Imperial College London

Imperial College London

Department of Electrical Engineering

Final Year Project Report 2023

Building An Accessible Project Selection System And Improving Project Allocation

Student:
Dominic Justice Konec

CID:
01945883

Course:
EIE3

Supervisor:
Dr Thomas Clarke

Second Marker:
Dr Edward Stott

June 2023

Final Report Plagiarism Statement

I affirm that I have submitted, or will submit, an electronic copy of my final year project report to the provided EEE link.

I affirm that I have submitted, or will submit, an identical electronic copy of my final year project to the provided Blackboard module for Plagiarism checking.

I affirm that I have provided explicit references for all the material in my Final Report that is not authored by me, but is represented as my own work.

I have used ChatGPT v4 as an aid in the preparation of my report. I have used it to improve the quality of my English throughout; however, all technical content and references come from my original text.

Acknowledgements

Firstly a thanks to Dr Thomas Clarke who has exhibited remarkable dedication and responsiveness, providing invaluable assistance throughout the project. His quick response time and significant commitment of time have been instrumental in ensuring prompt help whenever needed and ensuring the project stayed on track.

Thank you also to Amine Halimi who's knowledge on Power Apps proved vital in applying the software.

Finally thank you to my friends and family for making my life outside of work so enjoyable.

Abstract

This report presents the development and evaluation of an accessible and efficient project selection and allocation system for the final year project (FYP) process in the Electrical and Electronic engineering department. It addresses challenges faced by students and staff in project selection and allocation.

The report includes background research on allocation algorithms and existing infrastructure. It highlights the requirement capture phase, identifying the needs of students and staff.

The system implementation utilises PowerApps as a frontend and Microsoft lists for backend storage. It focuses on guiding participants towards optimal allocation while preventing manipulation. Measures are taken to limit gameability through dishonest preferences.

The algorithm choice for project allocation is explored, with experiments to gauge the viability of modeling the problem as a linear sum assignment problem (LSAP) which an be solved to give an optimal solution improving on the previous heuristic approach. The Hungarian algorithm is chosen to solve the problem as a LSAP with tests being carried out to ensure an effective execution time. Preprocessing code is designed and created to interface with the algorithm, prioritising ease of use and efficiency.

The created system is critiqued and further work discused.

Contents

Chapter 1

Introduction

In the final year of their studies, students at Imperial College London are tasked with completing a final year project (FYP). Projects are either self-proposed or allocated from a list of supervisor-proposed projects. The allocation is based on student preferences and suitability rankings assigned by supervisors. This is the case in the Electrical Engineering (EE) department, which will be the case study for this project. Each year in the EE department around 200 students select from approximately 230 projects or self-propose their own. Each projects is supervised by one out of about 50 supervisors, that supervise between 1 to 8 FYPs. It is important to match students to projects that are both well suited to them, and that they will enjoy.

This project will have three key focuses: what is the necessary information that must be supplied by supervisors and students in order to achieve high quality student preferences, how can this information be collected and displayed to the involved parties, and how can the project allocation based off this data be optimised. These focuses will be analysed with the goal of building a project discovery and allocation system. The aim of the system is to generate allocations that satisfy the majority of students and supervisors, without requiring an excessive amount of their time.

The current system for creating projects, self-proposing projects, searching for projects, submitting preferences, and obtaining approval from supervisors is rather disjointed. This can cause unnecessary confusion and delays for both students and supervisors alike. This is particularly problematic as both parties are often busy with academic commitments and other responsibilities. Furthermore, the current allocation algorithm used does not guarantee an optimal solution and must run for at least half an hour or so to get viable allocations. With all this in mind, there is a need to streamline the process and create a centralised system, with a more efficient allocation algorithm, that will benefit all parties involved.

The motivation for this project is that improved project allocations lead to more students with FYPs that are very well suited to them, both skills wise and interest wise. A student being well suited to their project enables them to perform their best and get the most out of the FYP. An argument can also be made for the improved accessibility making it fairer for students with accessibility issues.

Chapter 2

Background

2.1 Introduction

The background chapter serves as a comprehensive overview of the current system and its inherent limitations. It delves into the challenges faced by students in accessing project information, submitting preferences, and the existing allocation algorithm. Additionally, it addresses the concerns and obstacles encountered by the FYP Coordinator and supervisors. In order to explore potential solutions, two frontend architecture approaches, namely PowerApps and traditional JavaScript, are examined, highlighting their respective advantages and trade-offs. Moreover, the choices for the allocation algorithm and its integration discussed, with a specific focus on LSAPs and the suitability of F# and Python for data processing and optimisation. By providing this contextual information, the background chapter lays the groundwork for understanding the issues and challenges addressed in this report.

2.2 The Current System

2.2.1 Project Discovery

Currently, the years projects are accessible through different links on a web page hosted in the EE intranet. The links take users to different lists of projects that are available to different streams of students. The lists are plain text and consist of one project followed by the next with a line separating them. Each project has sections, ID, Supervisor (which comes with their room and a link to email them), Title, and Description. There are no options to filter the list and all project information gets displayed at once, which makes it hard for users to keep track of where they are and can be overwhelming, especially for students with accessibility issues[1].

There is nothing identifying how long a project has been released for, meaning the whole list has to be scanned each time to check for new projects. Alternatively, there is a PDF released with projects submitted after the first week of term, where new projects can be found. Unfortunately, this PDF faces the same issues as the project list and isn't filtered by degree stream, so holds a lot of unnecessary information.

Taking into consideration all of the above, it's conceivable that some students miss projects that could be suited to them, or at the very least spend longer than necessary discovering projects.

2.2.2 Preference Submission

At the moment, the system allows students to add up to ten preferences. For the student to be considered for allocation, it requires at least three preferences for bachelor students, five for masters students, and a project from at least three different supervisors. This improves the quality of the allocation as students are made to put a wider range of projects, this leads to a lower chance of students having the same preferences which causes an increased number of students not being allocated a project.

Preferences can be made equal, for example one can have two 1st preferences and a 3rd preference, instead of a 1st, 2nd, and 3rd preference. This flexibility is nice as it allows for students who aren't as particular to get a higher chance of allocation.

Students are encouraged to check their preferences and sign-off page, '**every day**' to see the status of the projects they have added as preferences, in case they are deemed not suitable, or to check for new projects that they may be suited to. This is quite inconvenient and excessive, a better solution could be automated emails when students are ranked or when a new project for their stream is released.

Automated emails could also aid with another issue faced by the FYP Coordinator. They are in charge of making sure that students fill out their preferences by certain deadlines, and encourage a speedy submission of initial preferences. This helps the FYP coordinator, through trial allocations, to get an idea of students preferences that can lead to a low chance of allocation. The current method manually looks through student preferences to identify students that need their preferences altered. This is an arduous task that may be better performed by having automated emails dispatched on configurable dates to students not meeting configurable requirements with their preferences.

When the preferences are finalised, only the preferences up to the required number are taken into account, as to not bias the allocation algorithm towards students who don't put down as many preferences. This seems sub optimal, as although it does make it fair on students that are not suitable for many projects, a great deal of preference information goes unconsidered. Instead a potential improvement could be allowing the FYP Coordinator to easily configure how many preferences are taken into account when running trial allocations. With quick algorithmic integration the Coordinator could see if an alternate number of preferences being taken into account leads to a more optimal allocation.

2.2.3 The Allocation Algorithm

The existing allocation algorithm as described in the final report of Ramon Jutaviriya was reviewed as they implemented it as part of their own FYP[2]. This consists a search heuristic algorithm, more specifically an original hill climbing algorithm[3] [4], due to a lack of flexibility in traditional allocation algorithms. This notion will be investigate in this report.

Due to the nature of heuristic algorithms, the resulting allocation has no guarantee of being optimal[5]. In addition, the algorithm can be run indefinitely in hopes of finding a more optimal solution. This creates a number of issues for the FYP Coordinator. Talking with the coordinator, they explained the algorithm would usually take around half an hour to get a plausible allocation set. They continued that improvements were often still being seen two to three hours into execution. This shows that at thirty minutes results are not optimal. This is a long time especially when compared to numbers seen when testing allocation algorithms in section 4.6.

With no way to tell if the solution is optimal, the FYP Coordination has to make a decision on how long to let the algorithm run for and when to start it. What's more, hyperparameter[6] tuning, such as the limit on student preferences considered, becomes far more difficult. The Coordinator must decide when a certain set of hyperparameters has reached a good enough solution to move onto the next set of hyperparameters. Moreover, they have to also consider that letting one set of hyperparameter run for the time it would take to test 5 sets, may result in a more optimal set of allocations.

2.2.4 Students Not Allocated

Students who aren't allocated a project in the first round of allocations are handled on a case by case basis. They are contacted by the FYP Coordinator and alternative projects, whether in the remaining project list, self proposed, or a hybrid, are discussed.

2.2.5 Supervisors Experience

In the current system, when a student submits a project as their preference, they must be signed-off by the project supervisor. Students can either be signed off as 'Definitely Suitable', 'Maybe

Suitable' or 'Not Suitable'. Supervisors have reportedly had many issues with the current signing-off process, due to the software being defective. This causes delays to the signing-off process, making students unsure whether they need to be finding additional preferences. Furthermore, if the system is unreliable, students could be incorrectly signed-off to projects they are not suited for, and vise versa.

Students are requested to talk to a supervisor before listing their project as a preference. However, project proposals have no section for how students should get in contact. Some projects have a meeting timetable, but those with limited or no availability, can cause supervisors to become inundated with emails if the project proves to be popular. This provokes confusion and takes away time that supervisors could be spending signing-off students in a more timely manor. Potentially, an 'If Interested' section could be added to proposals detailing the next steps that should be taken when debating on including the project in preferences.

2.2.6 EIE Students Taking DoC Project

The Department of Computing (DoC) have a similar yet separate system to the EE department. Students taking electrical and information engineering (EIE) can elect to take a FYP from DoC instead of EE. Currently the FYP Coordinator must manually find out which students want to take DoC projects and remove them from the EE allocation. Time and difficulty could be saved for the FYP Coordinator by having it built into the system, so students can identify themselves as wanting to go into the DoC allocation system and be automatically removed from the EE one.

2.3 Frontend Architecture

2.3.1 PowerApps

One option for the frontend is a group of Microsoft PowerApps[7], utilising SharePoint Lists[8] to store student data. There are various factors as to why PowerApps could be the best choice. Firstly, the department has recommended Power Apps due to the availability of support and advice from individuals with prior experience in creating and managing Power Apps, as well as the university already having a license.

Industry advice[9] also highly recommended Power Apps for its fast development and ease of use. This is ideal for the project, as speed is crucial when considering the large amount of functionality needed. While an article[10] comparing React[11] and Power Apps rated React as superior, the pricing negatives for Power Apps were not relevant to this project, and the low-code aspect of Power Apps held more weight.

The low-code aspect of Power Apps greatly contributes to the speed of development, as there is no need to spend time learning frontend JavaScript[12]. This saves time, as learning to develop in JavaScript would take longer than with the low learning curve of Power Apps development. Additionally, the security aspect of Power Apps was considered in the decision-making process[13]. Using a JavaScript library like React leaves the possibility of human error, leaving the system open to attacks[14] like SQL Injection[15], or authentication issues that could be exploited by students seeking information on others' preferences to angle their own.

