u>***

From my conversations with staff and students, mixed with my own observations, there were a number of administrative barriers associated with students tracking their service hours and earning their awards.

As aforementioned, the only guidance students were given regarding tracking their hours was that they should keep a log somewhere and that they would then submit this information at the end of the year via a form. There was no template provided nor explanation of what information students would need to record and provide at the end of the year, and consequently, all of the students I talked to said that they had found this process (or lack thereof) confusing.

I first talked to my students about how they were recording their hours, which revealed that very few of them had developed a logging system that allowed them to effectively track their progress. 33% of the students were recording their hours in a notes app on their phone, a format which meant that to calculate their current number of hours they would have to manually add them all up. The remaining 67% were using a spreadsheet, although they were not competent in their usage and only one had figured out how to have their total service hours automatically be added up. In this way, students were not able to track their overall progress, or even their contribution to each group, which could have been decreasing motivation or a sense of accomplishment with the system. This was an area that would need to be improved in the new system.

Moreover, there was also confusion amongst staff regarding their role in recording service hours, some of which were recording the service hours of students for them. I spoke to the teachers in charge of the Debating Club and Sustainability Council about this. Both of these teachers had implemented their own methods of recording hours, although they both admitted that this was based on their conversations with other teachers and not on direction from staff managing the system. In the case of the debating teacher, she had just sent out a spreadsheet where students could record them, but she was not too bothered about whether this was used or not. The teacher managing Sustainability Council was recording the service hours of students for them on a spreadsheet, with the intention of then sending this to senior management at the end of the year. Despite this, he noted that he had never had any direction from senior management on doing this and was unsure of who he would send the form to. His model for this was based on the school's Environment Group model, a large club which many Year 13 students joined where this same system was also used. His main reasoning for this system was so that he could verify that students were not falsifying service hours. His concerns about the verification of hours highlighted an area that I would need to explore further.

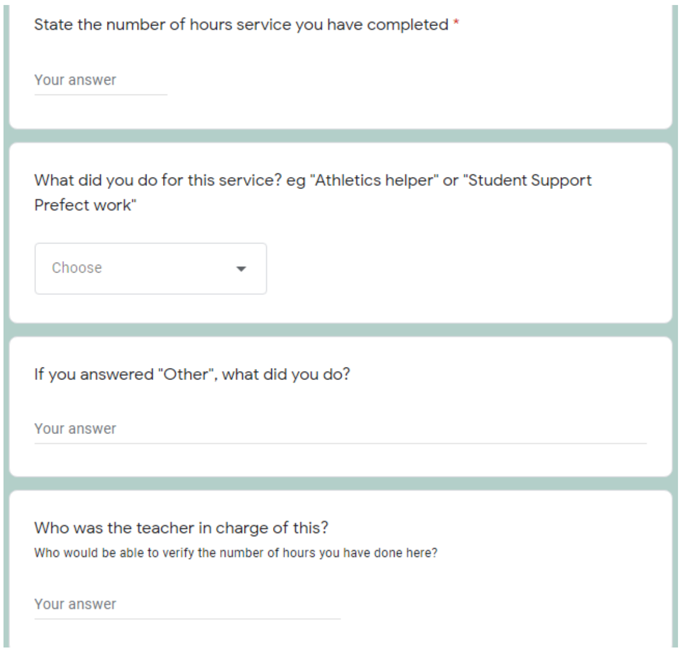

[redacted]@burnside.school.nz has invited you to **edit** the following spreadsheet:

Service hours 2021

*The service hours spreadsheet sent out by my debating teacher*

As was reflected in my conversations with stakeholders, the current service hours tracking system was confusing for students, as they were unsure of what, where, who or how they should be recording them. Although most of the students I talked to had set up some kind of system for recording them, the majority of students were not keeping up to date with logging them. This was largely due to service hours not needing to be submitted till the end of the year, which would mean that when the end of the year rolled around, reverse-engineering a service log would be challenging, putting off some students.

I brought up these concerns when I talked to the member of staff in charge of the Service Award system, Mr R[redacted] the Senior Division principal. He was receptive to my idea of improving the system and noted that he had already had some concerns about the current one, particularly surrounding that the submitting was left till the end of the year. After our conversation, he decided to launch a new system in the interim and during Term 2 the system was changed to a form that students could fill out during the year. While this gave students more of an indication of what information they should be recording, I went back and talked to the students in my stakeholder consultation group, who indicated that it seemed like more work. Many were unsure of whether they should then be submitting totals for each group as a total for a week or term, or filling it out after each time they volunteered. All of the students also complained about the new form not giving them the ability to track their progress during the year, which meant that they did not know how many more hours they needed to do to get their award. Moreover, as the new form did not allow students to see what hours they had logged previously, many of them were forgetting which hours they had logged in the past, making it difficult to track where they were up to with their logging. All of the students I interviewed said that they would be more likely to record their hours and subsequently get an award if the process was more streamlined and easier to navigate.


State the number of hours service you have completed *

Your answer

What did you do for this service? eg "Athletics helper" or "Student Support Prefect work"

Choose

If you answered "Other", what did you do?

Your answer

Who was the teacher in charge of this?

Who would be able to verify the number of hours you have done here?

Your answer

This form also introduced a new issue; students would have to select from a dropdown list what role/group their service had been completed under. For the majority of students, this would be helpful, but it would also send the message to students who were getting their service hours from different sources that their service did not qualify. Mr D[redacted] noted that this would exacerbate the focus currently placed on service coming from official and traditional work, as was mentioned in an earlier section.

There were several issues occurring in this area and from these conversations, I reached the following conclusions:

- Students were unsure of what they should be recording and how they should be doing it. Within this, students were confused about:
    - Whether they should record their service hours on the new form as they went, or at the end
    - What information they needed to submit (although this had largely been addressed by the new form, it would be an important consideration of a new form)

    - Whether they should actually be recording their hours. As many teachers were recording the service hours of students for them (when they weren't really supposed to be), students were further confused about whose job it was to be recording their hours. Notably, this would differ between different groups, making it even more confusing for students.
- Students were not being provided with adequate feedback on their progress towards getting an award.  This meant that students were missing out on feedback from:
    - Themselves - students were unable to easily track how many hours they currently had, or even which hours they had logged in the past. This was making it difficult for students to track their overall progress towards their goals.
    - Staff - since there was no data accessible about student totals, staff would also be unable to encourage or talk to students about their progress. This could mean that students were missing out on encouragement from teachers e.g. form teachers congratulating students in their class about their progress so far.
- The submission process continued to emphasise service hours earned from traditional opportunities through its dropdown form. While the majority of service hours would be earned through work of this type, it was further discrediting service that fell outside of this scope, stopping it from being recorded.

## Reflection on Stakeholder Engagement

While there were some issues surrounding the perceptions of service and how motivated students were to earn an award, the majority of issues with the current system involved the logging of hours. Therefore, I decided that my project would focus on developing a new system for submitting hours, which would aim to make this process clear, provide students and staff with adequate feedback and allow for all kinds of service to be entered. This would decrease barriers to recording service as it would no longer be as cumbersome, while also providing additional incentives by allowing students to monitor their progress towards their goals and for staff to provide encouragement. By addressing these key factors, the system would hopefully see a greater uptake from all students, particularly those heading directly into employment next year, who would benefit the most from a service award.

My stakeholder engagement also revealed that there were other issues that fell slightly outside of the scope of the logging system, namely the perceptions of what counted as service and the value that students saw in the awards. There was room for a new system to at least partially address the issues surrounding the perceptions of what counted as service by encouraging service from all kinds of work to be logged, however these two factors would need to be addressed by staff who explained and advertised the awards to students. I would need to pass this feedback on to Mr R[redacted] to ensure that these messages were communicated to students in future years.

## Planning

### Platform

I began by choosing how I would develop the new service hours system. While I did briefly consider creating a mobile app, with my personal expertise being in Flask, Python, HTML and CSS, making a Flask web application with Python was the most logical choice for a platform for the service hours tracking application. This would mean that my application would be accessible from virtually any device via a browser. Notably, 33% of the students I interviewed indicated that they were recording their service hours on the notes app on their phone, and so it was important that the web application that I developed was mobile-friendly. As I would be storing users, hour logs and other data and then essentially having students/staff query this data to view particular results, I would essentially be building a database program. I was familiar with SQL database language (the preferred one at my school), so I would use this one for my project.

### Requirements from Mr R[redacted]

I was just about ready to develop a brief for the new system based on my stakeholder engagement, but I needed to talk to a few key stakeholders to figure out what they needed from the system first.

The first of these was Mr R[redacted] who would be in charge of administering the new system. We set out that the website would need to let students log their hours (similarly to how this was currently done) and view their progress, which would also need to be accessible to staff, thus requiring a user login system. He would need a way to easily add users and then remove them all at the end of the year and clear the database. He also needed to be able to export the total hours or awards of each student at the end of the year to be entered into the school database. His final personal concern with the system was that it would need to be maintained after I left the school and so I would need to find some staff who would be able to take over the project after I left the school.

Mr R[redacted] also mentioned that I would need to ensure that my site was really easy to use and provided clear instructions in places where it was not. Although this seemed intuitive, he mentioned that he'd had a lot of uptake problems in the past when introducing new digital tools to students. This was in essence why I was undertaking this project, to avoid this kind of situation, however, his comments confirmed that I would likely need to specifically mention this ease of use in my brief.

## Project Handover

With Mr R[redacted] having raised that my project would need to be able to be handed over and maintained by the school, I first needed to check with Mr D[redacted] about what the school would need from my project to do so successfully. He said that my aim with the project would be to develop a well thought out and tested prototype, which they would be able to take on, tweak, and then launch the year after I finished school. For them to do so, he said that I should focus on creating modular code that made good use of Flask templates (which would allow me to reuse the same HTML across multiple pages), classes and functions (which would allow again allow for server-side code to be reused in multiple places). This would ensure that if there were issues with my code, they would be able to change it in one place and have it change everywhere where that same function was used. They would also look after scaling the project to be suitable for use school-wide, so this modular approach would be even more important. Finally, he said that it was important that my code was well commented and documented so that they would be able to understand what I was doing and any complicated bits of code.

## Verification

A final area I needed to investigate and one of the points of contention which arose during my initial stakeholder consultation was the degree of verification that was needed for service hours. The teacher in charge of the Sustainability Council had raised this, particularly in the context of the school environment group, and so I wanted to find out more regarding the need for service hours to be verified by staff to ensure that students were not falsifying them. When I spoke to the TIC of the environment group, he indicated that he liked to monitor the hours as there was a high chance of them being falsified as the Environment Group was popular with Year 13 students as an easy way to earn service hours.

I spoke to Mr Roberston about this when I met with him to discuss his needs and he said that he was not overly concerned about each individual hour being verified, mainly as this would be an excessive administrative burden on staff. Instead, he said that he would generally look over the hours to check if any of them seemed unbelievable, but that beyond this he did not feel the need to check them more. Mr R[redacted] noted that this laid back approach would not be the impression which students would have of the system, as they would need to indicate a teacher in charge of a group to verify their hours when submitting them, thus incentivising students to not falsify their hours. He noted that whether or not this actually happened, there was always the option for staff to look over the hours, but that he did not feel that any verification beyond this was necessary and would be a waste of staff time. There were some staff, such as the TIC Environment Group, who wanted to manage the service hours tracking as hours of their group were often recorded fraudulently, while others, such as Mr R[redacted] who were not overly concerned about service hours being verified.

At this point, I decided to investigate how other volunteer tracking solutions managed the verification problem. I had briefly looked into other alternatives earlier in the project to check that there was not a pre-existing solution to this problem, but none of the options available online were able to work with the specific needs of our tiered award system and support the group-based tracking that was essential. MobileServe was a US-based offering, which managed verification by three means - tracking the phones of users to see if they were at the right location, getting someone in charge to sign on their phone to say they'd completed it or via email, where the person in charge would need to click a link to confirm their hours. A more local offering, the University of Canterbury's Student Volunteer Army Service Award program, did not require any type of verification before hours were confirmed, but did allow staff to look over hours once they were entered.

Therefore, I identified two points at which service hours could be verified:

1. As they are entered - each service hour needs to be approved by the TIC of a particular group before being accepted into the system
2. After they are entered - students enter their service hours and these are automatically accepted, however staff can look over these hours and modify/remove them as they see fit

The first option would obviously add a significant administrative burden on staff, as indicated by Mr R[redacted] The best case scenario for this option that I could develop was that teachers would be sent an email each week if they had service hours to approve, which would then take them to a portal. At this portal, teachers would see a breakdown of all hours submitted under their groups and then be able to bulk approve them, or modify/delete hours which they did not think had occurred. I proposed this option to both the TICs of Debating and Sustainability Council, who both agreed with my concerns regarding teacher workload, but they felt that this system was fast, efficient and something that teachers would be willing to do.

However, further investigation into the school's Environment Group revealed that this system would not work for all groups. At this group, the TIC takes a role each week (which has 182 students), and so having to compare the hours of the Year 13 students against this role would be particularly difficult. This would be made worse if students did not enter their hours each week, or aggregated their hours across weeks in one entry, creating an administrative nightmare. Therefore, it was confirmed that having to approve all hours was not going to work, and so a different system would be needed.

Based on these discussions, the second option, therefore, seemed to be the best system for verification, where students' hours were automatically accepted into the system, but could be updated/deleted by staff as they saw fit. This would still work for groups that wanted a higher level of verification. For example, in the case of the Environment Group, who were taking a role each week anyway, if the staff were able to see a list of total hours at the end of the year and compare this with their own records from their role, this would make verifying the hours of students easy. These conversations revealed that my system would need to give staff the ability to edit the service hours of students after they were submitted.

### Group-Based Tracking

One of the notable features of the current service hours form was that it required students to indicate the group/role they had completed work under. I had identified that this was problematic as it illegitimised service completed outside of these traditional activities. However, this would still be the way that the majority of students would complete their service hours, so I wondered if it would be worth adding this functionality to the website. This would make it easy for staff to manage hours - if students logged their hours under a particular group, they could then easily sort through the hours to just view the ones for those in their group, and perhaps view totals from students in that group to compare with their own data (like in the case of the environment group). I proposed that giving students the option to enter the name of a teacher if there was no group would work for hours outside these groups (which would meet the verification needs).

Mr D[redacted] had raised this conundrum as a particular concern of his, so I went back to gather his opinion on my proposal. He agreed that the group-based tracking system made sense from an administrative point of view for the majority of students, but emphasised that I would need to provide clear instructions on the website to indicate that hours did not have to be recorded within these groups.