Moreover, hosting the system on Power Apps allows for easy integration[16] with .NET[17] (discussed later), and Power Automate[18], which has the added benefit of features like automated emails[19]. This would be beneficial in fulfilling the requirement that students and supervisors need to act quickly, with the system being able to identify users who are not meeting the speed requirements and email them with reminders. These notifications would be especially helpful for students with accessibility issues, as discussed[20] by the Web Accessibility Initiative, stressing the helpfulness of these reminders.

2.3.2 Traditional JavaScript Approach

When it comes to developing a system that requires a high degree of customisation and control, traditional frontend coding in JavaScript can be an excellent choice[21]. While Microsoft Power

Apps may have certain advantages, like a low-code development process, using JavaScript allows for a much greater level of flexibility and customisation[22].

Having more customisation and flexibility, could allow for more elements that make the system more accessible. It would also allow for more freedom in implementation, adding elements that best fit the task at hand rather than being conformed to PowerApps component library.

JavaScript is not the most complicated language, however it does take longer than Power Apps to grasp. It is also not inherently designed to be accessible, as although JavaScript libraries like React[11] do have functionality to enable creating functional apps[23], Power Apps can implement screen reader text and tool-tips very easily and efficiently.

2.3.3 Comparison Table

Approach	Cost	Learnability	Amount of Code	Flexibility	Easy Accessibility
PowerApps	Existing licence	Easy[24]	Low-code	Low	Built-in tool-tip and screen reader features
JavaScript	Free	Basics 8-16 weeks[25]	High-code	High	More involved to make accessible

Table 2.1: Frontend approach comparison.

2.4 Algorithm Integration

The data holding the student and staff preferences needs to be processed into a form that can interface with the allocation algorithm.

2.4.1 F#

A language that could be chosen for transforming and manipulating the data is F#[26]. Discussed in this article[27] are factors that make F# such a data processing powerhouse. It is a non-mutable functional ML-derived language. The functional nature of F# makes it well suited to manipulation data and has many built-in functions for data transformation. The ability to pipeline the data through functions allows a clear and easy way to read code, the non-mutable data helps reduce exceptions and lower the likelihood of bugs. F#'s pattern matching[28] makes its easy to extract data in different cases. An example on the Microsoft tech community website[29] displays how F# type providers can easily extract data from HTML and this web page[30] informs on F#'s parsing power and ability to generate database dummies which prove helpful in testing. Furthermore, F# has the ability to run external functions stored in Dynamic Link Libraries[31]. This is helpful as many allocation algorithm implementations are written in C++[32] [33] which can be turned into a dll[34]. F# is also part of the .NET framework meaning it has access .NET libraries[35] making for simple API connections with SharePoint[36], this would make it easier, in this project or future work, to access the SharePoint lists with student and supervisor preferences if it is decided to have the frontend be Power Apps. Unfortunately, F# is not a very popular programming language[37] so the online support resources are less deep than other more popular languages. However, the electrical engineering department does have people with deep knowledge of F# such as this projects supervisor professor Clarke[38]. This means help should still be accessible even if not as immediately.

2.5 Python

Python[39] is a very popular programming language[37] so there are a wide range of resources with PyPI[40] having over 450,000 projects at the time of writing. Python also has a deep online knowledge base owing to its popularity, meaning help is accessible quickly and implementations of allocation algorithms are readily available[41]. One of these libraries, Pandas[42], provides easy-to-use data structures and functions for data manipulation such as filtering, grouping, and merging data[43]. Python is not part of .NET so if Power Apps are used for the frontend of the system then obtaining the data stored on the Microsoft Lists will not be as simple. Python does however, have

excellent support for data visualisation with libraries like Matplotlib[44] and Seaborn[45] which could enable further study of student preferences and ranking down the line.

2.6 Optimisation Algorithms

The problem of finding the best possible set of pairing between students and project is an optimisation problem[46]. The class of optimisation problem that best describes the FYP allocation is matching problems[47]. Matching problems arrive when two groups need to be connected with each member of each group only being connected to one member of the other group. There exist both one-sided[48] and two-sided[49] matching problems. In one-sided problems only one group has preferences/constraints whereas, two-sided problems see both parties having preferences/constraints.

In the case of the FYP allocation one group, students, have preferences on the other group, projects. There are also further constraints though, for example, students have suitability levels for each project and supervisor loading is also an issue. These could be handled by modeling projects as having preferences about students. This would make the problem a two-sided problem. But, further constraints like these could be worked into the preferences on the student group or handled by dummy participants as investigated in section 4.5. This would keep the problem one-sided opening the possibility for faster, simpler algorithms.

2.6.1 One-sided Matching problems

If the FYP allocation problem can be modeled as an an one-sided matching problem then algorithms discussed in this section will be applicable.

Linear Sum Assignment Problem

The simplest form of one-sided matching problems is the Linear Sum Assignment Problem (LSAP)[50]. The transformation of the student allocation problem to the LSAP is simple, students are agents and projects are tasks. The cost of assigning agents to tasks is generated through the student and staff rankings. Algorithms that solve LSAPs find the optimal solution of pairing where the total cost of matching is lowest. The simple nature of LSPAs mean it can be hard to capture all the requirements of problems such as the FYP allocation problem. However, if the FYP allocation problem can be represented by an LSAP then the use of LSAP optimisation algorithms such as the Hungarian Algorithm[51] made.

The Hungarian Algorithm has two different main categories of implementation, ones that view the problem from the context of a cost matrix[52] and ones that view the problem from the context of a bipartite graph[53]. Both have time complexity implementations of $O(N^4)$ and $O(N^3)$ with N being the number of agents. More specifically $O(N^3)$ matrix implementations are actually $O(N^2M)$, M being the number of projects. $O(N^3)$ bipartite graph implementations are actually $O(NE)$ where E is the the number of edges in the bipartite graph.

The simplicity of the Hungarian algorithm means on small groups implementations can execute very quickly before the time complexity start to drastically increase execution time.

In Jutaviriya's report2.2.3 they argue that there are two main issues with representing the FYP allocation problem as a LSAP. Firstly, its stated that because the transformation does not include a model for supervisors that it is unable to "address the load-balancing of lecturers", later in the report assignment algorithms are described as simply not flexible enough to be altered to the context of the FYP Allocation problem. Secondly, its stated that the time complexity of the Hungarian algorithm, the obvious choice for solving LSAPs, "can be impractical" in the case of the FYP Allocation problem. These points should be examined to understand whether or not the problem can in fact be modeled as a LSAP.

Linear Bottleneck Assignment Problem

An extension of the LSAP, the Linear Bottleneck Assignment Problem[54] (LBAP) focuses on minimising the maximum cost of allocating an agent to a task. This theoretically would fit the

FYP allocation problem, by reducing the dissatisfaction of the least satisfactory allocations. For example, whilst the LSAP would allocate three students to their first, second, and eighth preferences (totalling eleven, max cost eight), the LBAP instead would allocate all students their fourth preferences (totaling twelve, max cost four). However, this is not the most ideal, as the algorithm doesn't take into account total cost. Furthermore, students tend to submit around five preferences out of approximately 230 projects, so an algorithm that solves LBAPs would have to be able to handle not being supplied with the costs of allocating agents to most tasks. This is because a blanket expensive de-allocation cost is unusable within LBAP.

2.6.2 Two-sided Matching problems

If the FYP allocation problem cannot be modeled as a one-sided matching problem or, implementations of algorithms mentioned in the previous section don't execute quick enough, then a two-sided algorithm may be used.

Two-sided Student Project Allocation Problem

The two-sided student allocation problem[55] (SPA) is a very close fit to the FYP allocation problem. With it too focusing on how to match students to supervisor projects. It takes into account supervisor loading and makes the allocations based on a student preference list. Rather than having student suitability classes like in the current FYP system, the problem has supervisors submit their own preference list on students. These preferences are treated as the project group having preferences, this makes the problem a two-sided matching problem.

In the FYP allocation problem student by student suitability rankings are unfeasible for supervisors to submit. This is due to some projects having very high interest and so rankings that in-depth are too time consuming. This difference could be subverted however, using a variant of the SPA that allows ties[56]. It is conceivable that additional suitability ranks could be added to the system and these used in place of the student by student supervisor preferences. However, these ranks would hold a large number of students in the FYP system and so the SPA may not be a suitable model.

It should also be mentioned that solutions to the SPA aim for stability rather than optimally, reasoning being the problem is NP-hard[57] so algorithms producing optimal solutions with polynomial-time are not worth looking into[55].

2.7 Conclusion

The background chapter has shed light on the limitations of the current system and has established the necessity for an improved solution. It has underscored the difficulties faced by students in navigating project lists and submitting preferences, as well as the shortcomings of the existing allocation algorithm. Through a comparison of two frontend architecture approaches, the chapter has emphasised the benefits of PowerApps' low-code development and its seamless integration capabilities. Furthermore, it has explored the potential utilisation of F# and Python for efficient data processing and optimisation algorithms specifically finding the Hungarian algorithm to be a good potential fit. By presenting this comprehensive background information, the report has set the foundation for the subsequent analysis and proposed solutions. This understanding of the system's shortcomings and the context in which they arise will inform the development of an enhanced allocation system.

Chapter 3

Requirement Capture

3.1 Introduction

In this chapter, the process of requirement capture for the project is presented. It begins with a discussion held with the FYP Coordinator, Doctor Thomas Clarke, who provided valuable insights into the issues with the current system and the behaviors of students and supervisors. The problems identified range from students submitting a low number of preferences to supervisors abusing the sign-off system. Additionally, the tendency of students to try to game the system was recognized as a concern. To gather further information, surveys were conducted among students to understand their preferences and behaviors regarding the system. The survey results highlighted the importance of including desired and required skills in project proposals and revealed that many students tend to focus on projects within a specific area of study. Furthermore, opinions were collected regarding the level of competitiveness information that students would like to have access to. Finally, the chapter concludes with a discussion on personal experiences, which led to a shift in focus towards improving the communication of information and frontend usability of the system.

3.2 Meeting with the FYP Coordinator

Doctor Thomas Clarke is the supervisor for this project and also in charge of coordinating the project allocations in the Electrical Engineering department at Imperial College. Many calls took place with professor Clarke to discuss issues with the current system and behaviour of the students and supervisors. The many problems spoken about would be resolved by a clearer centralised system. However, there were some more nuanced problems with solution that would require more thinking.

3.3 Surveys

Surveys were conducted on students to get both an idea of how they behave and what preferences they have regarding the system.

3.3.1 Project Proposals

Students were asked to say what they found most important in project proposals to get an idea of what aspects supervisors should include and exclude as to not overwhelm students with information.

The feedback shows how important many students find having the desired and required skills included in the project specification, something that was often omitted in the previous year. Both options were favored by approximately 90% of respondents. The option with the highest pick rate (94%) was the project's main area of focus, for example, control systems. This makes sense as it is the main information students need to quickly identify if a project is in an area of interest and if they have the necessary skills to do it.

Problem	Details
Student submit a low number of preferences.	Students providing a low number of preferences makes it very hard for the assignment algorithm to make a satisfactory allocation. This is a tricky problem as it is unclear in each case, whether the low number is due to the student actually not having many projects that they would be suitable for or them just not taking the time to go through the entire project list in detail and find projects that they may not initially go for but are suited to. Another take away is that care must be taken not to make it too easy for students to only look at projects in their comfort zone and ignore reading through projects that may suit them well, even if said projects don't appear that way at first glance.
Students and supervisors wait until the deadline is very close to submit preferences and sign off on students.	This makes it very difficult to run trial allocations and identify students who are at risk of being unallocated with there initial preferences.
Supervisors need say over who can do their projects. However, some of them abuse the sign off system to try to get specific students.	Not only is this quite dubious but also these desired students tend to be in high demand so supervisors can end up with their project going unallocated and other students, who were in fact suitable, getting one of their lower preferences or being unallocated entirely.
Students are smart and will try to game the system	It should be considered at all times in the project that what is being implemented or shown to the students doesn't allow them to enter a version of preferences that does not represent how they feel, but manipulates the system into allocating their desired project.

Table 3.1: Information on main issues discussed in meeting with FYP Coordinator

Students were given the option to put forward their own ideas of what's important. There was mention of including where on the research vs implementation spectrum the project falls, and also the flexibility of the supervisor to make alterations to the project focus/deliverable, which could give students the confidence to ask about slightly altered projects that they may have before decided aren't for them.

3.3.2 Information on Competitiveness

Students were also asked to state what information they would like to see about other students' preferences. The most favored idea was to know for each of a student's preferences how many other students had that project ranked in their top 2. This would help students get an idea of how competitive a project is. In previous years many students only put down highly competitive projects projects leading to a high change of not being allocation. Giving incite on the competitiveness of a project will encourage students to put down more options as to not risk going unallocated.

One student did voice concern over providing this information to students. The student believed it the additional information would be used to create "complex ranking strategies that defeat the purpose of the project assignement algorithm". This is a genuine concern and the reason for not putting forward systems where students are given a likely hood of their allocation to a specific project. The fear being that students would brute force their preferences, with the goal of ensuring certain allocations. The student suggested that instead the total number of students with the project as a preference should be shown. With their reasoning being "this is the minimum info that allows students to know if a project is high-demand". Although I believe the students alternative does reduce the ability to game, it should be considers that the larger numbers that come with

this alternative may scare students away from choosing well suited projects. It could be the case that the project is in a high number of students preferences, but 90% of those students have the project in options 4-9. This would mean a student looking may think they have little chance of getting the project, when in fact putting it as their first preference would give them a very high likely hood of getting allocated the project. It should be noted that some strategising from the students is positive, if it leads to the consideration of a wider range of projects or students including more preferences because this leads to less students unallocated. If a students top five preferences are all extremely competitive, then that student switching one of the preferences out for a less competitive project that was originally lower in their preferences, could increase their chance of at least getting a project they have some interest in. This could be viewed on the line of gaming as they students isn't being completely honest about their preferences. Nevertheless, this isn't inherently unethical or something that should be discouraged. It is the students choice if they want to lower their chance of being allocated one of the projects that most enticed them, in favor of maximising their chance of a first round allocation. Especially if that student has not been ranked with good suitably for that project by the supervisor.

3.3.3 Preference Diversity

A majority of surveyed students said their top 3 preferences where all in the same area of study, this could suggest that students tend to only look at projects in a specific area they are interested in and don't consider other suitable options. Whilst talking with professor Clarke he mentioned how in recent years projects in the area of machine learning have been extremely popular with there being far more demand than supply. An indicator of project competitiveness and potentially suggesting similar projects could potential help vary the areas in student preferences.

3.3.4 Suitability Ranking

Over 60% of students surveyed would be happy with supervisors being able to rank students on there suitability to do a particular project. The reluctance of others is no doubt down to concerns of fairness issues and supervisor biases, the theory is enforced by a comment from a student voicing the importance of supervisors correctly using the current sign of system and not abusing it to get a specific student allocated. A ranking system would hopefully encourage supervisors to look deeper into students viability. If the system was made easily accessible to the FYP Coordinator with email alerts sent when dubious rankings were detected then this issue could be widely mitigated.

3.4 Personal Experience

I myself underwent the FYP selection process and it was part of the reason i decide to alter the original direction of this project to have a much larger focus on the communication of information between participants and the frontend ease of use plus accessibility. Typically, students doing FYP on the allocation problem have mainly concentrated on the allocation algorithm used and how to optimise it. I believe this altered focus arises from my own experience using the system with dyslexia, and anecdotal evidence from my fellow students going through the process. I found the system quite difficult to use owing to the plain text single colour nature of the web pages. I often missed out on information, or was overwhelmed by it in the case of the large lists of projects. Friends also commented on the dated look of the site, and found the system inefficient with how much time had to be sent checking for new projects and the state of their preference lists.

3.5 Conclusion

The requirement capture phase played a crucial role in identifying the key issues and gathering valuable feedback from various stakeholders involved in the FYP allocation process. The discussions with Professor Thomas Clarke shed light on the challenges faced by both students and supervisors, emphasizing the need for a clearer and more efficient system. The surveys conducted among students provided insights into their preferences and behaviors, revealing important considerations

such as the importance of project skills, preference diversity, and the willingness to incorporate suitability rankings. Moreover, personal experiences highlighted the significance of improving the user interface and accessibility of the system. The findings from these activities will serve as a foundation for developing an enhanced FYP allocation system that addresses the identified issues and better aligns with the needs and expectations of the users.

Chapter 4

Analysis and Design

4.1 Introduction

Chapter 4 of this project focuses on the analysis and design phase, it begins at looking in depth into two of the most important app pages. Then the allocation problem is examined in detail, and the use Hungarian algorithm for solving it are evaluated. The chapter explores possible adaptations to the Hungarian algorithm, which is considered as a potential solution for the allocation problem. These adaptations include handling project duplications, dealing with algorithms that provide only complete solutions, handling unequal numbers of students and projects, and addressing the limitation of supervisors being able to supervise only a limited number of projects. Each adaptation is discussed along with its corresponding solution. Two different implementations of the Hungarian algorithm are then evaluated: jamespayor/weighted-bipartite-perfect-matching and mcximing/hungarian-algorithm-cpp.

The objective function for the allocation problem is discussed, focusing on the cost of assigning a student to a project. The code that interfaces the allocation algorithm and the Power Apps data is designed.

4.2 Frontend

The frontend of the system must have functionality to enable:

- Students to efficiently and easily find suitable projects
- Students to self propose projects
- Supervisors to create, edit and delete projects
- Supervisors to rank students suitability if they have selected one of their projects as a preference
- Supervisors to request duplicate or specifically allocate projects and the FYP Coordinator to review these requests
- Supervisors and the FYP Coordinator to review student proposed projects

The design below aims to implement this in one system through a set of forms. The outlines for the original layout of these forms are attached in the appendixA. The final pages differ from the original designs, these changes were necessitated due to limitations of Power Apps and different set ups working better in practice.

In conclusion with the information drawn from Section 2.3 directly looking at Table 2.1, the use of Microsoft Power Apps is considered the best option for the upcoming system due to its fast development, ease of use, and security features. The low-code aspect of Power Apps saves time, and integrating with Power Automate allows for helpful additions, such as automated emails, to improve the user experience.

4.2.1 Project Proposals Form

To facilitate efficient project searching and effective filtering, subcategories of information within project proposals will be incorporate. Below is Table 4.2.1 identifying the subcategories and their reasons for inclusion

Subcategory	Reason for Inclusion
Title	Allows for quick lookup by project title
Supervisor	Allows for quick filtering by project supervisor and contacting supervisor
Eligible Streams	Allows for filtering by stream
Project Focus	Has options: Embedded Systems, Control Systems, Electronics, Renewable Energy, Communications, Biomedical Engineering, System Optimisation and Modelling, High-performance Computing, Computer Vision, Digital Signal processing, Instrumentation and Measurement, Cybersecurity, Robotics, Signal Processing, Power Systems, Machine Learning, Photonics, and Other. By providing these choices, students can filter projects based on their preferred fields of interest. The extensive selection aims to encompass most project types, thereby minimising the number of projects falling under the "Other" option.
Candidate Requirements	Enables students to quickly see if they are eligible for the project
Desired Skills	Assists student in understanding if they may be highly suited to the project
Flexibility	Allows students to see how open the supervisor is to altering the project
Project Description	Provides an overview of the project's goals and scope
If Interested	Informs the student the next steps to take if they are interested in the project
Attachments	Lets supervisors attach any additional resources to there proposal such as graphs or diagrams

Table 4.1: Subcategories of Project Proposals and their Inclusion Reasons

To enhance the accessibility and user experience of the system, it is advisable to adopt a more user-friendly approach to displaying project proposals. Instead of presenting all project details at once, a condensed overview with the option to expand and access the complete information would be beneficial. This approach prevents students from being overwhelmed with excessive information and helps them navigate the available projects more effectively.

In addition, implementing a bookmarking feature would be advantageous. This functionality allows students to mark and save projects of interest, enabling them to easily track and revisit those projects later. By providing a quick and convenient way to access bookmarked projects, students can efficiently manage their preferences and stay organised throughout the selection process.

By incorporating these features, the system not only ensures ease of use but also enables students to maintain a clear overview of their preferred projects while avoiding information overload.

4.2.2 Student Preference Submission and Review Form

In order to achieve the most satisfactory allocation, it is crucial that students provide suitable preferences. However, it's important to note that the best preferences for a student may not always align with their favorite projects ranked in order of personal preference. While this simple approach is acceptable and students are free to adopt it, in many cases it can result in a low chance of allocation.

For instance, with machine learning projects gaining significant popularity in recent years, consider the scenario where a student who is not well-suited for machine learning projects lists five projects with a focus on machine learning, it is unlikely that they will be allocated to any of those projects.

To increase the likelihood of a successful allocation, students are encouraged to consider their suitability for specific project types rather than solely relying on personal preferences. By carefully assessing their skills, interests, and relevant background, students can make more informed choices

and prioritise projects that align well with their capabilities. This approach enhances the chances of a favorable allocation outcome, ensuring that students are assigned projects where they can contribute effectively and derive maximum value from the experience.

To aid in getting the students to put good preferences students can be given feedback on their selected preferences.

Feedback	**Explanation**
Suitability Ranking	The project supervisor's suitability ranking is a valuable feedback mechanism, as it represents their evaluation of the student's compatibility with the project. Students are advised that the suitability ranking significantly impacts their allocation likelihood. This guidance encourages students to carefully consider their preferences and avoid including lots of popular projects they are not well suited to.
Number of Students with Project in Top 3	The number of students including the project in their top three preferences is a useful feedback metric. It indicates the level of interest and demand for the project among students. Projects with a higher number of students ranking them in their top preferences are going to have a lower chance of being allocated to the student so this encourages students to look at less competitive projects that they may have not at first considered.
Communication with FYP Coordinator	Messages exchanged with the FYP coordinator provide valuable feedback on students' concerns, queries, and suggestions. The Coordinator will also be able to message first in the scenario that they think the students preferences may not be very suitable and they are risking not being allocated. This form of feedback helps identify areas of improvement, resolve issues, and enhance the quality of student preferences which in turn improves the overall project allocation.