Another important consideration was how this group-based structure would function. The current hours form listed all the main groups/roles that students might earn service from for them to select. However, this list was not particularly thorough, which contributed to the problem of not all types of service being recognised. To create a thorough list, I proposed that it would be best to give staff the ability to add new groups to the system which they were responsible for. This would decrease the burden on Mr

R[redacted] to add all of the groups, and would allow for smaller groups to be recognised as well, which would help to encourage more types of service to be recognised, thus achieving the goal of my project. However, having a larger list of groups could be a pain for students to search through when selecting a group when logging hours, so I would need to add a page where users could see all the groups and join the ones which they were involved with. This would make it easier for students to log their hours as they would only need to select their group from a shorter list of just their groups, making the logging process easier and faster.  I went back to Mr R[redacted] to check this, and he agreed that it was the best way forward.

**Brief**

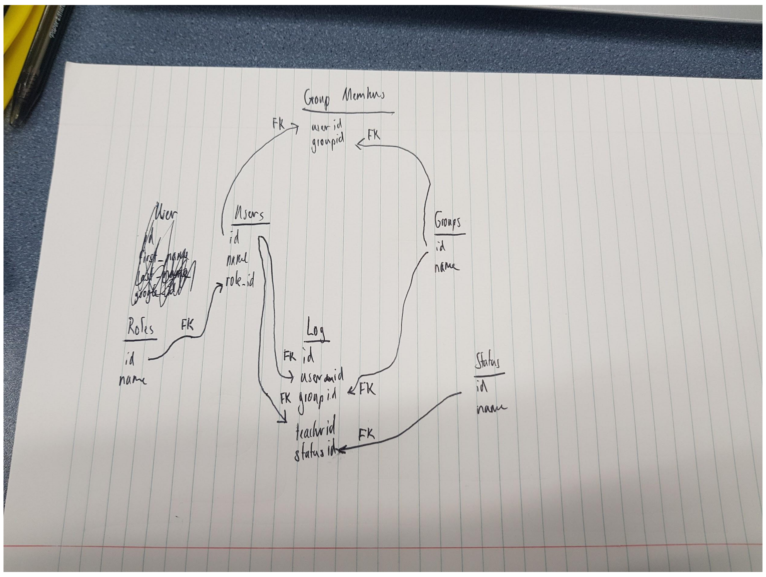
Having thoroughly investigated the current system and the requirements for a new one, my next step was to develop the specifications for the new platform that I would develop for service hours from the conclusions I had drawn from my investigations. These included:

- Must allow students to easily and quickly record their service hours, thus decreasing the administrative burden the system places on students and making service hours more likely to be recorded
- Must allow and encourage students to enter all the kinds of service that they complete, helping them to understand what counts and what doesn't
- Must allow students to monitor their progress towards service awards so that they are aware of their progress and are motivated towards achieving each level of the ward
- Must be easy for students to use and learn how to use, providing documentation where needed to do so
- Must be accessible on a range of devices (desktop, tablets, and mobile phones) as 33% of the students I interviewed were currently recording their hours on their phone, while the remained were using a computer
- Must clarify the roles of staff and the roles of students in recording service hours so that students are not confused about whether or not they should be recording their hours
- Must contain up-to-date staff and Year 13 student lists and have an easy process for staff to manage this so that staff and students can always access the system
- Must allow staff to verify service hours where they wish to (for example, in the case of the Environment Group, where the teacher in charge had genuine concerns over falsified hours) by editing or deleting logged hours
- Must allow senior staff to extract service hour data from the website to be put into the school database for awards at the end of the year or data analysis
- Must be able to be maintained by staff after I, as the developer, leave the school. To do so, the code must be modular, well-tested and thoroughly documented.

- Must be as error and bug free as possible
- May provide a framework for staff to encourage their students to complete and log more service.

## Database Plan

Since my website would essentially be a database, I began by planning the database which I would need to store the data on my website. I began by sketching out rough ideas of the tables I would need, the fields they would have and then the relationships between the tables.



I started with the user table, which would be used to store the various students and staff who would have access to the database. This table would need the following columns:

- id column, which would be used as a unique identifier and would probably just use the student's/staff school id number/code, which were used across all of our school apps. This would allow staff to lookup students via their student id, which they would be used to doing in our school database KAMAR.
- name column, which I decided would need to be split into first name and last name columns to allow for just the first name to be used in places like site greetings.
- roleid column, a foreign key that would store the id of the role (e.g. student, teacher, admin) of that particular user so that they could be differentiated between e.g. to access

Since I was using a group-based tracking system, I'd need another table that stored information about the various groups that students could join. At this stage, the only two columns that I thought I'd need were:

- id, the unique identifier for each group that was required by SQL

- name, the name of the group which would be displayed to users

I then needed a table that would store the logged service hours of students. This would need the following columns:

- id, the unique identifier for each group that was required by SQL
- studentid, to record the student who the hours belonged to, linked by a foreign key to the user table
- groupid, to record the group which the hours had been completed under. This group would then be linked to the teachers in charge, so that they could then easily view those hours recorded under their group.
- teacherid, to record a teacher who could verify the service of students if they had completed it outside of a particular group, a second foreign key with the user table
- I would also need a few columns to store the name of the activity undertaken

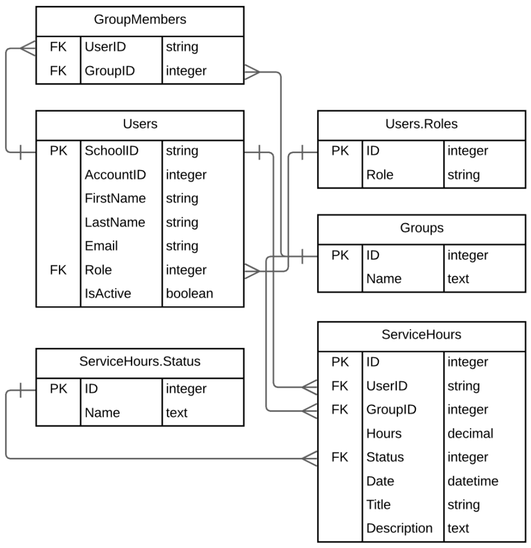
This table would effectively be a many-to-many relationship between the group and users tables.

I also needed a further table to track which students were in each group, the ones which they had opted into via the page where they could join/leave groups. This would be a direct many-to-many relationship between the users and groups tables. It would need the following columns:

- userid, to record the user id of the student in the group
- groupid, to record the group the student was a part of

I then pondered how I would record the staff in charge of each group. I briefly considered recording the id of a teacher in the group table for each group, indicating the teacher in charge. This would work fine if each group had only one teacher in charge, however I soon realised that this was not always the case; some larger groups, like the school environment group had more than one teacher (four in this case). To solve this problem, I resolved to add staff to the group members table. I'd then be able to differentiate between staff and students in the group by using SQL joins in my queries, which would allow me to filter users by their role id.

Having now planned out my database, I created an entity-relationship diagram for my database that I then ran past Mr D[redacted]



There were a few new columns that I added, which included:

- IsActive to the user table, to track whether users had logged into the website before. I thought that this could allow staff to find students who might need some help, allowing them to follow up with them
- Date to the ServiceHours table, which would record the date when the service took place

I had Mr D[redacted] check my database plan, who said that it looked good but that I needed to change my naming to use only lower case letters and underscores and remove plurals for compliance with Python conventions. This was an easy change e.g. UserRoles became user_role.

## UI Plan

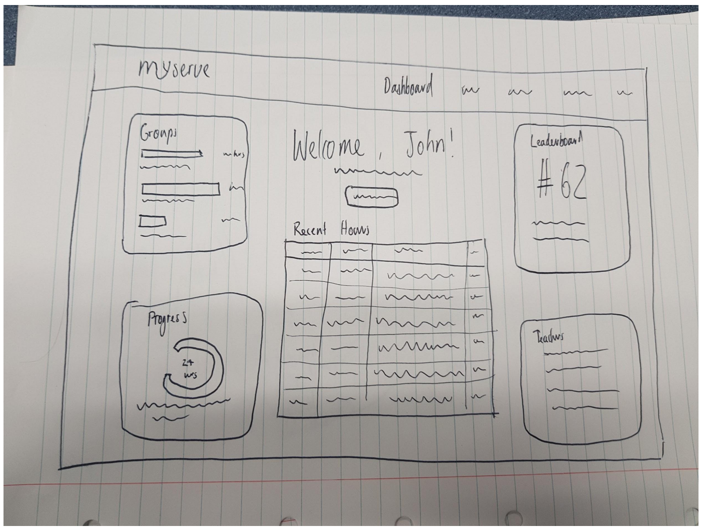
Before embarking on the development of my website, I also wanted to prototype my user interface and run this past my stakeholders to ensure that it was effective. Key to encouraging students to log hours was having a UI that was not only easy to use and functional, but that was aesthetically pleasing to users.

I began by setting out some branding details for my app by coming up with a name, logo, fonts and colour scheme. I settled on the name MyServe, as it drew emphasis to the personal nature of a student's service (by using the word “my” and also sounding like “I serve”). Using Adobe Illustrator, I whipped up some logo prototypes.

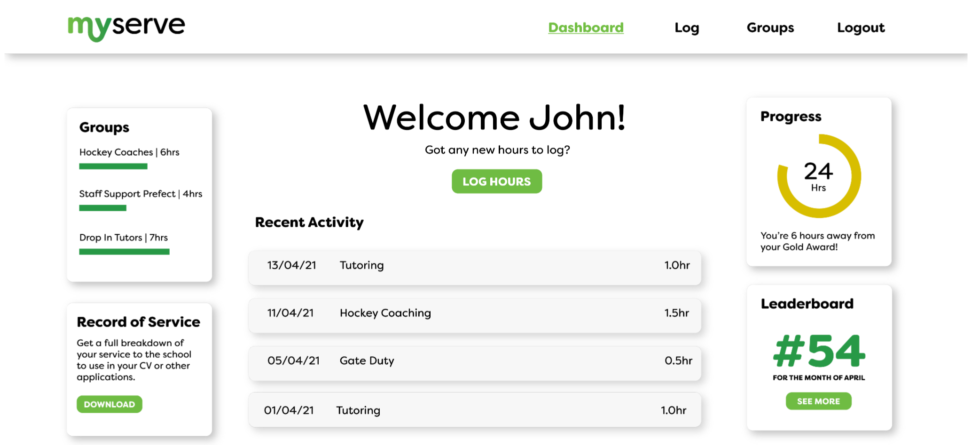
| | |
|---|---|
| Site Logo | myserve |

|  |  |
|---|---|
| Favicon |  |
| Font Families | Open Sans (Body), Filson Soft (Headings) |
| Colours | **#000000, #73bf43, #209646, #d3002a** |

I then moved onto the main layout of the website, choosing the dashboard page as a starting place as it would set the tone and styling scheme for the entire website. I started by sketching some potential layout ideas. I wanted the dashboard to use elements of gamification to make logging service hours seem more fun - for example, by adding goals, leaderboards and plenty of progress indicators.



Using Adobe Illustrator, I then produced a digital mockup of this layout to gather feedback on.  I added rounded corners and light shadows were used to give the website a soft and friendly appearance.


myserve

Dashboard Log Groups Logout

Welcome John!

Got any new hours to log?

LOG HOURS

Groups

Hockey Coaches | 6hrs

Staff Support Prefect | 4hrs

Drop In Tutors | 7hrs

Record of Service

Get a full breakdown of your service to the school to use in your CV or other applications.

DOWNLOAD

Recent Activity

| | | |
|---|---|---|
| 13/04/21 | Tutoring | 1.0hr |
| 11/04/21 | Hockey Coaching | 1.5hr |
| 05/04/21 | Gate Duty | 0.5hr |
| 01/04/21 | Tutoring | 1.0hr |

Progress

24 Hrs

You're 6 hours away from your Gold Award!

Leaderboard

#54

FOR THE MONTH OF APRIL

SEE MORE

I showed this concept to the students in my stakeholder group who liked the name, layout and logo, but were not fans of the gradient in the 'my' (they said it looked dated). Subsequently, I changed the logo so that it no longer featured a gradient.

myserve

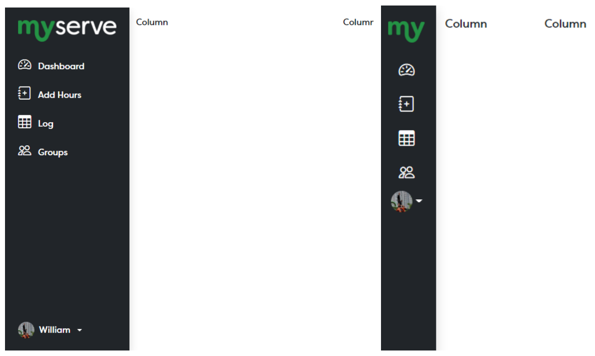
I also showed my planned UI to Mr D[redacted] to get his opinion on whether it would be feasible to create. He liked the dynamic widgets but suggested that I should scale them back and reduce the number of graphs, as these would suck up a lot of time to develop that would be better utilised on other aspects of the project. He also noted that many web apps choose to move their menu bar to the side, rather than placing it on top, as this makes them feel less like generic websites and more like an actual app. He suggested that doing so would give my app a more cohesive, sophisticated feel.

I took his advice on board and, rather than just creating another mockup in Illustrator, began writing a simple side menu using HTML, CSS and the CSS library Bootstrap. Although I wasn't intending to begin developing just yet, doing this small piece of HTML meant that I'd then be able to adapt it directly for use in my template files when it did come to development, saving me time later in the process. Using the Bootstrap library meant that I did not need to spend time styling every individual HTML element but would instead use the preset Bootstrap styles (with some modification), which would also ensure that my website was consistent across all pages. Bootstrap also had really great site grid/layout features, which would make it easy to create layouts that would adjust across different screen sizes.

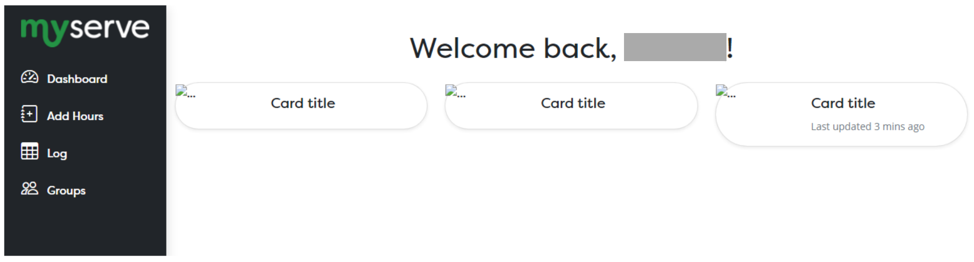
I quickly noticed that in order to have a menu on mobile phone-sized devices, I'd need to create one that could either be hidden away or used icons and took up less room. I decided to add icons so that the menu was always visible, allowing users to switch between pages quickly.

```
<div class="col-2 col-sm-4 col-md-3 col-xl-2 px-sm-2 px-0 sidenav shadow">
            <div class="d-flex flex-column align-items-center align-items-sm-start px-3
pt-2 min-vh-100 sticky-top">
                <a href='' class="align-items-center py-3">
                    <img class="img-fluid d-none d-sm-block"src="img/logo_wht.png">
                    <img class="img-fluid d-block d-sm-none"src="img/favicon_lg.png">
                </a>
```

The different breakpoints were used to specify a different size for the menu bar for each size of screen (e.g. col-sm-4), so that it generally took up the same amount of space on all screen sizes. d-none and d-block were then being used to change the logos between the desktop and mobile menus, plus get rid of the text next to the menu items.

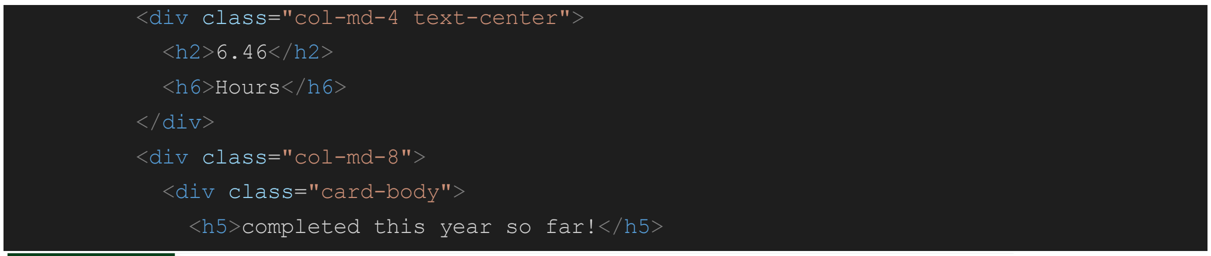


I then began to work on adding widgets to the home screen that would display the user’s total hours, current award and hours needed to get the next award. I used Bootstrap Cards for these widgets.

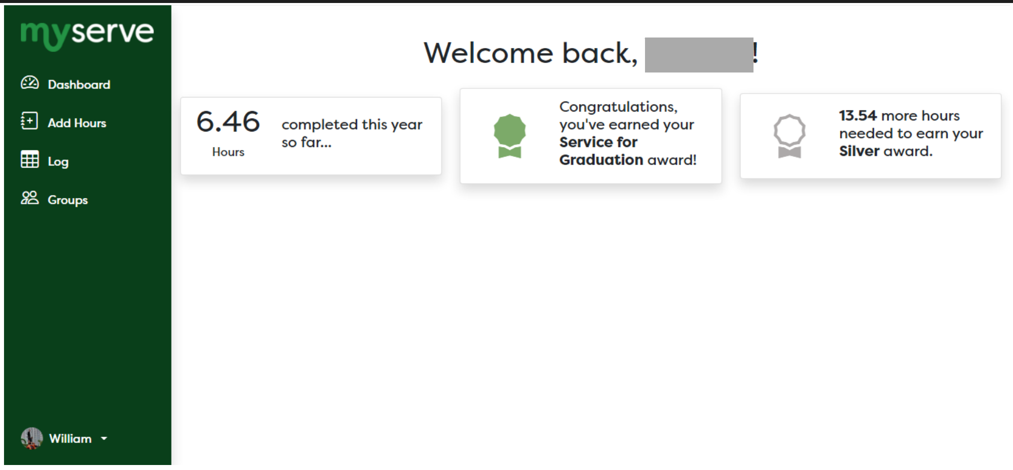


At this point, my UI was differing significantly from the one which I had shown to my four students, so I took it back to them to hear their thoughts. They all liked the new sidebar but thought that the grey colour was uninspiring and suggested using green on it instead. I changed the colour of the sidebar and finished off the demo widgets, giving the dashboard below.

```
<div class="col-lg">
    <div class="card mb-3 shadow p-2">
        <div class="row g-0 align-items-center">
```



```
          <div class="col-md-4 text-center">
            <h2>6.46</h2>
            <h6>Hours</h6>
          </div>
          <div class="col-md-8">
            <div class="card-body">
              <h5>completed this year so far!</h5>
```



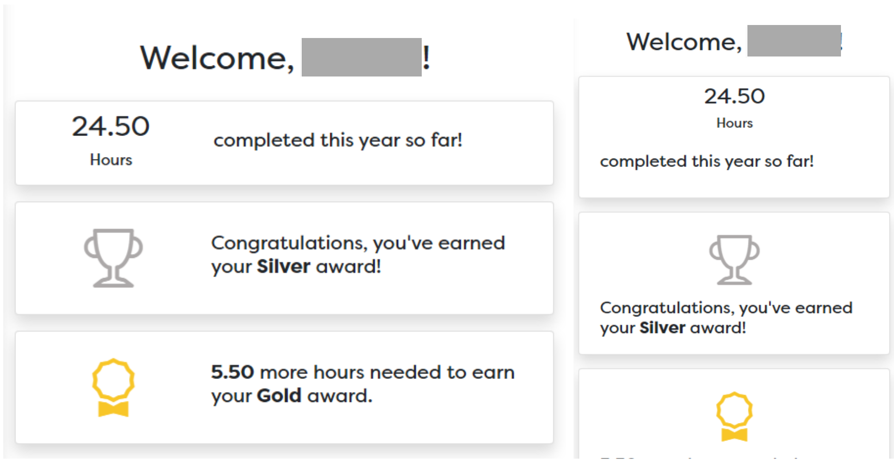
Welcome back, [redacted]!

6.46 Hours completed this year so far...

Congratulations, you've earned your **Service for Graduation** award!

**13.54** more hours needed to earn your **Silver** award.

These cards proved slightly challenging to make responsive across different devices, and so I ended up having to have two different breakpoints to change the layout of the cards. The first one would be for tablet-sized devices and smaller, where I used the col-lg class to make the cards stake vertically to fit better. Even with this, the cards still squished up too much on phone-sized devices, so I had to add a second which caused the description text on the cards to move beneath the main icon in the card. This was achieved with the col-md-8 class, which would signal this layout change on devices smaller than tablets.



Welcome, [redacted]!

24.50 Hours completed this year so far!

Congratulations, you've earned your **Silver** award!

**5.50** more hours needed to earn your **Gold** award.

Welcome, [redacted]!

24.50 Hours completed this year so far!

Congratulations, you've earned your **Silver** award!

## Detailed Project Plan

Now that I had developed the requirements for my project and my basic UI/database plan, I was able to create a detailed plan for my project. Development wise, my plan was to split the project into sections that would be developed, starting with core functionality before then layering on more features. This sprint-like development structure would allow me to develop a functional website as I went that could then be tested and receive stakeholder feedback as it was developed.

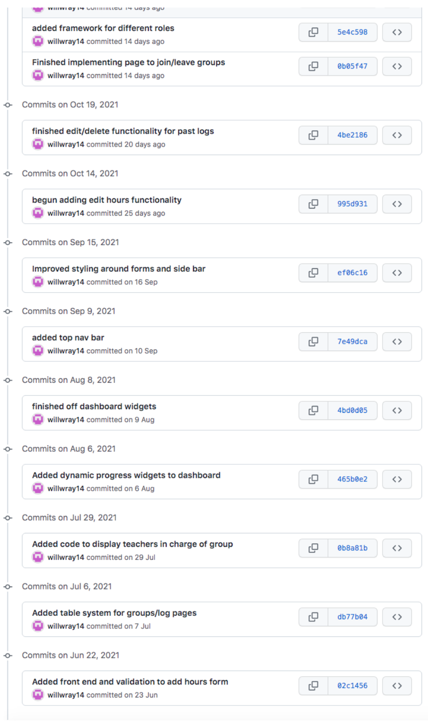
The order of development would go as follows:

- Set up my database, models.py and flask app
  The requirements of the database would be defined once I had planned the rest of my website, with the models.py file linking this database to my python code and flask app.
- Add ability for users to sign in with their school accounts
  All apps/websites used by the school required students and staff to login with their school email and password, so my website would need to do the same. Mr D[redacted] my technical stakeholder, suggested that I use the Sign-In with Google API. Each student and staff member at our school has a Google Account, and so this would allow students and staff to sign in without creating a new login. This will also alleviate some security concerns, as through this method my app will not have to store or handle passwords.
- Develop template structure for HTML
  To avoid repeating code and make it easy to have cross-site consistency, my website would need to use Jinja2 templates that would work with my Flask app to dynamically update. Moreover, these templates would allow me to split up my website code into different files that could be reused across pages. For example, a page displaying a student's group might import the file with the code for the HTML file setup, a file that defined the layout of a navigation bar, a file that defined the content of that navigation bar, before adding the content of that page to it. This would ensure that, say I wanted to change the appearance of the navigation bar, I could change it in that one file and it would be updated across all pages where it was used in my website
- Develop the student portal
  The crux of my project revolved around making it easier for and encouraging students to record their service hours, so I would start by developing the interface that students would see when logging hours. This would include the following pages:
  - A dashboard page, where students would see their progress towards their awards
  - A page with a form for users to log hours
  - A page that displayed each of the logged hours of the student
  - A page that allowed students to edit their previously logged hours in case they had made a mistake
  - A page that displayed the groups a student was in

        - A page that allowed the student to join/leave groups

- Before adding a staff portal, add a user permission system to only allow access to the correct pages for each user type
- Develop the staff portal
  This portal would contain the necessary functions staff would need to monitor the progress of their students, with extra functionality being available to staff like Mr R[redacted] to add/remove users. The following pages would be needed for the staff portal
    - A dashboard with quick links or the likes to pages staff would use
    - A page displaying all students and their totals/awards - Mr R[redacted] would need to be able to export the data from this page
    - Pages showing each user's log and groups
    - A page for staff to see groups they are in
    - A page for staff to join/leave/create groups
    - A page for staff to see the students in a particular group
    - Pages for staff to see each student in each groups breakdown of their hours in that group
    - A page to add/delete users

To divide my progress into these manageable chunks I would use GitGub, setting up a repository and then committing my changes whenever I had finished one of these chunks. Of course, there were some deviations from my plan as I had to return to previous features and address bugs or changes (which will be explored in further detail), but this modular development structure would prove to be effective. See the supplementary documentation at the beginning of this document for a full breakdown of my GitHub log and development.



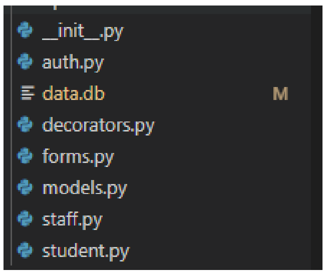
```
added framework for different roles
willwray14 committed 14 days ago                    5e4c598
Finished implementing page to join/leave groups
willwray14 committed 14 days ago                    0b05f47

Commits on Oct 19, 2021
finished edit/delete functionality for past logs
willwray14 committed 20 days ago                    4be2186

Commits on Oct 14, 2021
begun adding edit hours functionality
willwray14 committed 25 days ago                    995d931

Commits on Sep 15, 2021
Improved styling around forms and side bar
willwray14 committed on 16 Sep                      ef06c16

Commits on Sep 9, 2021
added top nav bar
willwray14 committed on 10 Sep                      7e49dca

Commits on Aug 8, 2021
finished off dashboard widgets
willwray14 committed on 9 Aug                       4bd0d05

Commits on Aug 6, 2021
Added dynamic progress widgets to dashboard
willwray14 committed on 6 Aug                       465b0e2

Commits on Jul 29, 2021
Added code to display teachers in charge of group
willwray14 committed on 29 Jul                      0b8a81b

Commits on Jul 6, 2021
Added table system for groups/log pages
willwray14 committed on 7 Jul                       db77b04

Commits on Jun 22, 2021
Added front end and validation to add hours form
willwray14 committed on 23 Jun                      02c1456
```

## Development

Now that my planning was complete, I could begin to develop my website, based on the chunks that I had separated my project out into.

## Setting Up My Database and Flask App

I started by creating the files that I would need for my program. I split my program up into several files to make it easier to manage, with __init__.py containing the code to set up my app, auth.py containing the routes (pages) that controlled users and logging in, models.py which would manage my interactions with the database, students.py that would manage the student-related routes (pages) and staff.py that would manage the staff-related routes.


```
__init__.py
auth.py
data.db      M
decorators.py
forms.py
models.py
staff.py
student.py
```

```
# import the routes from the other files
from myserve.staff import staff as staff_blueprint
from myserve.student import student as student_blueprint
from myserve.auth import auth as auth_blueprint

app.register_blueprint(auth_blueprint)

app.register_blueprint(student_blueprint, url_prefix="/student")

app.register_blueprint(staff_blueprint, url_prefix="/staff")
```

The pages with routes would all then be imported using blueprints, with /student/ and /staff/ prefixes added to all URLs e.g. /student/dashboard, making them easy to differentiate between. This would be particularly important for when I handed over my project, as it would make it clear to anyone making changes to the app in the future where the code was located.

With my files setup, I added the basic code needed to create a working flask app to the __init__.py file. The next job was to set up my database, which I did using SQLite studio, turning my planned database into a reality. Now I needed to bring the tables over into the models.py file of my Flask app, allowing me to access the data from the tables. I used the Python extension sqlalchemy, rather than traditional SQL statements. This had two advantages; first of all, it would meant that I wouldn't have to write SQL statements like this:

```
SELECT user.user_id AS user_user_id, user.role AS user_role, user.first_name AS user_first_name, user.last_name AS user_last_name, user.email AS user_email, user.form_class AS user_form_class, user.photo AS user_photo, user.total AS user_total
FROM user
WHERE user.email = ?
LIMIT 1;
```

(the code to get a user's info via their email address) and could instead use something simple like `cls.query.filter_by(email=email).first()`. This had the added advantage of also being more in the style of python, which would make this code easier to understand for anyone editing my program in the future.