Table 4.2: Forms of Feedback

4.3 Messaging

In order to facilitate efficient resolution of student queries and minimise the risk of missed emails, the implementation of mini messengers, inspired by the example found in [58], will be included. These mini messengers will be integrated into specific sections of the apps and involve the parties listed in table 4.3.

Table 4.3: Messaging Use-cases

System Section	**Parties Involved**
Student Preferences	Student and FYP coordinator
Allocation Requests	Project supervisor, student requested to allocate to and FYP coordinator
Duplication Requests	Project supervisor and FYP coordinator
Self Proposals	Proposed supervisor, student and FYP coordinator

The mini messengers will provide a convenient means of communication within these sections, allowing students to quickly address any concerns or inquiries they may have. By incorporating this feature, the system will enable prompt and direct communication, enhancing the overall user experience and streamlining the resolution of issues.

4.4 Allocation Algorithm

As the Hungarian algorithm is possibly the simplest solution to the FYP allocation problem, if it can be adapted to take into account all restrictions of the problem and execute in a timely manner, then it is the best choice for this project. The time complexities discussed in section 2.6.1 may prove unsuitable for a large set but if the size FYP allocation problem proves small enough, then the simplicity of the algorithm may make it the best choice for an efficient optimiser. The execution times of Hungarian algorithm implementations are tested in Section 4.6

4.5 Adapting the Hungarian Algorithm

The FYP allocation problem isn't a perfect match for the base Hungarian algorithm so some modifications may have to be made, depending on the implementation, to allow it to be used.

4.5.1 The Problem With Students Not Being Allocated

This is not an issue for some implementations, such as this $O(N^4)$ cost matrix solution [59], as all costs are checked, meaning that very high costs can be added between students and projects that they have not listed as preferences without affecting performance. If a student is assigned one of the high-cost projects, they will be deemed unallocated. The need for more complex approaches arises when considering graph implementations with time complexity involving the number of edges. Algorithms such as [60], with a time complexity of $O(NE)$, will only return complete solutions where all agents are assigned tasks. If the only edges included are the students' preferences, it is extremely likely that no solution exists where all students are assigned a project. To address this, edges could be added between all students and projects they have not listed as preferences, with high costs similar to the cost matrix implementations. However, this would result in a time complexity of $O(N^3)$. Alternatively, a dummy project could be added for each student, with a high-cost edge connecting them. In this scenario, add n projects equal to the number of students with one extra high-cost edge per student to one of the extra projects. If $|S| \approx |P|$, this roughly doubles the number of projects and increases the number of edges by $\left(\frac{N+1}{N}\right)$, where N is the number of preferences per student. Assignment to one of the dummy projects would be treated as an unallocation.

4.5.2 Dealing With An Unbalanced Problem

This is acceptable in the case of the cost matrix implementation here[59], as the matrix is not required to be square. However, in cases like this bipartite graph implementation[60], the algorithm does not behave correctly if the number of nodes on the left does not equal the number on the right. Adaptation is needed here, as it is unlikely that the number of students is equal to the number of projects, and this will definitely not be the case if we add dummy projects to deal with the allocation issue discussed in Section 4.5.1. This is an example of an unbalanced assignment problem[61]. One solution discussed here is to add a number of dummy nodes equal to the difference between the number of projects and the number of students, each with 0 cost edges connecting them to every node on the other side of the graph. However, if the dummy projects previously mentioned in section 4.5.1 have been implemented, then the number of projects will be around double that of the students. If we take the expected numbers of around 200 students and 230 projects, we end up with an additional $460 \times 260 = 119,600$ edges. An alternative method put forward is the doubling technique mentioned here[62], where both the left and right sides of the graph include all projects and students, and an edge is added between each node and its corresponding node on the other side of the graph. This leads to a total of $460 + 200 = 660$ new edges, a significant improvement.

4.5.3 Dealing With Supervisor Loading

Supervisors can only supervise a certain number of projects. This is the a complex issue to overcome. In the previous system, the inability to handle load balancing was one of the main driving factors in not treating the allocation problem as an LSAP[2]. The LSAP model does not

incorporate any sort of limit on the projects being allocated. Because of this, if it creates the lowest cost, the algorithm will assign all of a supervisors projects to students even if it is impossible for the supervisor to supervise that number of projects.

The issue is, it is not clear which of the supervisors projects should be taken out of the allocation pool to bring them down to supervising only the number of projects that they are able to. The solution is to, for each supervisor with this issue, add $T - L$ dummy students to the system, where T is the total number of projects that supervisor has proposed and L is the limit on the number of projects that can be supervised. Each of these dummy students will have one edge to each of the supervisors projects with cost zero. This will ensure that they always get allocated and remove $T - L$ projects from being allocated to students. The students will be allocated to the projects that add the most cost to the allocation, and so will keep the solution optimal. An example of this process can been seen in figure 4.1 where the green projects are all from the same supervisor with a T value of 3 and a L value of 2. The thick lines show the allocations that took place, and the number above the line is the cost of the allocation. It can be seen that the addition of the pseudo student causes student 3 to be allocated to project 4 even though it has double the cost of allocation to project 3.

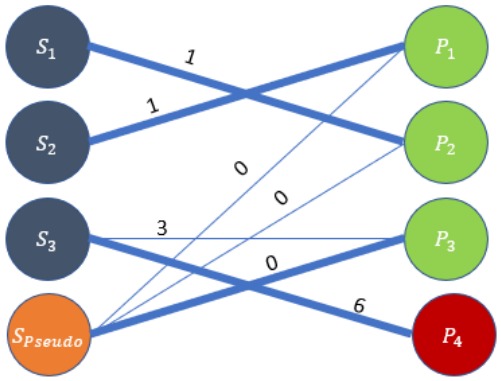

Figure 4.1: Example of how pseudo students help load balancing

4.5.4 Possible Adaptations Table

Projects can be duplicated	During reprocessing, an additional task will be added for each duplicated project with the same student preferences.
Some algorithms only give complete solutions	Create a dummy project for each student with a high cost edge connecting them, or add edges between all students and project they don't have preferences for. Further explanation of the issue and solution in section 4.5.1
The number of students isn't equal to the number of projects	The doubling technique, discussed further in section 4.5.2, will be applied on the final unbalanced system before being passed to the algorithm.
Supervisors can only supervise a limited number of project	As discussed in section 4.5.3, dummy students will be added with preferences such that the systems allocates them to projects that relieve overburdening on supervisors.
Some students are already allocated to a project or self-proposed a project	In preprocessing remove students/projects that are already allocated.

Table 4.4: Possible Adaptions to the Hungarian algorithm.

Although adaptions are required, in the table 4.5, solutions are presented that allow the problem to be modeled as an LSAP. These adjustments show that if the Hungarian algorithm is able to execute within a reasonable time at the size of the FYP allocation problem, then it can be used as the allocation algorithm in this project.

4.6 Algorithm Implementation

To be able to use the Hungarian algorithm[51] to solve the FYP allocation problem, an implementation must be found that can be adapted to the requirements discussed in table 4.4. The implementation must also execute within a acceptable amount of time for use in the project. In this section different implementations will be evaluated to see if they meet the required criteria.

4.6.1 jamespayor/weighted-bipartite-perfect-matching[60]

This was the first algorithm considered for the project with it being recommend by professor Clarke. It tackles the problem from a graph perspective and works on a vector of graph edges. The implementation is O(NE), N being the number of agents and E, the number of edges. Although the Hungarian Algorithm is considered an optimal solution, one of these implementations' drawbacks is that if a solution where all agents are assigned a task does not exist, the algorithm fails. The program will then return that there are 'no solutions'. This can be dealt with by adding a project for every student with a high cost edge between them as discussed in section4.5.1, doing this comes with the penalty of N new edges and creates a very unbalanced problem. Furthermore, this implementation cannot deal with unbalanced problems so preprocessing like earlier discussed in section4.5.2, must take place. Balancing the problem leads to additional complexity and many more edges. Finally, the implementation requires the supervisor loading adaptation discussed in Section 4.5.3, pre-allocations and duplicated projects will have to be removed in preprocessing.

Contextualising to the problem at hand, it is assumed that the number of dummy students is $<<$ the number of real students and S is used as the number of students, P as the number of projects and R as the number of preference rankings. The initial unadapted implementation has time complexity O(S^2R). After applying the dummy projects it becomes O($S^2(R+1)$). Then after balancing it becomes O($(2N+M)(S(R+1)+(2N+M)$)

4.6.2 mcximing/hungarian-algorithm-cpp[63]

This implementation is highly rated on GitHub[64] and is described as a "C++ wrapper with slight modification of a Hungarian algorithm implementation by Markus Buehren", the referenced implementation can be found here[65]. The code would be better described as a port, and, unlike the previous implementation, it views the problem from a cost matrix perspective. The implementation has the capability to handle unbalanced assignments and agents not being assigned. In such cases, the algorithm still returns an allocation, with the agent assigned to task -1. The implementation of the algorithm uses a cost matrix representation, which requires filling in every entry in the cost matrix. As a result, the algorithm's complexity is at best O(S^2P), where S represents the number of students and P denotes the number of projects. In contrast to the previous implementation, where using only 5 costs per agent limited the inclusion of a de-allocation cost that could discourage students from providing minimal preferences, this improved implementation addresses this concern by allowing for the incorporation of such costs.

The algorithm requires the supervisor loading adaptations discussed in section 4.5.3 however the number of dummy students should be $<<$ than the number of real students and so the time

complexity does not need to be adjusted. The same goes for project duplications and preallocations which should not impact the performance in a meaningful way.

The algorithm was tested with a randomly generated balanced allocation problem, where for each agent 5 costs were randomly set to values 1-5 and the rest were set to 1000 to simulate preferences. The number of agents/tasks was increased and the execution time measured. At each number of agents/tasks the program was executed 5 times and the average taken. The tests were performed on a balanced problem, as the implementation solves them in the same way as balanced problems, so making it balanced was simpler for testing and N and M will be very similar with the dummy students added to handle supervisor loading discussed in section4.5.3 . Below are the results in figure 4.2.