For each table in my database, I created a Python class in the models.py table that would allow me to access the field from that table. For example, the class for my user table looked like this:

```
class User(UserMixin, db.Model):
    __tablename__ = 'user'
    id = db.Column("user_id", db.Integer(), primary_key=True)
    google_id = db.Column("user_id", db.Integer(), unique=True)
    first_name = db.Column(db.String())
    last_name = db.Column(db.String())
    email = db.Column(db.String(), unique=True)
    role_id = db.Column("role", db.Integer(), db.ForeignKey('user_role.id'))
    is_active = db.Column(db.Boolean())
```

This would allow me to access each field of the database easily. For example, say I had a user loaded called student, I could access their first name with user.first_name. The next step was to add the relationships between the tables to each class, which would allow me to access data from across the tables using my foreign keys.

```
    role = db.relationship("UserRole", back_populates="role_group")
    hours = db.relationship(
        "Log",
        back_populates="user",
        foreign_keys="Log.user_id")
    groups = db.relationship("GroupMembers", back_populates="user")
```

This made it easier to access relevant data from other tables for each user. For example, if I took that user object called student again, student.hours would return a list of the logged hours belonging to that user from the log table, which I could then use on a page which displayed all of a user's logged hours. This meant that I would avoid a cumbersome approach in my code. Without these relationships I would have to get the user's id, store it, then run a whole query of the log table for any items with that user id, request them from the database and then store them. This would require many more lines of code, thus highlighting the benefit of using sqlalchemy.

To increase code reusability, I decided that an object-oriented approach would work best for my website, as there would likely be a significant amount of crossover between different parts of the app. For example, with the user object, on one route a student might want to check their current award and on another, staff might want to see the award which each student currently had. I could then add a User.get_current_award method, which would return the current_award of that user object, which could be used in all locations across the website. This would also make it easier for anyone editing my code in the future, as they would be able to make changes to those methods once and then have them update everywhere they were used in the site.

I undertook some testing after setting up my database, which revealed that I had made a few errors in setting up my database. For example, my association table group_members had the column user_id in its model specified as being an integer (number), when it should have been a string (letters). This produced an error when I tried to add a staff user (who have letters as their school id) as it was not a number. To fix this, I changed the column to be a string in models.py, which would support ids with both letters and numbers.

### Adding Login System

Now that I had set up my database and basic Flask app, it was time to add the ability for users to log into the system. I had decided to use the Google Sign-In API earlier, which would allow users to log in with their school Google Account. To allow users to do so, I needed to create a Google Cloud Platform account and add my app, which gave me the security credentials I'd need to access the info of users signing in with their Google account. Along the way, I discovered that I could also retrieve the profile pictures of users. This would allow for further personalisation, and so I thought that it could be worth recording the URL of these. To do so, I added a new column to my user table, photo, which would store this URL.

I then needed to come up with a system to allow the website to only be accessed by Year 13 students and staff. Checking that the domain of the user's Google Account was [redacted]school.nz would not work as this would let all year levels of students access the database, so users would instead have to be pre-added to the database. Then, when users tried to log in, the program would check if their account was in the database, and if it was, let them in. I would set up a system for uploading users to the system in bulk later on when I developed the staff portal.

Now that this was done, I set up the routes that would handle the users signing in with Google. I also had the database update the name and profile picture of users each time they logged in to keep them up to date with any changes that might occur. This was how my login system would function:

1. User presses login button and is redirected to Google Sign-In Page

2. Google Sign-In returns users to the application with a token indicating their account and a successful login. The program would then send this to google to confirm that it was valid (and not just entered by the user) and then return the user's info
3. The application will check the user's email address against a list of students/staff in the database who have been granted access to the app
4. If the user has been granted access, they will be logged into the system, otherwise if they have not they will be redirected back to the login page with an error message.

I then translated this into Python code. One issue that arose was that I needed to set various variables in my program to let Google know what my app was, which would need to be stored securely and not kept in my code. If these variables were stored in my code, then anyone who had access to it (especially if it was publicly available on GitHub which I was using for version control) would be able to impersonate my app to Google. To address this, I used the os module and its environ function, which would allow the code to access variables set in the terminal by the user running the program. As such, the variables would not be stored in my code and would have to be entered by the user when they launched the program on the server where it was being run.

```
# get the various info we need from the environment for security purposes
GOOGLE_CLIENT_ID = os.environ.get("GOOGLE_CLIENT_ID")
GOOGLE_CLIENT_SECRET = os.environ.get("GOOGLE_CLIENT_SECRET")
```

```
$env:GOOGLE_CLIENT_SECRET='xxxxxxxxxxxxxxxxxxxxx'
```

To have the program check that the user had been pre-added to the database, I needed to add the first method to the user class, load_by_email(). This would query the database for an existing user with that particular email, returning the user if they existed or otherwise returning None. I also had the program update the name and profile picture of users each time they logged in in the database to keep them up to date with any changes that might occur (e.g. a user changing their Google profile picture).

```
user = User.load_by_email(users_email)
    if user:
        # updates the user's record with their info from Google in case
        # anything e.g. picture has changed
        user.update(first_name, last_name, picture)
```

```
@classmethod
    def load_by_email(cls, email):
        """Loads a user from the database using their email."""
        return cls.query.filter_by(email=email).first()
```

### Adding Jinja2 Templates

With a basic app, functioning database and user system, I now wanted to set up the Jinja templates that I would use throughout the app. I'd already put together some HTML for my site when planning my UI, so I started by separating this out into different sections that could then be imported into the required files. The first layer was base.html, which imported stylesheets and javascript required across all parts of the website,

*Excerpt from base.html*

```
<body>
    {% block layout %}
    {% endblock %}
```

I then took the HTML for the menu bar and added it to a template titled app_container.html. This extended the first template, base.html, that I had created, filling in the layout block.

*Excerpts from app_container.html*

```
{% extends "base.html" %}

{% block layout %}
<div class="container-fluid">
```

Although this menu bar would be present on both the staff and student portals, they would have slightly different pages, so I added a menu block, which would then allow for the menu for each portal to be imported from a different HTML file.

```
<ul class="nav flex-column mb-sm-auto mb-0 align-items-center align-items-sm-start"
id="menu">
                    {%block menu %}{% endblock %}
                </ul>
```

I also added the various variables needed in the menu bar e.g. {{user.photo}}, {{user.first_name}}, which would now dynamically update depending on the user logged in.

```
<a href="#" class="d-flex align-items-center text-white text-decoration-none
dropdown-toggle" id="dropdownUser1" data-bs-toggle="dropdown" aria-expanded="false">
                        <img src={{user.photo}} alt="Profile Photo" width="30" height="30"
class="rounded-circle">
                        <span class="d-none d-sm-inline mx-2">{{user.first_name}}</span>
                    </a>
```

Finally, I added a content block, where the contents of each page would be placed.

```
{% block content %}
{% endblock %}
```

### Adding Student Dashboard Page

With my templates set up, it was time to make the widgets I'd developed earlier dynamic based on the total hours logged by the student. This created a range of issues. I started by defining a route for the student portal. This route would define which template to return based on which URL the user had navigated to. @login_required ensured that this page was only accessible to logged in users.

```
@student.route('/dashboard')
@login_required
def dashboard():

    return render_template(
        "student/dashboard.html",
        user=current_user)
```

I then needed to set up the template for the route, dashboard.html. This would extend the app_container.html template I had set up earlier in the development process. I then needed to specify the menu to use, the title of the page (the name that would be displayed in the browser tab), before then adding the main content of the page. This process would be the same for all page templates that I would add to the project.

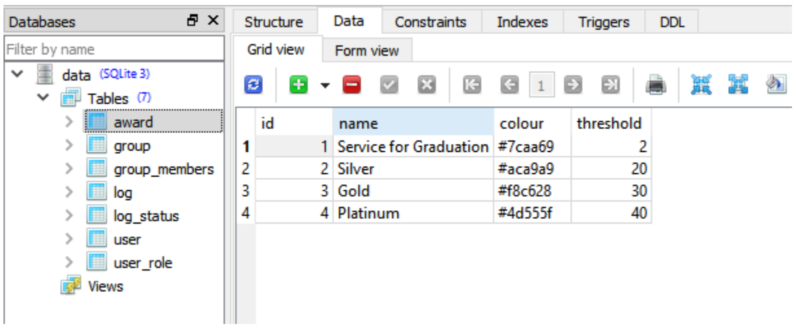
```
% extends 'app_container.html' %}

{% block menu %}{% include 'student/menu.html'%}{% endblock %}

{% block title %}Dashboard{% endblock %}

{% block content %}
...
```

To find the current award of the user for use in the widgets, I needed to find a way of calculating the total hours of students. I considered when I should calculate the totals of users both overall and for the groups they were in (either when new hours were added, requiring them to be stored, or adding them up whenever they were requested). I talked to Mr D[redacted] who informed me that calculating these at the time when new hours were entered or altered and then saving them would be the way to go. This would mean that if errors occurred, they would occur at the time that hours were added, rather than each time a total was requested (which would occur if I did not save the totals). This meant that I needed to alter my database, so I added columns to both my user table and my group_members table. I updated my models.py file accordingly.

```
total = db.Column(db.Numeric())
```

I realised that I need to use the db.Numeric data type for this column, which would support decimals, as my system would need to support students logging service which took less than an hour. For example, if a student helped a teacher with an odd job for 30 minutes, then they would need to be able to add 0.5 of an hour. As my new system needed to allow all kinds of service to be recorded, enabling these smaller or partial hours to be logged would help to ensure that all service was recorded.

Now that I had access to the total hours of students, I needed a way to find out which award a user currently had, and how many hours they needed to earn to get their next one. I wanted to avoid hardcoding the awards into the code e.g. where the system used something like if the user's total >= 40, current award is platinum, as this would make it difficult to change the requirements of the awards if they were ever altered in the future or more were added. I realised that I needed to add a new table to the database that would store this information, and thus allow for changes to be made to the awards in the future. As such, I added an awards table into my database, data.db in SQLite Studio and updated my models.py file with the new table.



| | id | name | colour | threshold |
|---|---|---|---|---|
| 1 | 1 | Service for Graduation | #7caa69 | 2 |
| 2 | 2 | Silver | #aca9a9 | 20 |
| 3 | 3 | Gold | #f8c628 | 30 |
| 4 | 4 | Platinum | #4d555f | 40 |

```
class Award(db.Model):
    __tablename__ = "award"
    id = db.Column(db.Integer(), primary_key=True)
    name = db.Column(db.String())
    colour = db.Column(db.String())
    threshold = db.Column(db.Integer())
```

Now that I had access to the various awards and a user's total, I needed to create a method for the user object that would find that user's current and next award. I quickly realised that I needed another method to do so, one which query the database for a list of all the awards, in ascending order so that I could iterate over them to find the highest award that a user had met the threshold for.

```
@staticmethod
    def get_awards():
        """gets all awards stored in the database sorted in order of size ascedning"""
        return Award.query.order_by(Award.threshold).all()
```

I then created the User.get_current_award() method as expected, which found the award via iterating over the list of awards as mentioned earlier.

```
def get_current_award(self):
            """takes the current amount of hours and returns an award object with the current award
earned by the user."""
        awards = Award.get_awards()
        current_award = None

        # compares the users total with each award, giving us the max award
        # they've gotten
        for i in range(len(awards)):
            if self.total >= awards[i].threshold:
                current_award = awards[i]

        return current_award
```

Adding the get_next_award function was relatively simple - I would just need to call the get_current_award function to find the current award, find what place it was in the list, and then find the award that was located after this one. However, I ran into errors when a user didn't have an award or had the maximum award with this approach - either there was no current award so it was unsure which award was next in the list, or there was no award in the list after the current one. This resulted in errors being thrown, so I had to add some additional code to catch these edge cases.

This was achieved by checking if the index of the current award in the awards list was equal to the length of the list, in which case the maximum award had been found and None would be returned. Likewise, if there was no current award, the first item from the awards list was returned instead.

```
            # check if there is actually another award for them to get or if
            # they've maxed out
            if index < len(awards) - 1:
                next_award = awards[index + 1]
            else:
                next_award = None
        else:
            # they've got no award, so it must be the first one
            next_award = awards[0]
```

With both of these functions returning an award object with all its accompanying information, it was time to use this info to create the dynamic widgets for the student dashboard.

I added code to my Jinja 2 templates that would dynamically change based on which award a user had/was next or if there was no award for either of these.

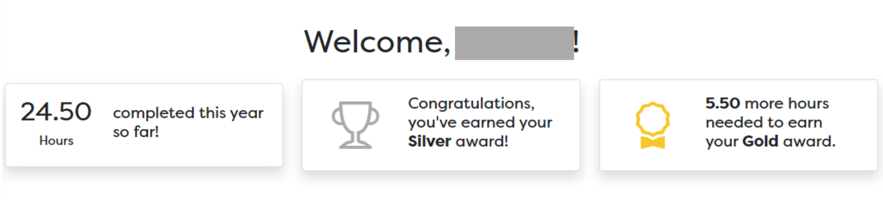
*Code for next award widget*

```
{% if next_award != None %}
        <i class="bi bi-award dash-icon" style="color: {{next_award.colour}};"></i>
</div>
<div class="col-md-8">
        <div class="card-body p-0">
                <h5  class="card-title"><strong>{{'%0.2f'  |  format(next_award.threshold  -
current_user.total)}}</strong> more hours needed to earn your <strong>{{ next_award.name
}}</strong> award.</h5>
               {% else %}
                 <i class="bi bi-check-square dash-icon" style="color: green;"></i>
```

```
              </div>
              <div class="col-md-8">
                <div class="card-body p-0">
                      <h5 class="card-title">Well done, you've earned the highest level of
award possible!</h5>
              {% endif %}
```

I ran into an issue where, because the user's total hours was stored as a decimal, the numbers on the dashboard would be displayed to a ridiculous number of decimal places e.g. 24.50000000000000., hence why I had to use `{{'%0.2f' | format(next_award.threshold - current_user.total)}}`, which rounded the numbers to two decimal places for display.

With all this completed, the dashboard page now functioned as expected.


Welcome, [redacted]!

24.50 Hours completed this year so far!

Congratulations, you've earned your **Silver** award!

**5.50** more hours needed to earn your **Gold** award.

*Thorough testing of the dashboard widgets can be found in the testing appendix.*

## Adding the Add Hours Form

The next feature to add was the ability for students to log their hours. I began by making some refinements to my database. I decided that it was no longer necessary to record both a title for the service and a description of what was being logged as per my ER diagram, as these would likely be very similar and thus just be more hassle for students (which I very much wanted to avoid!). Finally, I added a new column to the log table called log_time, which would record when the service item was logged. As I was now spending more time thinking about how staff and students would use the website, I realised that some staff might want to check hours that had been added recently (if they were regularly going through and checking them), which might not necessarily correlate with when the service was completed if students were not logging their work as they went. Recording when the hours were logged would allow staff to sort through the hours in this way, making it easy for them to check hours that had been recently added or since they had last checked.