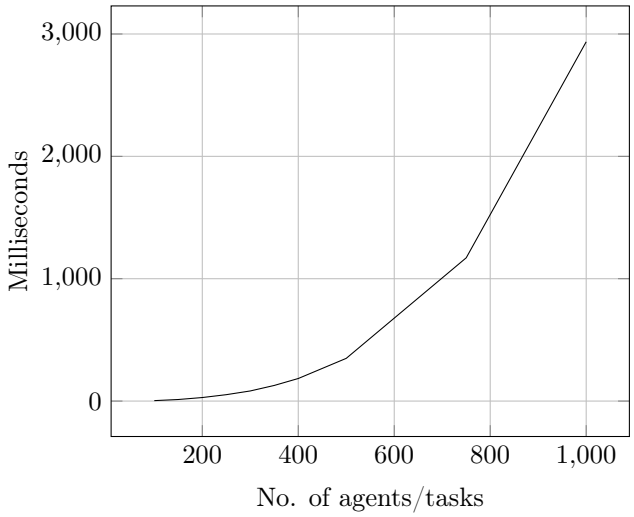

Figure 4.2: Graph showing the relationship between number of agent/tasks and execution time

4.7 Summary and Conclusion

Starting with jamespayor/weighted-bipartite-perfect-matching[60] in table 4.5 it allows to take advantage of the fact that the cost graph can be very sparse and its execution time is very fast. However, with the large amount of adaption and preprocessing required, plus the poor time complexity that comes with said adaptations, elude to this implementation not being very optimal in the context of the project.

Although slower by a factor of 10, mcximing/hungarian-algorithm-cpp[63] is alot more resilient to the EE allocation problem. In figure 4.2 it is shown, despite the time complexity and at the number of students and projects the system will have to handle, the implementation will be significantly faster than the heuristic search approach currently used disused in section 2.2.3. These factors make it the best choice of the implementations.

Furthermore, considering the solution from the Hungarian algorithm is optimal and the speed at which this implementation runs, the Hungarian algorithm is a better algorithm for the project and will be used in the system.

4.8 Student Suitability Ranking

The assessment of student suitability is divided into four distinct categories, each with its corresponding definition and equivalent cost. These categories, along with their definitions and associated costs, are summarised in Table 4.8. The suitability of a student for a specific project is a crucial factor that directly impacts the cost of assigning that student to the project. By considering the student's suitability level, the allocation process incorporates the necessary cost adjustments to ensure an optimal match between students and projects.

Implementation	Key Features	Summary
jamespayor/weighted-bipartite-perfect-matching[60]	Graph-based approach	• Does not require the cost of assigning a student to all projects • Time complexity: $O((2N + M)(S(R + 1) + (2N + M))$, where N is the number of students • Extremely fast 5ms execution time at typical EE allocating numbers before adaptations • Requires a lot of preprocessing and adaptations to handle unbalanced or incomplete solutions
mcximing/hungarian-algorithm-cpp[63]	Cost matrix representation	• Highly rated on GitHub • Doesn't require complete solution • Requires costs of assigning every student to every project. • Handles unbalanced assignments • Time complexity: $O(N^2M)$, where N is the number of students and M is the number of projects • Around 50ms execution time for typical EE allocation numbers

Table 4.5: Summary of different implementations

Table 3.1 examines the issue of supervisors sometimes employing questionable student rankings to manipulate the allocation process and obtain specific students. To address this concern, the project coordinator app includes a feature that enables the coordinator to access the student suitability rankings for each project. In cases where evident manipulation is identified, the app allows the coordinator to merge the categories of low, medium, and high suitability into a single category, effectively mitigating obvious instances of this issue. To aid in lessening the impact of less obvious instances of this problem, a weighting on suitability is discussed in section 4.9. However, it is important to note that the app does not directly tackle situations where supervisors incorrectly categorise students as unsuitable, which remains an unfortunate limitation of the system.

4.9 Objective Function

The Hungarian Algorithm requires costs of assigning an agent to a task and in this projects implementation, costs are calculated with student and supervisor rankings. The cost calculation can be described as an objective function.

Firstly we will look at the cost of assigning a student to a project that isn't in their list of preferences $C_{Unallocated}$. If the algorithm assigns a student to a project with this cost it is be treated as the student not being allocated a project. The natural course of action would be to assign a massive cost to these assignments to motivate the algorithm to find the most number of allocations possible. However, it is important to consider that through doing this the algorithm will potentially reward students who submit a low number of preferences. For example if a student only has two preferences they are much more likely to get one of those two projects and a student with the same top two but 5 other lower preferences. Two possible solutions are as follows; one is to use a lower cost that creates a balance between maximising allocations and fairness to students with a large number of preferences. The other is use the high cost but force students to submit at least a specified number of preferences and remove any preferences higher than this required number in reprocessing, the later having the drawback of some students simply not being suitable for that many projects, and loosing out on preference data. A mix of the two will be fleshed out in this project.

Suitability	Definition	Equivalent Cost
Unsuitable	Students who lack the necessary skills or knowledge for the project.	Preferences to projects where the student has been ranked Unsuitable will result in the preference being filtered out in preprocessing.
Low Suitability	Students who have some relevant skills but may require significant support and guidance.	3
Medium Suitability	Students with a good understanding of the project domain and the required skills.	2
High Suitability	Students who possess exceptional skills and knowledge directly applicable to the project.	1

Table 4.6: Suitability rankings, their definitions and equivalent costs

The cost of assigning a project to a student with the project in their preference list will be calculated as followed.

Case	Result (Cost)
$R_{Supervisor} \neq$ Not Suitable	$C = 3 * R_{Student} + 2 * R_{Supervisor}$
$R_{Supervisor} =$ Not Suitable	$C = C_{Unallocated}$

Table 4.7: Cost calculations

In the above equation $R_{Student}$ is the rank the student put the project in their preferences and $R_{Supervisor}$ is the rank the supervisor of the project signed off the student as. The preferences of the students are valued over supervisor sign-off ranks at a a ratio of 3:2. Student and supervisor preferences are both of great importance. Students need to be satisfied with their project in order to be motivated to put in the required amount of time and gain experience that will be valuable to them. Supervisors need to have suitable students in order to effectively supervise them, they don't have the time to be holding the hand of a student throughout a project so a low suitability student may find themselves often stuck and unable to make progress. The decision to place student preferences at a higher value has been made to further try to combat supervisors using the sign off as a personal preference ranking rather than a suitability ranking. Furthermore, students are the party that spend a disproportionate amount of time on the project and so trying to make as many of the them as possible satisfied their project is important.

This will be the default equation, however, if the FYP Coordinator sees fit they will be able to easily change the calculation to alter the importance of each ranking. They will also be able to set the cost of not allocating a student and how many student preferences are taken into consideration. This should allow the Coordinator to find a sweet spot for the specific years preferences. Although a brute forcing technique could be used to find the combination of hyperparameter that gives the lowest cost, it has to be remembered that that is not the ultimate goal, the algorithm needs to not be biased towards students with less preferences.

4.10 The Interfacing Code

4.10.1 Interfacing the C++ Algorithm with F#

The original plan involved compiling the selected C++ implementation of the Hungarian algorithm into a Dynamic Link Library (DLL) for utilisation in the F# code[31]. Initially, the compilation process seemed straightforward, as it was anticipated that the required functions could be directly exported to F#. However, complications arose due to the implementation's utilisation of a class item that proved challenging to export directly. Consequently, it became necessary to develop interface functions as intermediaries.

Another obstacle arose from the fact that the cost matrix used in the implementation was of type vector<vector<float», which does not have a direct equivalent in F#[66]. Therefore, in order to facilitate data transfer between F# and the C++ DLL, the implementation required the utilisation of marshalling. Marshalling involves the conversion of data from one representation to another, enabling interoperability between the two languages[67]. Specifically, when dealing with vectors, it is not possible to pass a C++ vector object by reference directly in F#. Instead, it is necessary to modify the C++ code to return a pointer to an array of integers.

To handle the potential memory management issues, careful consideration is required when integrating the C++ DLL into the F# code. When calling into a C++ DLL from F#, the responsibility for memory management typically lies with the C++ code. Since F# relies on the .NET Common Language Runtime (CLR), the CLR's automatic memory management system does not directly handle memory allocated within the C++ DLL. Therefore, it is important to devise a proper deallocation strategy in the C++ code to prevent memory leaks. The instance data of the C++ class used in the implementation needs to be managed, as it is not subject to garbage collection by default. This can be achieved by either using static data or implementing explicit memory deallocation mechanisms in the C++ code. Failing to address memory management adequately can lead to memory leaks, system crashes, or instability.

Considering the inherent complexities and debugging challenges associated with working across multiple languages and their interactions, as well as the project's limited timeframe, a decision was made to adopt a file-transfer approach for simplicity. This alternative approach involves generating the cost matrix from the F# pre-processing code and exporting it as a text file. Subsequently, the C++ implementation of the Hungarian algorithm will read this file to obtain the required input data.

By implementing this file transfer mechanism, the integration process is simplified; however, it introduces a performance drawback. The output allocation of the Hungarian algorithm necessitats post-processing. This post-processing step is crucial to facilitate the FYP Coordinator's review of the quality of allocations and to easily determine which students were assigned to which projects.

To enable this functionality, the preprocessing code needs to be executed twice. Firstly, to generate the cost matrix, and secondly, to regenerate all processed data. This regeneration is necessary to allow the postprocessing code to reference the updated data and determine the allocations for each student.

4.10.2 Accessing List Data

This alternative approach, although requiring a few minutes of interaction from the FYP Coordinator to manually export and transfer the CSV files, provides a viable solution within the project's limited timeframe. There is the issue of having to update the files when running trial allocations throughout the process, but once the apps have been closed off from users this will not be an issue. Moreover, the code is flexible and can be modified in the future to make an API call instead, should the need arise.

4.10.3 Mapping Data to the Hungarian Algorithm

To implement the Hungarian algorithm for student-project allocation, it is necessary to generate a cost matrix based on data from the SharePoint Lists.

Project Processing:

The processing of project proposals will involve considering allocation and duplication requests, resulting in a refined list of projects that contain only the necessary information. This list will include the total number of copies required and the number of copies already allocated. Simultaneously, a list of supervisors will be created, holding their current and maximum loading.

To ensure efficient tracking of approved allocations, a list will be established to maintain a record of students and the projects to which they are assigned. Furthermore, this allocation process will be updated to accommodate self-proposals that are accepted. Any new supervisors solely responsible for supervising self-proposals will be incorporated into the supervisors list, and the workload of existing supervisors will be adjusted accordingly.

Student Filtering:

In the PowerApps frontend, students are restricted from submitting preferences for certain projects. However, there is the possibility that if a student submits preferences for projects that subsequently become unavailable to them, those incorrect preferences may still pass through to the interfacing code. Preferences will be removed for the following reasons:

- The student is not in a stream that is eligible for the project
- The supervisor of the project is at maximum capacity
- All the copies of the project have been allocated through allocation requests
- The project has been removed by its supervisor
- It is not the first time the project has appeared in the students preferences

As each preference (excluding the first) has an equal to above option, the students preferences can not just be shifted down, they need to have their equality option updated. In the case when a preferences is removed that is not equal to its above preference, if the preferences below it have been set equal to the removed preference, then that preferences equality needs to be turned off.

To illustrate this case:

1. Suppose a student has preferences P1, P2, P3, and P4, in descending order of preference, where P2 is not equal to P1.
2. If P2 is removed from the list, and P3 has its equality option set to P2, then P3's equality relationship need to be updated.
3. Since P2 has been removed, the preferences below it (P3 and P4) should be shifted up and if the equal to above option of P3 is not updated then it will have the same priority as P1. Therefore, the "equal to above" option of P3 should be turned off.

By adjusting the equality options in this manner, it is ensured that the allocation process accurately reflects the students' preferences and maintains the appropriate equality relationships. It prevents the incorrect carryover of equality from removed preferences to the preferences below them, thus maintaining the integrity of the allocation results.

Before applying the Hungarian algorithm for project allocation, it is necessary to filter the student list. Students who already appear in the allocations list or have opted into being allocated by the Department of Computing system for one of their projects do not require allocation via the Hungarian algorithm. Therefore, their preferences can be discarded.

Project Filtering:

Next, the project list needs to be filtered to include only those projects that will be allocated using the Hungarian algorithm. Projects that meet any of the following conditions are excluded:

- There are no students who have expressed a preference for the project.
- The project supervisor has reached maximum capacity.

- The project has no remaining available copies.

The resulting filtered project list will serve as the basis for future allocation steps.

Pseudo Students:

As discussed in Section 4.5.3, the final project list will be complemented by the creation of pseudo-students. These pseudo-students facilitate the allocation process and ensure supervisors' loadings stay below their maximum.

Cost Matrix Generation:

With the final project list and student list (including pseudo students), each pair of list items is examined. Based on the preference and suitability rank of each pair, a cost is assigned. These costs form the cost matrix, which is subsequently output as a text file. The C++ implementation of the Hungarian algorithm will use this cost matrix for further allocation computations.

By following these steps, the allocation process is streamlined, ensuring that only eligible students and projects are considered, while generating the necessary cost matrix for the subsequent application of the Hungarian algorithm.

4.10.4 Post-Processing

After modifying the algorithm to generate a total cost and a list of index pairs indicating the allocation of students from the student list to projects in the project list, the interfacing code will undergo another run. This time, the newly created file will be used to process the index pairs and convert them into actual allocations to be added to the allocation list.

During this process, any index pairs where the project index is -1, or pairs with a cost equal to the unallocation cost, will be considered as unallocations. These unallocations will be tracked and recorded. Any allocations to pseudo-students will be ignored.

In addition to the allocations, the interfacing code will produce a histogram illustrating the distribution of preferences among the students, indicating how many students received which preference rank. This information will be valuable for the FYP coordinator's review.

Furthermore, the output will include details on the supervisor loading. This information will provide an overview of the workload distribution among supervisors, helping the FYP coordinator assess allocation quality.

By generating allocations, tracking unallocations, presenting preference distribution, and offering supervisor loading insights, the interfacing code will provide comprehensive data for the FYP coordinator to evaluate the allocation results and make informed decisions.

4.10.5 Hyper-parameters

The allocation process can be fine-tuned by adjusting hyper-parameters, enabling the FYP Coordinator to influence certain aspects of student-to-project allocation. Table 4.8 provides an overview of the different hyperparameters and their respective justifications.

These hyperparameters provide flexibility in shaping the allocation process according to specific objectives. By manipulating these parameters, the FYP Coordinator can guide the algorithm's behavior and achieve allocations that align with desired outcomes.

4.10.6 Conclusion

Chapter 4 concludes the analysis and design phase of the project, focusing on the allocation algorithm and the objective function. The chapter explores different implementations of the Hungarian algorithm and evaluates their suitability for the FYP allocation problem. Two implementations, jamespayor/weighted-bipartite-perfect-matching and mcximing/hungarian-algorithm-cpp, are discussed in detail, with the latter being selected as the preferred choice due to its resilience in handling unbalanced assignments and its reasonable execution time.

Furthermore, the chapter looks at key features to improve the effectiveness of allocations through two app pages. Features like competitiveness feedback are explored.

Table 4.8: Hyper-parameters for Student-to-Project Allocation

Hyper-parameter	**Justification**
Unallocation Cost	Allows adjustment of bias towards allocating as many students as possible and not punishing students who have put forward more than the required number of preferences
Suitability Multiplier	Adjusts the impact of suitability on cost
Suitability Ranks	Adjusts the costs associated with each suitability level
Preference Multiplier	Modifies the influence of preference on cost
Preference Ranks	Modifies the costs associated with each preference level
Disabled Levels of Suitability Cost	Sets the suitability cost for assigning students to a project with suitability costs disabled
Preference Cutoff	Discards preferences above a certain number to reduce chance of students who have put forward more than the required number of preferences getting a high preference
Required Preferences	Sets a minimum for the required number of preferences that the Coordinator has given students
Less than Required Preferences Cost Multiplier	Affects the cost of allocating a student who has provided less than the required number of preferences to a project.
Supervisor Cost Weights	To aid with evening supervisor loading each supervisors cost weight can be altered to affect the cost of allocating a student to one of their projects.

Finally, the design of the interfacing code is given analysing key important features and theyre reason for inclusion.

Chapter 5

Implementation

5.1 PowerApps

In total 3 Power Apps were created; FYP Student Tool, FYP Supervisor Tool, and FYP Coordinator Tool.

FYP Student Tool

The student app has 6 screens:

- Home
- Project List
- Preference Submission
- Manage Requests
- Manage Self Proposals
- New Self Proposals

FYP Supervisor Tool

The supervisor app has 8 screens:

- Home
- Project Proposal
- My Projects
- Request Management
- New Project Allocation
- New Project Duplication
- Student Ranking
- Manage Self Proposal Requests

FYP Coordinator Tool

The supervisor app has 7 screens:

- Home
- Project List
- Student Preferences
- Allocation Requests
- Duplication Requests
- Student Self Proposals
- Suitability Ranking

The report will not go into detail on every page created in Power Apps but instead focus on the ones that had interesting features to highlight and those that proved tricky to implement.

5.1.1 Project Discovery Page

Figure 5.1 depicts the dedicated page within the system where students can browse through the available projects. The design of this page draws inspiration from a tutorial video on PowerApps by Reza Dorrani [68].

The design of the project discovery page aims to provide an intuitive and user-friendly interface for students to explore and evaluate the available project options. It incorporates visual elements and layout techniques to enable the student to isolate one project at a time to not become lost or overwhelmed by information.

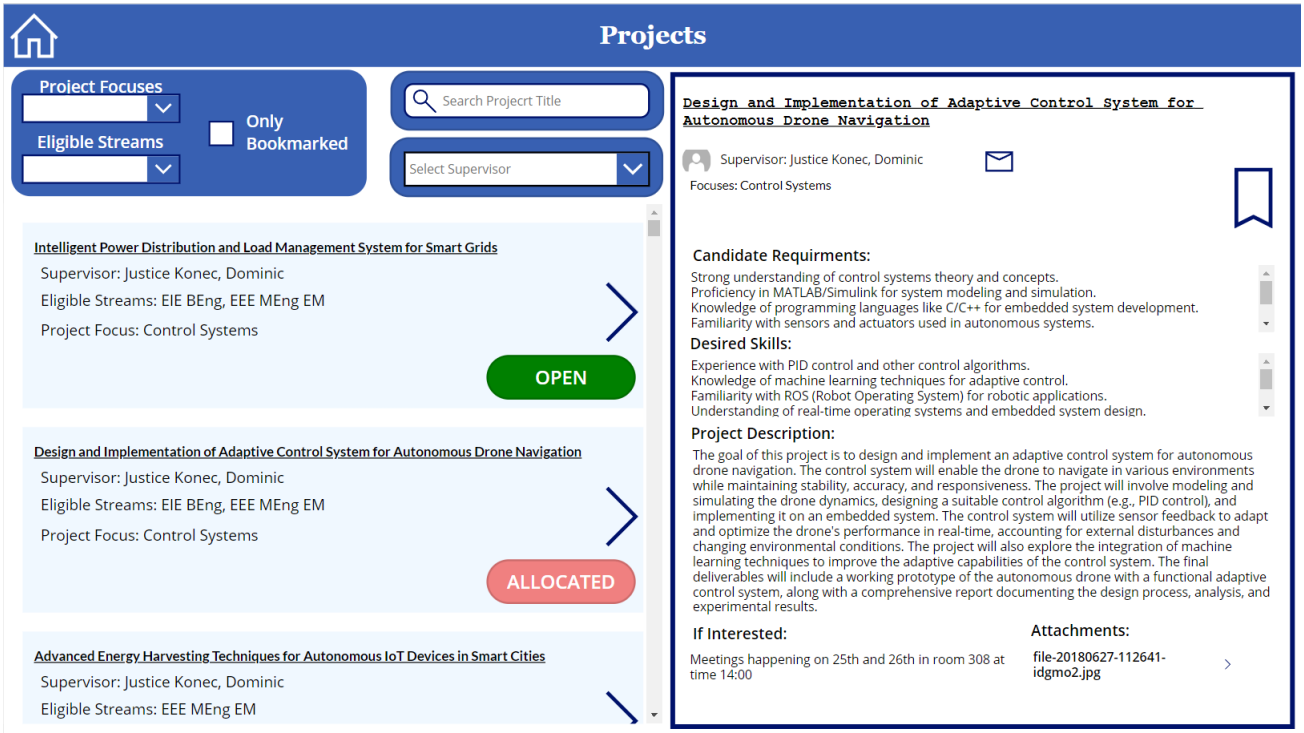

Figure 5.1: Students Project Discovery page

The page design prioritises simplicity and clarity. The layout optimises the use of screen space, ensuring that relevant project details are easily accessible without overwhelming the user with

excessive information. The gallery on the left allows students to gain a quick overview of the projects and when clicked brings up the projects information on the right detaching each project from the list to minimise confusion. The project information is broken into sections for easier digestion of information and there is easy access to bookmarking and emailing of the supervisor.

Moreover, the design aims to create a visually engaging and appealing environment vs the current systems plain text, leveraging concepts such as color schemes and typography to enhance readability and draw attention to important project attributes.

Filtering options created with help from another Reza Dorrani tutorial [69] allow students to sort through projects quickly and be efficient with their time.

PowerApps has proven to be efficient in creating a gallery to display the items in the project proposals backend list. However, it became less suitable for handling more complex logic, such as calculating the allocation status of projects, applying multiple filters simultaneously, and implementing bookmarking functionality.

The challenges with PowerApps arise from its limitations in setting variables on non-behavior attributes, which leads to a significant amount of duplicate code and a lack of abstraction. This lack of flexibility results in long and inefficient code snippets, making it difficult to read and maintain. PowerApps solution of invisible components holding information to be reused is only applicable in limited situations. The inefficiency was clear, when applying multiple filters, the absence of identifiers forces the filters to be nested in a single block, increasing code complexity and reducing readability.

Calculating the allocation status also presents difficulties due to the inability to save the first use of the LookUp function [70] (used to look up records in external data sources) and access its attributes efficiently. This limitation necessitates repetitive lookups, adding unnecessary overhead to the code.

Regarding bookmarking projects, the student's account item is patched into the "Has Favourited" column of the proposals. Unfortunately, there is no direct way to add the student without overwriting existing members in the column. Consequently, the selected project item, set when displaying more information for a project, has the student added to its "Has Favourited" attribute, and the updated collection is patched to replace the existing one. If another student has favorited or unfavorited the project, or if the project information has changed, the patch operation will fail, requiring the user to re-bookmark the project. The same process applies to unbookmarking, where the student is removed from the selected project's "Has Favourited" column.