I decided to use WTForms, another Python module, to create my forms, which would make it easier to add form validation and create form HTML, taking some of the work out of the form. I created a forms.py file to put this form into, setting out the various fields which would need to be filled in to match up with the fields in my database.

```
class AddHours(FlaskForm):
    description = TextField('Description', validators=[DataRequired(), ])
    hours = DecimalField(
        'Hours',
        validators=[
            DataRequired(),
            ])
    group = SelectField("Group", validators=[DataRequired()])
    teacher = SelectField("Teacher", validators=[DataRequired()])
    date = DateField(
        'Date Completed',
        format='%d/%m/%y',
        validators=[
            DataRequired()
    submit = SubmitField("Log Hours")
```

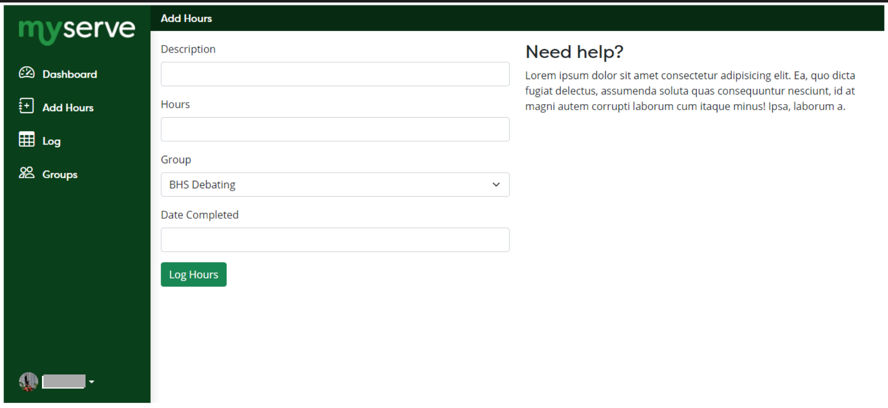
I then created a route for this page in the students.py file. As I was doing so, I encountered a number of issues.The first one was that I needed to find a way of adding the choices to the dropdown menus for each user's groups and all the teachers in the school. Normally, you would specify these choices in the forms.py file when setting up the form field, but as they would differ for each student, I would need to update them in the students.py file. I found that I could do this by specifying the .choices attribute of these form fields in the app route, which then required two new methods to be added to the User object, one which would get all of the groups a user was in and return it in the (group id, group name) tuples that WTForms required. I needed to then add a final option to the choices list for no group for instances where students completed service outside of a defined group.

```
form.group.choices = current_user.get_group_options() + \
        [(None, "No Group")]
form.teacher.choices = User.get_teacher_options()
```

```
def get_group_options(self):
"""returns a list of the groups a user is in as (id, name) tuples for use in forms"""
       return [(group_assoc.group.id, group_assoc.group.name)
              for group_assoc in self.groups]
@classmethod
def get_teacher_options(cls):
"""returns a list of all teachers in the school as (id, name) tuples for use in forms"""
       return [(teacher.id, f"{teacher.first_name} {teacher.last_name}")
             for teacher in cls.get_teachers()]
```

Now that the form was set up, it was time to sort out its appearance on the add_hours.html template. WTForms made this really easy, as I could just use Jinja to specify which form fields went where, and WTForms would then automatically add the required HTML. I then specified the Bootstrap classes needed to make the form appear properly.

```
<div class="mb-3">
    {{ form.description.label(class="form-label")}}
    {{ form.description(class="form-control") }}
    ...
```



Add Hours

Description

Hours

Group

BHS Debating

Date Completed

Log Hours

Need help?

Lorem ipsum dolor sit amet consectetur adipisicing elit. Ea, quo dicta fugiat delectus, assumenda soluta quas consequuntur nesciunt, id at magni autem corrupti laborum cum itaque minus! Ipsa, laborum a.

The next job was to add the dropdown field to select a teacher for instances where service was completed outside of a particular group. I need this field to only show up when the No Group option was selected, and then disappear again if the user decided to select a group. This would need to be done with some Javascript and jQuery, which would detect when there was a change in the group element and then hide or show the teacher question.

```
$('#group').on('change',function(){
       if( $(this).val()==="None"){
            $("#teacher").show()
            }
            else{
            $("#teacher").hide()
            }
       });
```

WIth the form now displaying correctly, it was time to add the backend code to process the form data once it was submitted. This was added to the route, where after checking that the form data was valid, would be then sent to be validated.

```
 if form.validate_on_submit():
        # send the form data to the database to be recorded
        Log.add_hours(
        ...
```

I added a new method to the Log object that would add these hours to the table. This began by creating a new instance of the Log object and adding it to the database. I then needed to add the hours from the newly logged item to the total, as I had decided upon earlier.

```
@classmethod
def add_hours(cls, user, group_id, teacher_id, time, description, date):
        """creates a new log object in the database to record the hours of users"""
        new_hours = cls(
            user_id=user.id,
            group_id=group_id,
            time=time,
            description=description,
            date=date,
            log_time=datetime.now(),
            status_id=1
        )
        db.session.add(new_hours)

        # we need to update the user's total
        user.total += new_hours.time
```

I then also needed to update the total for the user's group if the user had specified a group, which I did by uploading their group member object and then updating the total. I also only added the teacher_id if there was no group specified, and set the group_id to none if this was the case. If tech-savvy students altered the HTML in their browser to stop the student field from disappearing, they could have been able to submit both a group and a teacher, so ensuring that the group id was set to None would avoid this issue.

```
        # if the hours were done in a group, then we need to udpate the group
        # total
        if group_id != "None":
            group_member = GroupMembers.load(
                user_id=user.id, group_id=group_id)
            group_member.group_hours += new_hours.time
        else:
            new_hours.teacher_id = teacher_id
            new_hours.group_id = None

        db.session.commit()
        return
```

## Adding Form Validation

Now that my form was able to handle expected input, I needed to add handling so that the form could recognise invalid input and help users address the errors in what they had entered. This involved adding validators to each form field in the forms.py file. WTForms would already check that the fields were in the right format e.g. that the data in a TextField is text, but I need to add a few custom ones to make the form function as expected and ensure the integrity of the data in my database.
*In-depth information about the various validation requirements and their accompanying error text can be found in the testing appendix.*

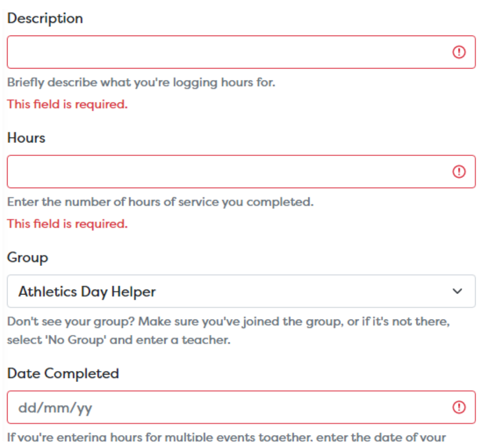
Although most of the validators were able to be created using those built into WTForms, I had to create a couple of my own. One of these in particular was a validator to check how many decimal places a user had entered the number of hours of service completed. As aforementioned, I wanted to support users adding partial hours and therefore decimals, but wanted to limit the number of decimal places that could be entered to ensure that they always displayed nicely e.g. 12.5963923409 would be long and mess up the formatting on my website in instances where the total was displayed. I settled on two decimal places as a good balance between accuracy and display convenience, as this would support 15-minute (0.25 hrs) and 20-minute intervals (0.33 hrs). To find the number of decimal places, I had to write a function that reversed the number, found where the period was and thus checked whether the number met the validation criteria.

```
def validate_legnth(form, field):
    """checks that the user has not entered an hours amount to more than two d.p."""
    hours = str(field.data)
    if hours[::-1].find('.') > 2:
        raise ValidationError(
            "Hours amounts cannot have more than two decimal places.")
```

Now that my app was validating and returning helpful error messages, I needed to have these displayed to the user when the page was reloaded. I returned to my templates to add this functionality.

```
{{ form.description(class="form-control" + (" is-invalid" if form.description.errors else
""), aria_describedby="descriptionFeedback") }}
{% for error in form.description.errors %}
      <div id="descriptionFeedback" class="invalid-feedback">{{ error }}</div>
{% endfor %}
```

Displaying the errors was simple enough as it just required the errors attribute to be iterated through, but styling the error messages proved to be quite complicated, as I needed to add the is-invalid Bootstrap class to the fields to make them pop up red, which would make it easier to identify where the errors on the form were. I managed to make this work with an in-line if statement, which turned the boxes read, however, the error messages were not appearing. It turned out that Bootstrap needed the error messages linked with the form fields via the aria-described-by attribute for them to be displayed, so I added these matching attributes to the template.


Description

Briefly describe what you're logging hours for.
This field is required.

Hours

Enter the number of hours of service you completed.
This field is required.

Group
Athletics Day Helper
Don't see your group? Make sure you've joined the group, or if it's not there, select 'No Group' and enter a teacher.

Date Completed
dd/mm/yy
If you're entering hours for multiple events together, enter the date of your

I'd also added some help text to the form to explain what information would go where, but I needed to run this past some students to check if it was clear enough. I was planning on engaging with them after I had added one more feature to ensure that I utilised their time well.

### Adding a Page to Show Student's Logged Hours

Now that students were able to log hours via the website, I needed to add a page that would show them all of their logged hours. A table seemed like the most logical way to do this, as this would display the data in a well organised and clear way, making it easy for students to interpret it and their progress.

I would likely be needing to use tables on many of the other pages on the website, so I decided that it would be sensible to develop a table style that would be universal throughout my site. I also wanted to have tables that were able to be sorted by different columns and searchable. I found the datatables.net library when researching options for producing such tables, which would automatically add the functionality I was after. To add the library, I added the required stylesheets and javascript to my base.html template

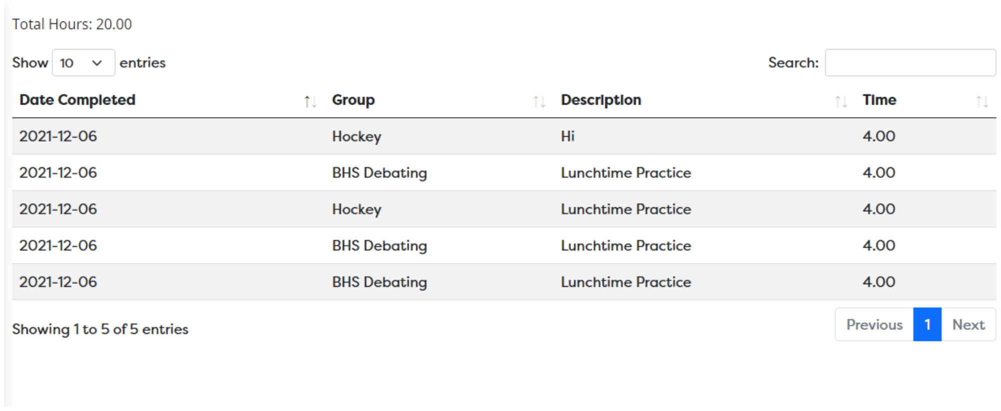
```
<link rel="stylesheet" type="text/css"
href="https://cdn.datatables.net/v/bs5/jszip-2.5.0/dt-1.11.3/b-2.0.1/b-html5-2.0.1/b-print-
2.0.1/datatables.min.css"/>
<link rel="stylesheet" type="text/css"
href="https://cdn.datatables.net/1.10.25/css/dataTables.bootstrap5.min.css"/>
<script type="text/javascript"
src="https://cdn.datatables.net/v/bs5/jszip-2.5.0/dt-1.11.3/b-2.0.1/b-html5-2.0.1/b-print-2
.0.1/datatables.min.js"></script>
```

From the backend, this was quite a simple route - I only needed to pass the current_user object as the logs of students could be accessed through the User.hours relationship with the log table. It was then just a matter of adding rows to the table by iterating through User.hours and then populating these with the data from the log object.

```
    {% for item in user.hours %}

        <tr>
            <td><p>{{item.date}}</p></td>
            <td><p>{{item.description}}</p></td>
            {% if item.group == None %}
            <td><p>Other</p></td>
            {% else %}
            ...
```

A small script was then all that was required to activate the javascript needed for the interactive table.

```
<script>
$(document).ready(function() {
    $('#log').DataTable();
} );
```

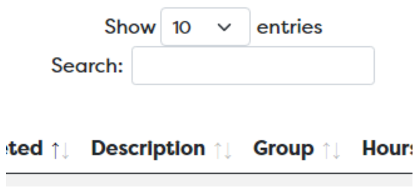

Total Hours: 20.00

Show 10 entries                                                    Search:

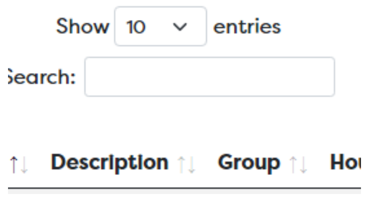
| Date Completed | Group | Description | Time |
|---|---|---|---|
| 2021-12-06 | Hockey | Hi | 4.00 |
| 2021-12-06 | BHS Debating | Lunchtime Practice | 4.00 |
| 2021-12-06 | Hockey | Lunchtime Practice | 4.00 |
| 2021-12-06 | BHS Debating | Lunchtime Practice | 4.00 |
| 2021-12-06 | BHS Debating | Lunchtime Practice | 4.00 |

Showing 1 to 5 of 5 entries                                        Previous 1 Next

There was one annoying issue that I encountered with the library when testing - on smaller screens, there was no spacing between the dropdown menu and the search bar when they adjusted to be on top of each other on smaller sized screens. This not only looked bad, it also made it really easy to tap on the wrong box on touch screen devices as they were not far enough apart, which was frustrating and would be for students logging their hours on a phone. I added some custom CSS to override that of the datatables.net library, which fixed the problem.