To provide visual cues to the user, the bookmark icon of a project becomes filled when it is bookmarked. The visibility attributes of the filled and unfilled icons are configured to check for the student in the selected project's "Has Favourited" column. However, the Collect/RemoveIf function used to add or remove the student from the existing list of people who have favorited the project does not properly update the selected project variable for visibility calculations. This issue resulted in significant debugging time and effort. Eventually, adding the counter-intuitive line

```
"Set(selectedproject, selectedproject);"
```

at the beginning and end of the bookmarking updating behavior resolved the problem.

These limitations and challenges with PowerApps necessitated careful consideration and workarounds to achieve the desired functionality, resulting in additional development time and effort.

5.1.2 Messaging

The message component was created to allow quick communication between parties. The final product is shown in figure 5.2.

PowerApps made implementing messaging functionality quick and simple. A Microsoft List was used to store all the messages. To organise conversations, a unique title was assigned to each conversation. Using a gallery component with filtering capabilities, it was straightforward to display the messages associated with a specific conversation.

To retrieve the relevant messages for a specific conversation, a filter function was employed. The filter function, with its full delegation support, ensured no issues would arise even as the number of messages grew significantly [71].

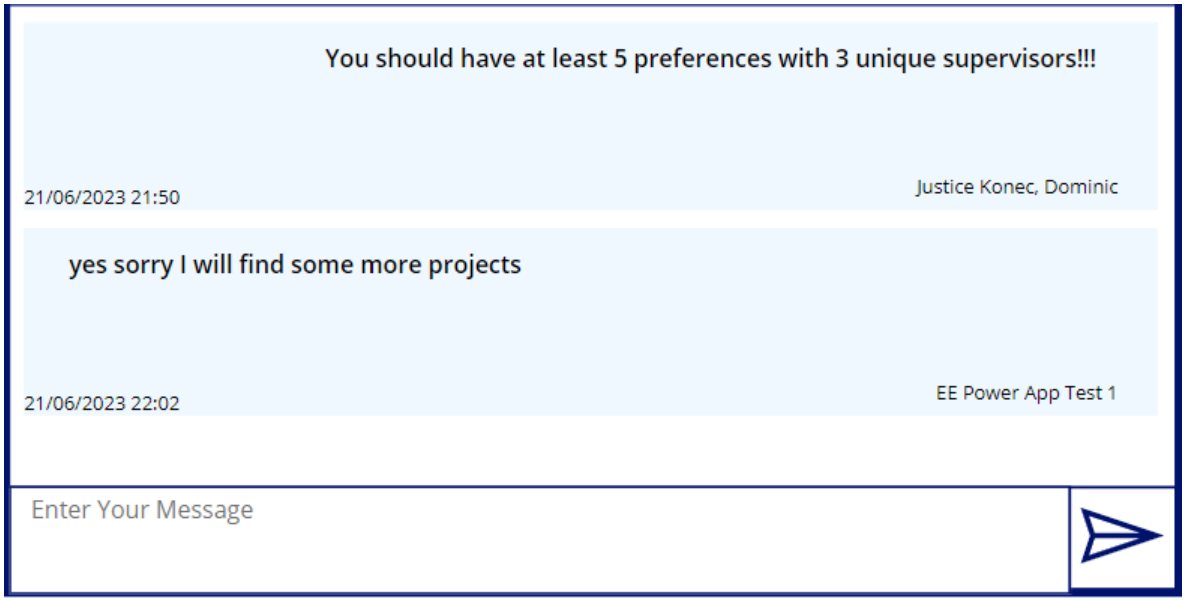

Figure 5.2: Messaging component used in PowerApps

For submitting new messages, a simple text input field combined with a patch function was utilised. The OnSelect property of the send icon is a behavior property and so variables were leveraged, allowing for the creation of abstracted and readable code to handle the slightly complex patching process.

Unfortunately, due to the varying title styles of different conversations, it was not possible to create a reusable PowerApps component for the messenger. However, a workaround was implemented by grouping individual components together and duplicating the group while making necessary code adjustments.

Overall, PowerApps excelled in implementing messaging. Although the messenger component could not be made entirely reusable, the implemented approach allows for effective communication within the system.

5.1.3 Preference Submission Page

The limitations of Microsoft Lists necessitate a specific approach to recording students' preferences. Each individual preference (excluding the first one) is recorded using two columns: a lookup column referencing the preferred project and a yes/no column indicating whether the preference is equal to the one above it. However, Microsoft Lists have a restriction of 12 lookup columns in a single view. Since all preferences need to be in one view to be accessed, this poses a limitation on the number of preferences students can submit.

ref future work

In our system, there are 8 available slots for preferences, which is higher than the typical preference cutoff set by the FYP Coordinator, making it unlikely to exceed this limit. Nonetheless, will explore two potential ways to increase this limitation and allow for a larger number of preferences to be recorded.

Figure 5.3 shows the student page for submitting their preferences and receiving feedback on them.

The submission of preferences in the system is facilitated through a dynamic form that adjusts its behavior based on the student's progress. The form utilises the DisplayMode.New mode to add a new row to the StudentPrefs List when students initially submit their preferences. Subsequently, it switches to DisplayMode.Edit mode to allow students to modify their existing preferences.

One notable feature of the form is the dynamic heading of each preference, which reflects the state of the "equal to above" toggles. This ensures that students receive real-time feedback on the status of their preferences, clearing up confusion.

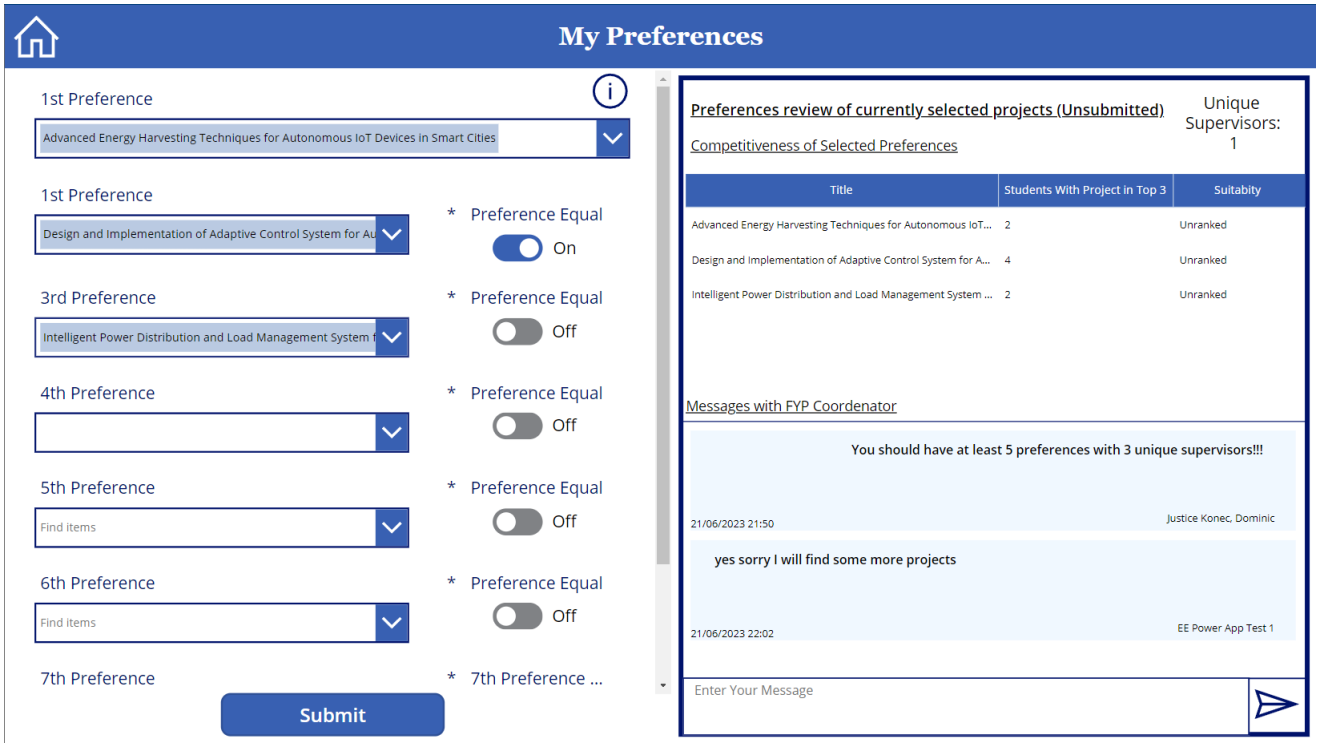

Figure 5.3: Students preference submission page

To provide additional guidance and support to students during the preference submission process, an information icon is included. Clicking on this icon triggers a pop-up window containing essential information on submitting preferences and advice on making wise project choices. This feature aims to assist students in understanding the preference selection process better and helps them make well-informed decisions regarding their project choices.

The "Submit" button serves multiple functions, including checking for gaps and duplicate preferences before submission. Additionally, it is responsible for adding the student to the awaiting ranking columns of the project proposals for the preferences they have submitted, provided they are not already listed or have been ranked. However, implementing this functionality proved to be more time-consuming than expected, resulting in code that was not as clean as desired.

PowerApps has a limited code editing space, accommodating only around 7 lines at a time. Combined with the substantial number of lines required for the duplicate lookup functions, which determines the student's ranking status, it became challenging to track and edit the code effectively. Although the button's OnSelect property is a behavior property, the need to perform the same actions for each preference necessitated the use of a ForAll function [72]. However, the ForAll function does not allow for variable assignment within it, making it less ideal. It would have been preferable to utilise a custom function like the one shown below:

```
function(pref, studentRank):
    !(true in ForAll(
        LookUp(
            ProjectProposals,
            Title = pref,
            studentRank),
        DisplayName = Office365Users.MyProfile().DisplayName
    ))
```

This would have improved code readability by reducing duplication, even though it would still involve unnecessary lookups. The ability to create functions that accept properties requires enabling

the experimental feature "Enhanced component properties", creating a custom component with a custom property that holds the function and placing the component on the page [73]. Not only is this process convoluted for the little reward, this feature does not have access to data sources [73] [74], limiting its usefulness. In this case, the result of the lookup could have also been passed into the function but it was decided that, as this was the only place the function would be used, it was not worth the additional development time. Another approach commonly used to mimic functions is to utilise an invisible button component that hold the code in their OnSelect property, which can be triggered from other components. However, this method does not allow for supplying arguments, resulting in the need to create separate buttons for each preference option box and execute their respective code within the "Submit" button. While this approach would allow for variable usage reducing overhead, it would result in convoluted code within multiple nearly identical buttons and it would not be as easy to expand the number of preferences. Also the extra lookups didn't seem to hinder performance significantly.

This was once again a place where PowerApps was not able to provide an efficient coding experience when faced with a more complex task.