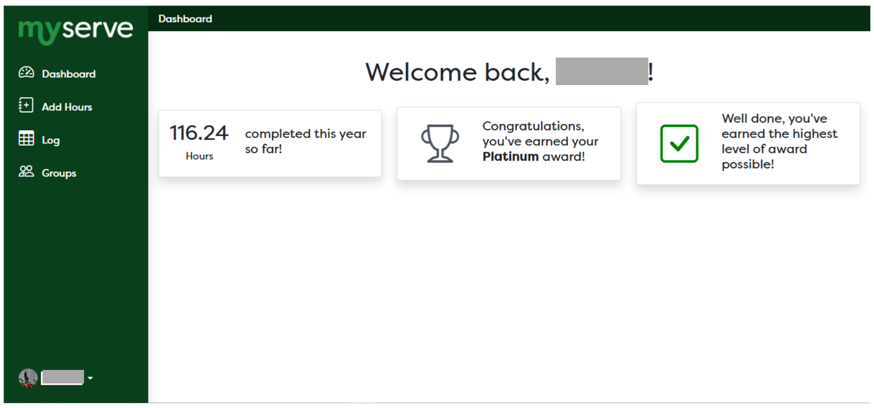
```
.dataTables_length {
    margin-bottom: 0.5rem;
}
```



## Feedback and UI Adjustments

Now that I had some functioning parts of the website, I decided it would be a good time to take the website to Mr D[redacted] and a few students and hear their thoughts. An issue that Mr D[redacted] noticed with my website was that there wasn't any indication of the page which users were on. As well as this, I also felt as though the menu bar looked disconnected from the rest of the site, so I added a green bar across the top of the page with the name of the page that the user was on. I used the ticky-top class to keep the bar at the top of the page, even when scrolling, and the same title block that was used in the tab name to display the name of the page.

```
<nav class="navbar navbar-dark bg-title px-3 py-0 sticky-top">
      <div class="container-fluid">
              <span class="navbar-text text-white">
                    {% block title %}{% endblock %}
              </span>
       </div>
</nav>
```



Welcome back, [redacted]!

116.24 Hours completed this year so far!

Congratulations, you've earned your **Platinum** award!

Well done, you've earned the highest level of award possible!

[redacted]

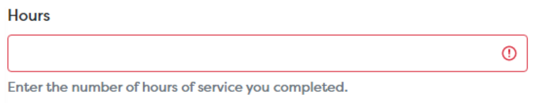
From the group of students I initially interviewed, I asked four of them to engage with me regularly throughout the project. This included two students who identified that they were heading into employment next year. As I identified earlier, the primary goal of my project was to help these students gain service awards and a record of volunteer work as they stood to gain the most from them. I also included two students who were intending to study at university next year. While these students did not stand to gain as much from the new system, it was still important that my project also worked for these students.

The first thing the students noted was that the transitions between pages (or lack thereof) were quite jarring, particularly as the app began to look and function less like a website and more like an app. To combat this, I added fade transitions that were triggered using jQuerry to the base.html file when clicking a link to a different page and loading a page, which made the white section (not the side/top bars) fade to white and then to the new page. This made the website feel smoother and more app-like.

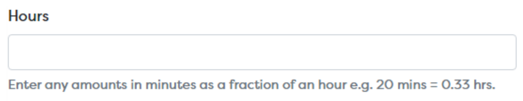
```
// delegate all clicks on "a" tag (links)
      $(document).on("click", "a", function () {
          // get the href attribute
          var newUrl = $(this).attr("href");

          // veryfy if the new url exists or is a hash
          if (!newUrl || newUrl[0] === "#") {
              // set that hash
              location.hash = newUrl;
              return;
          }

          // now, fadeout the html (whole page)
          $("#content").fadeOut(200, function () {
              // when the animation is complete, set the new location
              location = newUrl;
          });

          // prevent the default browser behavior.
          return false;
      });
```

In the above code, I added the #content id tag when specifying what items I wanted to fade out. I found that fading the whole page, which included the menu bars, looked weird as the menu bars would appear in exactly the same place when the pages faded back in. As such, I added the #content id tag and added a div with this id around the content block in the app_container.html code. This would mean that the content of the pages would fade in and out, but the menu bar would stay in the same place when switching places. This polish made the site feel less website and more app-like as transitions are not generally a part of most websites. While this was a small change, making the app visually appealing would encourage, or at the very least, not put off students from using the app.

I had identified earlier that I wanted to run the help text I'd written for my form to add hours past these students, and I did so when I spoke with them. The majority of the help text was fine, but there were a couple of the fields which students had trouble with when I got them to test it. The first of these was for the hours field. I hadn't actually specified how to enter part hours (minutes) as decimals in the help text and as such, the students were confused when it came to do so. As such, I changed the help text so that it explained how they should do so, providing an example for further clarity.

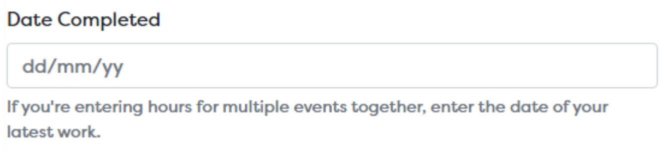

Hours

Enter the number of hours of service you completed.


Hours

Enter any amounts in minutes as a fraction of an hour e.g. 20 mins = 0.33 hrs.

A similar formatting issue occurred with the date field, where students ran into trouble with how they should format the date e.g. dd/mm/yyyyy or dd/mm/yy. This would generally mean that students would have to submit the form, get an error message and then go through a trial and error process to determine the correct format. Obviously, this was a big pain, so I added some preview text to the field which specified the format and then disappeared once they started typing, letting them know how to write the date.


Date Completed

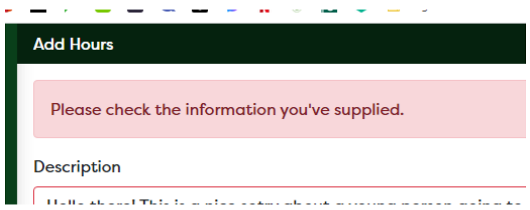
dd/mm/yy

If you're entering hours for multiple events together, enter the date of your latest work.

One final point of feedback which Mr D[redacted] had was that there wasn't any confirmation that hours had been successfully added after the form had been submitted. This would mean that users would either have to go to the log page to check if their hours had been added, or just be generally confused. Confusion was the biggest problem with the current system, and so I very much wanted to avoid this in my new system.

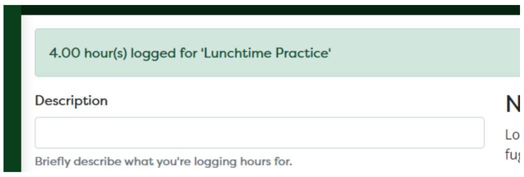
As I was coming up with a solution to this problem, I realised that there would probably be a number of instances where I would want to display confirmation or error messages, and so it would make sense to develop a site-wide system for this. Flask had a flash feature, which would automatically pass error messages to the template engine. I then added some code to my app_container.py which would display them. I had two categories of message, error and update, which would be coloured differently to represent their status.

```
{% with updates = get_flashed_messages(category_filter="update") %}
        {% if updates %}
                {% for msg in updates %}
                ...
                {% endfor %}
        {% endif %}
...
```


Add Hours

Please check the information you've supplied.

Description


4.00 hour(s) logged for 'Lunchtime Practice'

Description

Briefly describe what you're logging hours for.

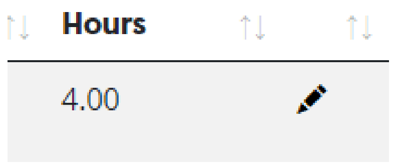
## Adding the Edit Hours Route

Students might make errors when adding their hours and as such, want to edit or delete their previously logged hours. As such, the next route to develop was the one to edit hours. This was the first route in my project to need a variable to be passed in the url, the id of the previously logged item to be edited. The Flask framework then made it easy to pass this id value through to the code for the route.

```
@student.route('/edit-hours/<int:id>', methods=['GET', 'POST'])
@login_required
def edit_hours(id):
...
```

To access this route, I added an extra column to the table in the log template with a pencil icon as a link to the edit hours route, which would pass the id of that log object. The pencil icon was chosen as it is used across the web to represent edit functionality.

```
<a href='{{url_for("student.edit_hours", id=item.id)}}'><i class="bi-pencil-fill"></i></a>
```



| Hours | |
|---|---|
| 4.00 | ✎ |

The next step was to figure out how to have the user edit their previous items. Some research with WTForms revealed that I could prefill fields in the form. Therefore, I thought that the best approach would be to take the form from add hours and pre fill it with the info for that logged item from the database. The user could then edit the information as required and resubmit the form, where it would then be updated in the database. This was relatively easy to implement code-wise, although I ran into a slight error where the group fields were not being pre-selected as they should have been. It turned out that WTForms was converting all of the ids of the groups to strings when it populated the dropdown form as that was standard practice for html, so when I tried to set it to the group id from the database, which was an integer and the wrong data type, it would not recognise it as being in the list. To solve this, I had to convert the group id to a string so that it would be recognised as being one of the options in the dropdown menu.

```
# prefill the form with data from the database
    if request.method == 'GET':
        form.group.data = str(item.group_id)
        form.teacher.data = item.teacher_id
        form.hours.data = item.time
        form.description.data = item.description
        form.date.data = item.date
```

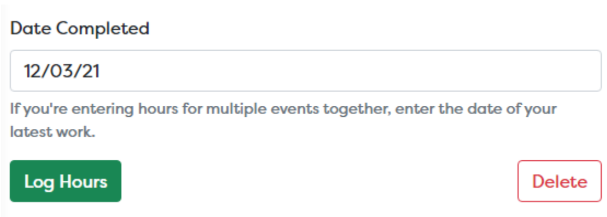
When it came to my templates, I noticed that I would need virtually the exact same HTML for this template as I need for my one to edit hours, so I decided to move the form HTML to a new template called hours_form.html, which the add_hours and edit_hours templates would then extend from, adding the ability for them to have separate help text, different titles and buttons. This meant that I could make changes to the form and they would show up on both pages.

Now I needed to add the server-side code to process the form once it was submitted. This was, for the most part, quite simple, as I would just need to pass the new data from the form and then update each property accordingly. It became more complex when it came to updating the totals, specifically when a user changed a log item from being under a group (which had a total) to being under a teacher (which had no total) and vice versa. I experimented with quite a few different methods of doing this, which tried to check if there had been a change in group and if so, what it was, but I ended up settling on a simpler method, which just removed the hours from the old group, if there was one, and added the hours to the new group (or back to the old one if it was the same). This worked in all cases.

```
        # if they recorded their hours under our group, we'll subtract them
        # from the total incase they've changed group
        if old_group:
            old_group_member = GroupMembers.load(self.user_id, old_group.id)
            old_group_member.group_hours -= old_time

        # if there is a new group, we'll add the hours to its total. If there's
        # now a teacher, then we will record that instead
        if group_id == "None":
            self.teacher_id = teacher_id
            self.group_id = None
        else:
            group_member = GroupMembers.load(self.user_id, group_id)
            group_member.group_hours += self.time
```

The final feature to add to this route was the ability to delete hours items altogether. I was able to do with by adding another submit button to the form in forms.py, which I would be able to check if had been clicked (and had the value True) using the code below:

```
elif form.delete.data:
        item.delete()
```


Date Completed

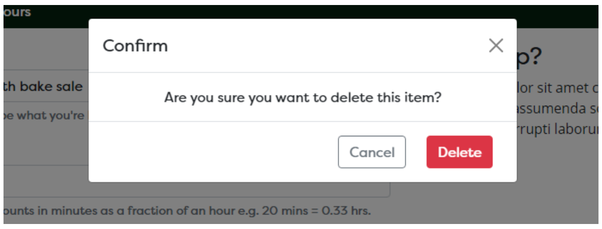
12/03/21

If you're entering hours for multiple events together, enter the date of your latest work.

Then I needed to build the Log.delete() route to delete the hours. This was straightforward, as I just needed to update the user's totals, delete the item from the database and then commit the changes.

```
def delete(self):
        """deletes logged hours"""
        # update the user's total
        self.user.total -= self.time

        # if the hours were in a group, then we need to change that total too
        if self.group:
            group_member = GroupMembers.load(self.user_id, self.group.id)
            group_member.group_hours -= self.time

        db.session.delete(self)
        db.session.commit()
        return
```

With the edit page completed, I ran through it with one of my students to hear their thoughts on it, although I wasn't really expecting a lot in terms of feedback as it was very similar to my add hours form. One thing which we did talk about was users accidentally clicking the delete button. I'd tried to space the delete button far away from the submit button to avoid this, but they noted that people would still do it by accident sometimes. I had a chat with Mr D[redacted] about it as I wasn't sure whether the issue was big enough to justify adding an extra layer of confirmation before deleting an item. He said that it was generally best practice to have some kind of dialogue box when deleting things, so I decided to add a Bootstrap  popup modal which would ask the user to confirm that they wanted to delete the item before deleting it.


Confirm

Are you sure you want to delete this item?

## Adding Pages to View Groups

The final functionality to add to the student portal was that surrounding groups. I started by adding a route to display all of the user's groups and their total hours in a table, which back-end wise was very similar to the log page.

The next route to add was one which displayed a breakdown of the user's service within a particular group. This was again very similar to the log table, with the only new feature needed being a method to find the teachers in a particular group. This required using a join to join the user and group tables, allowing the users in the groups to be filtered by their role id.

```
def get_teachers(self):
            """uses the group ID to return a list of user objects of the teachers in a
particualr group"""
        teachers = User.query.join(
            User.groups).filter(
            GroupMembers.group_id == self.id, User.role_id.in_(
                (USER_ROLE["staff"], USER_ROLE["admin"]))).all()
        return teachers
```

The page which did create some new complexity was the page which allowed users to leave and join groups. I needed a form with a multiple select field for this type of operation, which would allow users to opt in and out of groups in a list that would change over time. WTForms does not natively select multiple select fields, so I had to set up my own.

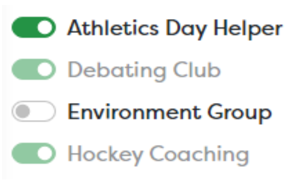
```
class MultiCheckboxField(SelectMultipleField):
    """creates a multiple select that utilises checkboxes"""
    widget = widgets.ListWidget(prefix_label=False)
    option_widget = widgets.CheckboxInput()
```

The form would then be populated with the data of which groups a user was currently in and wasn't. However, I wanted to make it so that users could not leave groups which they had logged hours under. I created a new method for the user class called get_disabled_groups that would do this by cycling through a user's logged hours and adding all the group ids to a set (which would only allow each group id to be added once).

```
group_ids = set()

        if self.role_id == USER_ROLE["student"]:
            for item in self.hours:
                # filter out hours which weren't put in under a group. If
                # they're added we get problems.
                if item.group_id:
                    group_ids.add(item.group_id)
```

I used set logic again to find the groups which the user had joined when they submitted the form, and the ones which they had left, so that these could be updated accordingly.

```
  # we add the disabled groups back in incase the user has altered the html and tried to
leave a group
        # that they've logged hours under already
        form.groups.data.extend([str(group) for group in disabled_groups])

        # time to figure out which groups they've actually left and joined
        groups_join = set(form.groups.data) - set(current_groups)
        groups_leave = set(current_groups) - set(form.groups.data)
```



I also decided to use switches rather than checkboxes for this page. This made it more clear that users were opting in and out of a group, and also made it feel less like a website and more like the settings page of an app.

### Adding Support For Multiple Roles

This marked the finish of my student portal. I planned on running some testing on it with my student stakeholders, but it was a busy period at school for all students with mock exams around the corner and so I delayed this till slightly later when they would have more time. In the meantime, I began work on the staff and admin portal. The first concern with this was that I needed to make it so that staff pages were only accessible by staff, and student pages only accessible to students.

Most inbuilt Flask access level systems that I found relied on the assumption that users with higher access levels simply got access to more pages, but did not lose access to pages. This was not the case with my app, so I had to create my own decorator to check whether the user had the right role.

```
# lets us check whether a user has the right role to be let into a route
def permission_required(role_required):
    def decorator(f):
        @wraps(f)
        def decorated_function(*args, **kwargs):
            if not current_user.is_allowed(role_required):
                return redirect(url_for(current_user.role.name + '.dashboard'))
            return f(*args, **kwargs)
        return decorated_function
    return decorator
```

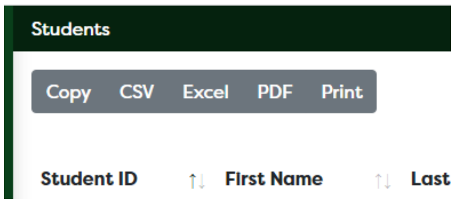
This was then slotted in for each route; the code below the decorator would only be executed if the conditions of the decorator were met i.e. the user had the right role for that page.

```
@student.route('/groups/edit', methods=['GET', 'POST'])
@login_required
@permission_required(USER_ROLE["student"])
def edit_groups():
```

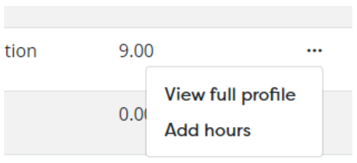
### Adding Page to See All Students

I was now ready to begin building the staff routes. By this stage, I'd created the templates for the staff menu and the dashboard routes, however these were very similar to their student counterparts so were not worth documenting (testing on these can be seen in the appendix).

The next route to add was one which displayed all of the students in the database. This would also be the page which Mr R[redacted] would need to export to be able to use the service award data of students in other applications. Thankfully, the datatables library supported exporting the data from tables client-side, which meant that I did not need to do any extra coding! A range of export buttons were added along the top of the page to add this functionality.



I then hit a predicament surrounding whether I should give teachers the ability to log hours for students. I was adding dropdown menus next to each student, giving the option to view their full profiles, and wondered if I should add a link to add hours to students. I initially decided that I did not want to give teachers the ability to do this as the lack of clarity around whose job it was to record hours had been one of the reasons why the current system was so confusing. I thought that making it so that only students could log hours would increase clarity and ensure that no student who was recording hours would get confused. In my next prefect meeting with Mr R[redacted] I mentioned this to him and he agreed with my position.

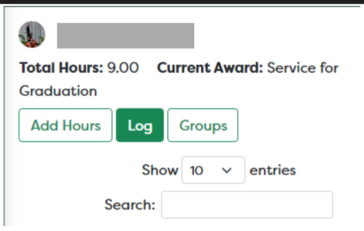


However, I was talking to Mr D[redacted] about this and he disagreed. He noted that while my concerns were true, having teachers give students hours for one-off jobs would feel like a reward for students, and would ensure that they were recorded for students who were less likely to do so. He also pointed out that if teachers recorded them once or twice, then students would see that the work they had done counted as service and be more likely to add them in the future. Mr D[redacted] had made a great point and afterall, the point of my project was to encourage all kinds of service to be recognised, which having teachers get students started with recording hours would encourage. Therefore, I decided to add this functionality to the staff portal, albeit with a warning message stating that teachers should encourage students to record their own hours where possible.

### Add Student Profile Routes

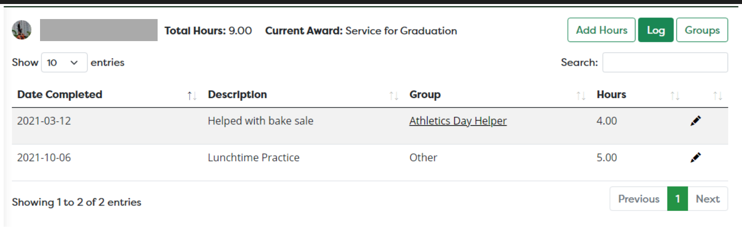
The next step was to add routes which would give a breakdown of each student, their logged hours and their groups. This was, again, another table centric page which did not use an extensive amount of new code. One thing that did require some attention was making the top info and buttons responsive and aligning them all in the middle. This proved challenging and I ended up having to use a lot of <span> objects and Bootstrap classes to achieve the layout below that would work on both mobile and desktop devices.

```
<div class="row align-items-center mb-3 g-2">
    <div class="col-md-auto">
        ...
    </div>
    <div class="col-md-auto">
       ...
    </div>
    <div class="col-xl text-xl-end">
        {% block buttons %}{% endblock %}
    </div>
</div>
```



[redacted]

**Total Hours:** 9.00 **Current Award:** Service for Graduation

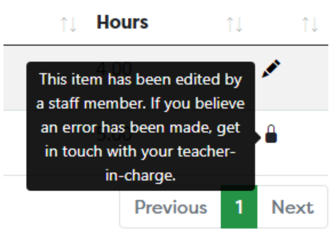


[redacted] **Total Hours:** 9.00 **Current Award:** Service for Graduation

| Date Completed | Description | Group | Hours | |
|---|---|---|---|---|
| 2021-03-12 | Helped with bake sale | Athletics Day Helper | 4.00 | ✎ |
| 2021-10-06 | Lunchtime Practice | Other | 5.00 | ✎ |

Showing 1 to 2 of 2 entries

In the dropdown menu next to each logged item, I added the ability for staff to edit groups, again using much of the same code as I had on the student portal. One change that I did make was that I had the form alter the status of the logged items (finally using the status table in the database) so that students could not try to revert any changes staff had made. I then went back to the student portal and added padlocks where the pencil icons would be to represent that these items would no longer be editable. I also added a message that appeared on hover explaining why these items were locked so that students would not be confused.



This item has been edited by a staff member. If you believe an error has been made, get in touch with your teacher-in-charge.

The student profiles also had a tab for the student's group and a page for a detailed breakdown of each group, which was also essentially the same as the student routes. A slight difference was that the group breakdown required passing two variables to the route instead of just one.

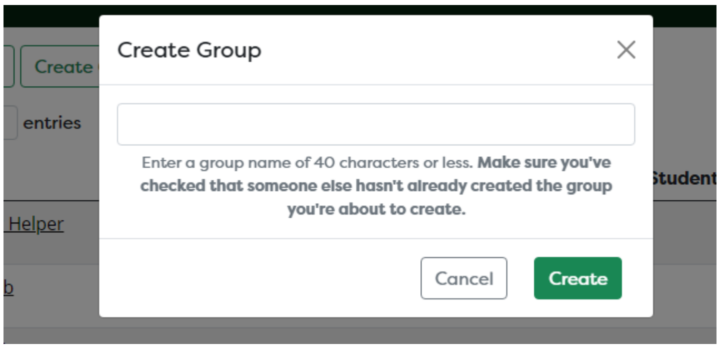
```
@staff.route('/students/groups/<int:student_id>/<int:group_id>')
@login_required
@permission_required(USER_ROLE["staff"])
def student_group_detail(student_id, group_id):
```

Consequently, I realised that I needed to be verifying the variables which were being passed to the server via the URL, as altering these variables would cause the website to crash (as mischievous students might do). I added this to each route in the website which had variables passed to it. In the code below, for example, the server checks that there are actually groups and students in the database with the ids provided, and then checks that the user with the specified user id is actually in the group they are looking for a detailed log of. If not, they are redirected back to their dashboard with an error message.

```
    student = User.load_by_id(student_id)
    group = Group.load(group_id)
    if not group or not student:
        flash("Whoops! That page doesn't exist.", "error")
        return redirect(url_for('staff.dashboard'))
    elif group not in student.groups_proxy:
        flash("Whoops! That page doesn't exist.", "error")
        return redirect(url_for('staff.dashboard'))
```

### Add Group Joining/Leaving/Creating for Staff

These routes were very similar to that of students. There were a few key differences however, one of which was that staff had the ability to create groups via a button which triggered a pop-up modal (similar to those used for confirmation messages). When submitted, this mini-form would create a new group in the database and add the user to the group.


Create Group

Enter a group name of 40 characters or less. **Make sure you've checked that someone else hasn't already created the group you're about to create.**

```
if request.method == "POST" and form.validate_on_submit():
        new_group = Group.create(form.name.data)
        current_user.join_groups([new_group.id])
```

I also added the ability to leave/join groups much like students, except rather than having this be determined by logged hours, I had it be determined by how many other teachers were in the group. There needed to be at least one staff member in charge of each group at a time to verify hours (or pretend to), so any groups where the teacher was the only teacher managing it would be disabled and unable to be left.

```
            for group in self.groups_proxy:
            # check if they are the only teacher in the group
            if len(group.get_teachers()) <= 1:
                group_ids.add(group.id)
```

I also added the ability for staff to remove users from groups. This was quite straightforward code as it just reused class methods I had already written. First, it would go through a list of all of a user's hours from a particular group and remove them, and then remove them from the group (like the edit groups page did).

```
def remove_user(self, user):
        """deletes all of a user's hours from a group (if student), then removes
them."""
        if user.role_id == USER_ROLE['student']:
            for item in self.get_user_log(user):
                item.delete()
        user.leave_groups([self.id])
        return
```

I also added the ability for groups to be removed all together. Once again, this just reused methods I had already written, as it would get all the students, remove them and their hours as per above, do the same for teachers and then delete the group all together.

```
# remove all the users in the group
    for student in group.get_students():
        group.remove_user(student.user)

    for teacher in group.get_teachers():
        group.remove_user(teacher)

    # delete the group
    group.delete()
```

I also added a page that would display all of the hours where students had specified that the teacher logged in had indicated that they oversaw their work which fell outside a group, allowing them to easily access and verify this. Once again, this just used previously developed methods and table structures.

## Adding Ability to Upload and Remove Users

This was the final feature to add (and perhaps the most complicated). As my database checked users logging in against those already in the database to see if they had been granted access, I needed to add a way for Mr R[redacted] or other staff to do this. I went and talked to Mr R[redacted] about what would be the easiest way for this to be done. He said that many of the school apps seemed to just have the user lists updated automatically or via IT, so I should look into this as a way of managing it. This would require setting up a system where the database was automatically updated via an API as new staff/students are added to our school's KAMAR database. I discussed this option with Mr D[redacted] my technical stakeholder, who noted that while this system would be ideal, it would be time-consuming and difficult to set up within my time frame. He suggested setting up a system where users could be added/removed in bulk by the deans via uploading a spreadsheet. I went to visit Mr Roberton to check if this would work for him, and while he did feel that an automated system surrounding user entry would be better, he said that he would manage and agreed that this was not worth pursuing if it would take up a lot of time. More so, he said that it was quite easy to export student info from KAMAR anyway, so this wouldn't be too much of a hassle for him, especially when compared to the current system he was using for service hours.

I started by setting out what data I'd need to be uploaded to the database in the spreadsheet. I knew I'd at least need the id of the user, their role and their form class if they were a student. Our school assigned emails by just adding "@[redacted]school.nz" onto the end of each person's id, so I could infer emails from the id easily without needing an email column in the spreadsheet. I tossed up regarding whether or not I'd need the names of users to be uploaded as these would be updated once they logged in with Google for the first time. However, I found that they would then show up as having the name "None None" on the website, which also meant that staff would be unable to search them up or give them their first hours in this way. As such, I added the first name and last name columns to the database. The next consideration was what kind of spreadsheet file I'd require to be uploaded. I briefly looked into excel files, but there were such a variety of file types which weren't all supported by the same modules and other difficulties, so I decided to use .csv files instead, which would be able to be exported from Excel or Google Sheets (which Mr R[redacted] would be using) anyway. I created a template .csv file and added this to the page so that it could be filled out easily.

I used a form to upload the .csv files, which WTForms proved helpful as it had built in functionality for checking that files were of the correct type. I then added code so that the uploaded file would be opened, split into lines and then each line split into a list of items.

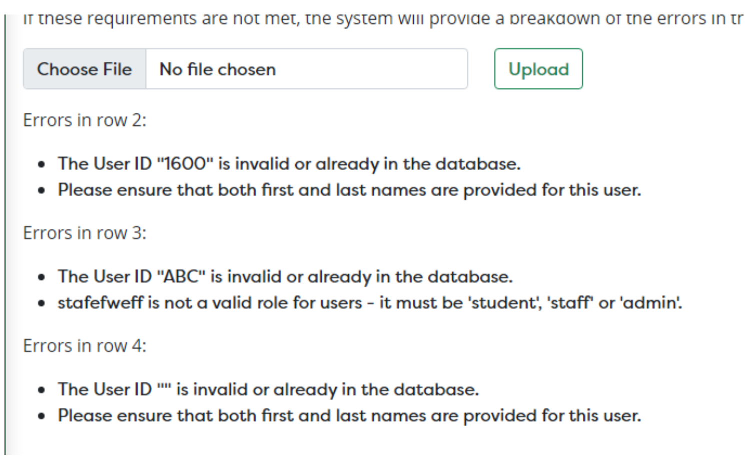
```
                if len(row_data) == 5:
                    # validate the user from the csv
                    new_user = User.enroll_new_user(
                        row_data[0], row_data[1], row_data[2], row_data[3], row_data[4])

                    # if new_user is a list, it means we've got a bunch of
                    # errors and shouldn't upload the file
                    if isinstance(new_user, list):
                        file_valid = False
                        errors.append((line_no, new_user))
                else:
                    file_valid = False
                    errors.append(
                        (line_no,
                         [f"The user at line {line_no} was missing required information.
Please ensure that you have at least entered the user's ID, email and role."]))
```

The data in the list then needed to be extensively checked before new users were added to the database. I also wanted to be able to provide users uploading new users with extensive information about any issues in the file, so that they could easily fix these. I came up with a system where if the right number of fields had been filled out, the program would then check each piece of data supplied to see if it was correct. It would run through each item, adding an error message to a list if there was a problem which would then be returned at the end. However, if there were no errors in the row of data sent through, a new user object would be added to the database, but not committed. For example, for the role check, as per the code below, the program would check that the specified role was one which the database recognised and, if they were a student, set their total to 0 as they would be using totals. If not, an error message would be returned specifying what changes need to be made.

```
        # check that they've put in a valid role
        if role in USER_ROLE.keys():
            new_user.role_id = USER_ROLE[role]
            if new_user.role_id == USER_ROLE["student"]:
                new_user.total = 0
        else:
            errors.append(
                f"{role} is not a valid role for users - it must be 'student', 'staff' or
'admin'.")
```

```
        if errors:
            return errors
        else:
            db.session.add(new_user)
            return new_user
```

The program would go through this process for each row. At the end, if any of the rows had errors, the program would rollback the committed users, causing them to not be added to the database. This would allow the user to go back and fix the users with problems, without also having to remove the users from their .csv file who had been successfully added. Because the errors for each row were stored, I was then also able to code a nice print out for the user that ran through what they needed to change.


If these requirements are not met, the system will provide a breakdown of the errors in th

Choose File | No file chosen | Upload

Errors in row 2:

- The User ID "1600" is invalid or already in the database.
- Please ensure that both first and last names are provided for this user.

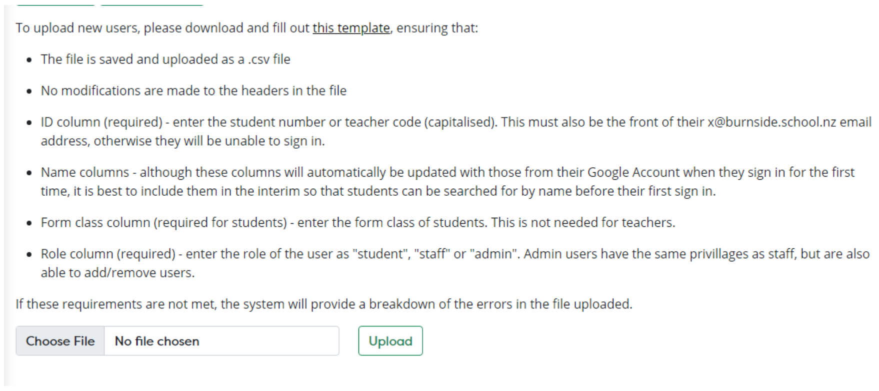
Errors in row 3:

- The User ID "ABC" is invalid or already in the database.
- stafefweff is not a valid role for users - it must be 'student', 'staff' or 'admin'.

Errors in row 4:

- The User ID "" is invalid or already in the database.
- Please ensure that both first and last names are provided for this user.

Extensive documentation was also added, helping staff to figure out what they need to put into the spreadsheet to upload it successfully.


To upload new users, please download and fill out this template, ensuring that:

- The file is saved and uploaded as a .csv file
- No modifications are made to the headers in the file
- ID column (required) - enter the student number or teacher code (capitalised). This must also be the front of their x@burnside.school.nz email address, otherwise they will be unable to sign in.
- Name columns - although these columns will automatically be updated with those from their Google Account when they sign in for the first time, it is best to include them in the interim so that students can be searched for by name before their first sign in.
- Form class column (required for students) - enter the form class of students. This is not needed for teachers.
- Role column (required) - enter the role of the user as "student", "staff" or "admin". Admin users have the same privillages as staff, but are also able to add/remove users.

If these requirements are not met, the system will provide a breakdown of the errors in the file uploaded.

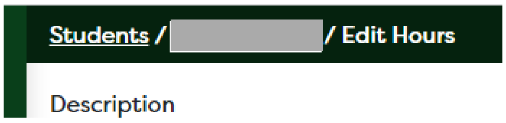
Choose File | No file chosen | Upload

I made sure to add code to delete each .csv file after it had been used, so that my web server would not get clogged up.

Finally, I added another page to remove users. This reused code from previous routes, as it would just iterate through each of the user's groups and remove them and their hours, remove any remaining hours that weren't entered under a group and then remove them from the database all together.

## Final Stakeholder Testing

I had now wrapped up my development, so I went back to each of my stakeholders and ran through the app with them. Since I had just finished the staff portal, I started by going through the app with Mr R[redacted] He was very impressed with the app, particularly with the upload feature and really appreciated the indepth error messaging it gave. We went through both the staff and student portals, and one thing that did come up, particularly with the staff portal, was that staff would often travel down quite a series of links and end up on a page far away from one of the ones listed in the menu bar. This made it difficult to track where exactly in the website a user had navigated from, so he suggested adding breadcrumbs, which are used in many other websites, to indicate where in the website the user was. These fitted really nicely into the menu bar along the top, which then provided a list of pages and links to get back to them, helping users to return to where they had been previously and track their journey across the site.



Mr R[redacted] didn't have any other suggestions, so I next went to see Mr D[redacted] to gather his thoughts. He was particularly happy with the modular code structure, which he said would make it easy for him and another digital technologies to scale and deploy the app next year. I had engaged with Mr D[redacted] quite extensively throughout the development process, so he didn't have any new feedback for me, from either a technical or staff perspective; he thought that the system would do a good job at enabling all kinds of service to be recorded as it put the power in the hands of individual teachers and students, particularly by allowing teachers to encourage students. I also took the finished app back to Miss H[redacted] and Mr G[redacted] who I had talked with earlier, particularly in terms of the verification needs for the new system. They were both relieved to see the responsibilities of staff and students in recording hours cleared up, with Mr Gibson noting that the page which displayed the group members and their total hours would make it really easy for him to verify this against the role he took each week at Sustainability Council. I'd put quite a lot of effort into understanding their needs before developing, so they didn't have any feedback for new features to be added and were instead just excited to use the new system.

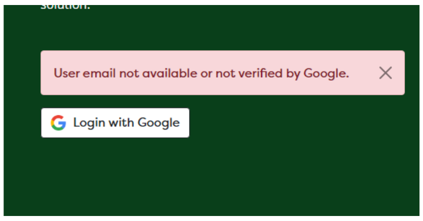
Finally, I ran through the project with some students, the stakeholder I had identified as being the most important of all. I started by taking my website back to the four students I had engaged with regularly throughout the development process, who all liked the finished product and found it easy to use. They hadn't seen the groups functionality before as I'd developed it around the period leading up to mock exams, and their only suggestions were to make a few pages look a bit cleaner, particularly the detailed group log page, which involved changing a few HTML classes to make some text smaller and better spaces. Other than that, they were all able to use the website and said that it was easy to learn and use.

The real challenge, however, came when I took my website to some of the students I'd interviewed back at the beginning of my project, who had not seen the website before. I sat down with a few of them, particularly those who had indicated they were heading directly into employment next year, to see if they found the site easy to use and if they thought that it would make them more likely to record their hours. I gave them some sample service hours to record and all of the students were able to follow the instructions on the add hours page and successfully log hours, join groups and the like without any help from myself, which showed that the instructions I'd developed with my stakeholder students were effective. This was particularly rewarding to see, as the whole point of the project was to make it easier and less confusing for students to record hours. All the students indicated to me that they'd have been more likely to record their hours if we'd been using this system and liked that they could track their progress easily. A couple of the students commented on the widget in the dashboard showing how many hours they needed to get the next award, noting that they thought that it would make them log their hours on time as they'd want to earn the next award. This again helped to achieve the goal of the website I'd set out, as if students were more likely to record their hours during the year, then they wouldn't have to retrospectively piece together a log at the end of the year, which would be a lot more work and put off some students.

## Extensive Site Testing

Now that I'd checked with each of my stakeholders that my website would meet their needs, I embarked on a final set of site-wide tests to search for any bugs or issues remaining. This involved testing every button and link, checking that every form handled errors correctly and that each route correctly handled being passed incorrect variables. *The full testing tables for this process can be found in the appendix of this report.*

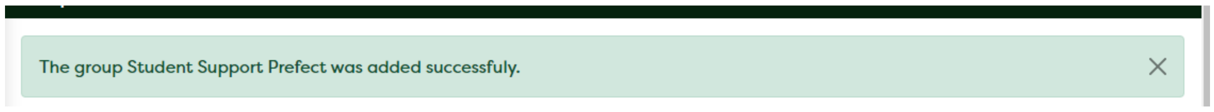
Through testing, I found a few errors, the first of which being that on the login/callback route, altering the url to not pass any codes would lead to an error, so I added error checking that redirects users to the login page with an error.


User email not available or not verified by Google.

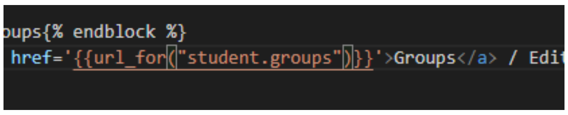
I also found that students could edit their hours which had been locked by a staff member by altering the edit-hours/id url. I patched this bug to prevent this from happening in the future by checking the status id of the item.

```
if item not in current_user.hours or item.status_id == 2:
```

Another issue discovered was that there was no confirmation message presented when a staff member made a new group. To address this, a flash was added to the staff.groups route.


The group Student Support Prefect was added successfuly.

A further issue was found on the staff.edit_hours route, where the group breadcrumb directed the user to the dashboard rather than to the groups page. This was because the breadcrumb was directing the user to the student groups page, which they were unable to access and thus was redirecting them to their dashboard. This was changed to the staff.groups route to fix this.



```
ups{% endblock %}
 href='{{url_for("student.groups")}}'>Groups</a> / Edi
```

Having completed extensive site testing and checked that my website met the requirements of stakeholders, I had finished the development process and was now ready to pass my project on to the school to be implemented after I graduated later that month.

## Conclusion and Evaluation

Frustrated by a cumbersome recording process that was not only irritating but disregarded some kinds of service which students were doing, I'd set out to create a new service hours tracking system that addressed these concerns. This has been achieved by removing barriers to recording service, notably by making the process to record hours easier by adding an easy-to-use form. Furthermore, the new website allows students to track their progress, which had previously been difficult for them to do, by showing them their totals, progress towards goals and breakdowns of their service, including by the group they've completed it in. With students being able to track their progress now, some will hopefully be inclined to do more service now that they know how far away they are from an award or want to achieve a goal. Finally, I'd ensured that the system valued all kinds of service equally and would allow students to understand this. This was achieved by having custom groups lists for each user, chosen by them, rather than the one list of all of the main ways that students earned service in the current list. As students could choose their own list, with staff now being able to add groups they were involved with to the options for students to choose from, students would be more likely to see groups they were actually involved with show up on their form to log hours. More so, the new website would fully support instances where students completed work outside of groups by allowing them to add these by selecting a teacher without having to select an "other" option or the like that suggested their work was inferior; they could simply select a new group. Teachers were also able to log hours for students to get them started with logging and encourage them to record their hours if they did the same activity in the future; it would signal to them that their work counted as service. Staff would also be able to monitor the progress of students and encourage them to do more service or log their hours. With all this put together, all students would be more likely to record service, have all of their service recorded and thus achieve an award. This would hold true for all students, regardless of what their after school intentions were, thus helping them to get service awards. These service awards will then lead to better outcomes for students, regardless of their after school intentions. As such, my project had been successful in improving outcomes for the students I had set out to help.

Throughout the development process, I had also discovered other requirements for my project. With my new system that added clarity to the service hour recording process and allowed staff to track, verify and edit logged hours, teachers like Miss H[redacted] and Mr G[redacted] will have an easier time managing the service hours of their system and more clarity about their responsibilities. Mr R[redacted] now also has an easier system for him to use that will allow him to better manage the progress of students and export their awards.

Finally, from a technical perspective, I'd paid careful attention to best practices throughout my back-end and code that allowed my project to be successfully handed over to staff at school to implement. I'd split my code into multiple files (for both my templates and python code) that would make it easy to locate the right sections of code if changes needed to be made and used an object-orientated approach that improved code reusability, which meant that it issues with my code could be changed in one place, and updated in all places they were used. Function docstrings and commenting had also been used throughout my code to explain every method, function and tricky piece of code. Extensive technical testing ensured that the site I was handing over was as bug free as possible.

Overall, this project was both an eye-opening and enjoyable experience; I really enjoyed developing a piece of software that would actually be used and that had clear users in mind. It seems well placed to be an asset to my school after I leave and to help future students reap the rewards that a profile of service offers.

*Following this page is the testing appendix. To see videos which run through the finished site, code as well as the GitHub repository and iteration documentation, please refer to the links at the beginning of the document.*

**Outstanding Scholarship exemplar 2021**

| **Subject:** | **Technology** | **Standard:** | **93601** | **Score:** | **19** |
|---|---|---|---|---|---|

| **Q** | **Score** | **Annotation** |
|---|---|---|
| Synthesis and integration | 7 | Requirements and attributes for the practice were informed by significant research. This established (and allowed the candidate to refine) a deep understanding of the issue, leading the candidate to find an authentic path forward for a project – the need to find a way to realistically streamline and manage the school's service award system. The candidate's understanding of the issue in the broadest possible sense was indicated by their discussions of how important service awards can be for candidates' future opportunities, and how the current manual system hinders candidates' engagement with awards.<br>Finding and managing stakeholder relationships was a critical part of the candidate's practice. This relationship management meant that appropriate knowledge was extracted (and presented in the report) in an efficient and concise manner. Consequently, the candidate was always moving their project forward in a controlled direction.<br>The candidate's commentary throughout the submission clearly communicates the high level of technological processes undertaken. The report flows easily, demonstrating powerful design thinking.<br>The candidate developed a comprehensive solution consisting of a database, backend, and UI. This demonstrated the candidate's high level of skill in drawing from previously learnt and understood techniques and processes, and understanding their own knowledge gaps. The candidate learnt new knowledge and gained understanding through targeted research. This enabled the candidate to find optimisations and to polish their practice. |
| Justification | 6 | The high-quality written report included strategic use of images to exemplify discussions in their text. This was aided by evidence that was always relevant, with limited repetition of content. In their larger body of work, the candidate avoided using irrelevant content associated with other Achievement Standards that fall outside the Scholarship criteria.<br>The candidate fully understood the limitations of the existing manual system, and this allowed them to develop a solution and comprehensively justify the decisions they made. This, in turn, enabled the candidate to justify their technological practice and decision-making, and the effectiveness of the outcome. The candidate has provided a clear and justified narrative that further enhances the perception of the outcome. |
| Critical reflection | 6 | In this report it is evident that the candidate has critically reflected on their own technological practice. The candidate regularly considered stakeholders' expert understanding of the manual system and used this to critically reflect on decisions made during the practice. The candidate demonstrated that they were able to independently integrate knowledge gained from their technological experiences. This is evidenced by their reflective commentary throughout.<br>Extrapolation is demonstrated through the candidate's ability to take known understandings about privacy and security and apply them when accessing a third-party database containing confidential student information. This had to be well articulated to convince the stakeholders of the viability and credibility of the outcome. |