During the creation of the update to be patched into the project proposal for updating the "Students awaiting ranking" column, some peculiar behavior was observed. The patch update is as follows:

```
{
    'Students awaiting ranking': Collect(
        students,
        student,
        LookUp(
            ProjectProposals,
            Title = pref
        ).'Students awaiting ranking'
    )
}
```

In this snippet, the "students" collection is created solely for the purpose of being used in this patch function. Ideally, it should not be necessary, as the Collect function can take an existing collection [75], allowing the Collect function to only include the lookup followed by the student. When implemented in this manner, the linter does not raise any complaints, and there are no execution errors. However, the Lists in the backend are not correctly updated.

The inclusion of the "students" collection introduces additional issues, as it is within a ForAll block that does not permit the use of the Clear or ClearCollect functions [75] [72]. This presents a problem because it means the "students" collection is not cleared between loops of the ForAll block, resulting in the merging of students from different projects' "Students awaiting ranking" columns, causing incorrect patches.

Fortunately, a workaround was discovered [76] by utilising the RemoveIf function [77] with the condition set to true, effectively clearing the collection at the end of each ForAll loop. It is important to note that with the clear functions disabled in PowerApps, there is a potential scenario where an update could be released that disables the RemoveIf function within ForAll blocks, thus breaking the preference submission system.

To simplify the setup of the supervisor's ranking system and avoid creating an additional column for each preference's suitability, the suitability of students for a project is stored as a multi-person column on the project proposal list item. However, this approach introduces certain scenarios where conflicts may arise. For instance, when both the supervisor is ranking students for a project and students are submitting preferences for the same project simultaneously, or when multiple students are submitting preferences for the same project concurrently.

In such cases, the "Submit" buttons' patch operation to the ProjectProposals list may fail because the students' awaiting rankings column will become out of sync after the initial successful patch. This poses an issue as the patch to the StudentPrefs list can still succeed, potentially leaving a student with a preference for a project for which they are not included in a suitability column.

During the allocation data processing, it would be beneficial to consider treating these students

as highly suitable to avoid penalising them for the system's mistake. However, this introduces the possibility of gaming the system, as students could attempt to coordinate synchronised submissions to deliberately trigger a failed patch. By not resubmitting their preferences afterward, they would guarantee being counted as highly suitable.

To mitigate this risk, students who are not included in a suitability column are deemed unsuitable. It is crucial to emphasize to students the importance of checking the suitability column in the preference review table. If any rows indicate "unsubmitted," it signifies the need to resubmit their preferences; otherwise, they will not be considered suitable for allocation.

The preference review table provides students with feedback on their suitability for a project and the project's competitiveness. However, its implementation encountered similar challenges to those faced by the "Submit" button. There was a prevalence of duplicate code and additional lookups, which once again resulted in a time-consuming and cumbersome process.

5.1.4 Student Ranking Page

The page dedicated to supervisors ranking students based on their suitability was a critical component that required careful attention. During the requirement capture phase, it became evident that many staff members had encountered difficulties and experienced bugs, resulting in an inefficient process. Figure 5.4 showcases the page developed to address this requirement within the system.

Figure 5.4: Student ranking page

The page incorporates five galleries, each associated with one of the project's suitability columns. When a student within a gallery is selected, the option to move the student to another column becomes available through side buttons. However, implementing the desired functionality for these buttons proved to be challenging and required meticulous attention to detail. The complexity stemmed from the need to add and remove students from different multi-person columns within the same patch, while ensuring synchronization issues were avoided. Care had to be taken to prevent students from appearing in multiple columns or being inadvertently removed from all

columns. Effective use of variables and their attributes was crucial to avoid redundant copies of the project columns. Additionally, PowerApps' peculiar typing of variables and variable attributes added to the complexity and confusion. Figure 5.5 illustrates an example where comparison was disallowed in an if statement but allowed in a switch statement, despite the earlier setting of '**SelectedStudentRank**' to '**selectedproject.'Highly suitable students'**'. Such issues created confusion and prolonged the implementation process.

The final code for the "Unsuitable" button is shown below:

```
UpdateContext(
    {
        lookedUpProject: LookUp(
            ProjectProposals,
            Title = selectedproject.Title
        )
    }
);
ClearCollect(
    SelectedStudentRankCollection,
    SelectedStudentRank,
    {}
);
RemoveIf(
    SelectedStudentRankCollection,
    Email = selectedstudent.Email
);
ClearCollect(
    newSuitabilityGroup,
    lookedUpProject.'Unsuitable students',
    selectedstudent
);
Switch(
    SelectedStudentRank,
    lookedUpProject.'Students awaiting ranking',
    Patch(
        ProjectProposals,
        selectedproject,
        {
            'Students awaiting ranking': SelectedStudentRankCollection,
            'Unsuitable students': newSuitabilityGroup
        }
    );
    ,
    lookedUpProject.'Low Suitability Students',
    Patch(
        ProjectProposals,
        selectedproject,
        {
            'Low Suitability Students': SelectedStudentRankCollection,
            'Unsuitable students': newSuitabilityGroup
        }
    );
    ,
    lookedUpProject.'Highly suitable students',
    Patch(
        ProjectProposals,
        selectedproject,
        {
```

```
                'Highly suitable students': SelectedStudentRankCollection,
                'Unsuitable students': newSuitabilityGroup
            }
        );
        ,
        lookedUpProject.'Medium suitability students',
        Patch(
            ProjectProposals,
            selectedproject,
            {
                'Medium suitability students': SelectedStudentRankCollection,
                'Unsuitable students': newSuitabilityGroup
            }
        );

);
Set(
    selectedstudent,
    Blank()
);
Refresh(ProjectProposals);
```

```
Switch(SelectedStudentRank, selectedproject.'Highly suitable students', Notify("hi"));
If(SelectedStudentRank = selectedproject.'Highly suitable students', Notify("hi"));
```

Figure 5.5: Example of weird PowerApps Typing

5.2 SharePoint Permissions

To enable access to their respective app, each party must be granted permission to the Microsoft Lists that serve as the backend storage for information. However, this introduced a significant challenge because users who accessed the backend lists gained access to data that was not intended to be available through the app. For instance, students could view other students' preferences, which clearly posed a problem.

Attempts were made to address this issue by configuring certain settings for the lists. For example, restrictions were put in place to ensure that users could only view and edit rows they had created. However, this approach did not resolve all the problems. Despite these settings, users could still modify their preferences to include projects that were meant to be restricted through the app's interface.

Furthermore, it was discovered that when the settings were enabled, although other users' preferences appeared blank, any attempt to edit them, even though it was not possible, resulted in the preferences being wiped entirely.

Amine Halimi was contacted and extensive research was conducted to tackle the issue at hand. In a response by user stevegeall in a Microsoft forum post[78], a method was outlined to provide users with the required permissions on the necessary lists for utilising PowerApps. This approach ensures that when users attempt to access the app, they are met with an "Access Denied" page, effectively safeguarding them from viewing any private data. A slightly modified version of the aforementioned method was implemented to accommodate the specific changes in the situation. Tests were conducted to validate its effectiveness, and it was found to work successfully. However, it is worth noting that the implemented solution is quite convoluted and highly specific, possibly indicating that it may not be an intended feature. It is important to mention that the method

was developed in 2020 and remains functional, but there is a possibility that future updates could potentially disrupt its proper functioning.

5.3 The Interfacing Code

5.3.1 Types Module

The `Types` module provides type definitions and data providers for reading CSV files. It includes the following type definitions:

```
module Types

open FSharp.Data

[<Literal>]
let ResolutionFolder = __SOURCE_DIRECTORY__

type CsvProjectData = CsvProvider<"./csvs/ProjectProposals.csv",
    ResolutionFolder=ResolutionFolder>
type CsvStudentPrefData = CsvProvider<"./csvs/StudentsPrefs.csv",
     ResolutionFolder=ResolutionFolder>
type CsvStudentData = CsvProvider<"./csvs/StudentInfo.csv",
    ResolutionFolder=ResolutionFolder>
type CsvAllocationRequestData = CsvProvider<"./csvs/
    AllocationRequests.csv", ResolutionFolder=ResolutionFolder>
type CsvDuplicationRequestData = CsvProvider<"./csvs/
    DuplicationRequests.csv", ResolutionFolder=ResolutionFolder>
type CsvSupervisorLoadingData = CsvProvider<"./csvs/
    SupervisorLoading.csv", ResolutionFolder=ResolutionFolder>
type CsvSelfProposalData = CsvProvider<"./csvs/SelfProposals.csv"
    , ResolutionFolder=ResolutionFolder>

type allocation =
    { StudentEmail: string
      ProjectTitle: string
      // -1 = allocation request, -2 = self proposal, -3
         unallocated
      Preference: int }

type supervisor =
    { Email: string
      MaxLoading: int
      NumberOfAllocations: int
      TotalCopiesOfProjects: int
      CostWeight: float }

type necessaryProjectData =
    { Title: string
      SupervisorEmail: string
      SupervisorName: string
      EligibleStreams: List<string>
      LowSuitabilityStudents: list<string>
      MediumSuitabilityStudents: list<string>
      HighSuitabilityStudents: list<string>
      UnsuitableStudents: list<string>
      UnrankedStudents: list<string>
```

```
        DisableLevelsOfSuitability: bool
        RemovedBySupervisor: bool
        NumberOfAllocations: int
        NumberOfCopies: int }

type necessaryStudentData =
    { StudentEmail: string
      StudentName: string
      Preferences: list<string * int>
      Stream: string }

type Preference =
    { ProjectName: string
      EqualToAbove: bool }

type pseudoStudentGroup =
    { projectsToSelect: list<necessaryProjectData>
      NumberOfPseudoStudents: int }
```

In the `Types` module, the `CsvProvider` types are used to read data from CSV files. The defined types such as `allocation`, `supervisor`, `necessaryProjectData`, `necessaryStudentData`, `Preference`, and `pseudoStudentGroup` are used to hold and manipulate only the necessary data from the csvs.

These types provide a clear structure and enable the code to handle the necessary data effectively.

The `Types` module serves as the foundation for the interfacing code, providing the necessary types and data providers for reading CSV files.

5.3.2 Data Manipulation Module

The `DataManipulation` module contains various functions for processing and manipulating data from the CSVs. Here is a breakdown of the modules functions:

- `shuffleList`: This function shuffles the order of a given list using a random number generator. It ensures that the order of the input CSVs do not bias the allocations. A seed can be set in hyperparams to make sure both runs of the main have the same shuffling.
- `parseStringOfList`: This function takes a string and a pattern and returns a list of strings by extracting items from the string based on the given pattern. It uses regular expressions to match and extract the desired substrings. This is needed as the exported csvs have lists of people and streams in strings.
- `processPreferences`: This function converts a student's preferences from the CSV format into a list of `Preference` type. It creates `Preference` objects for each preference, including the project name and whether it's equal to the above preference.
- `createOrUpdateSupervisor`: Given a list of supervisors, this function looks up the supervisor's email and increases their `NumberOfAllocations` and `TotalCopiesOfProjects` by a specified amount. If the supervisor doesn't exist in the list, a new supervisor is added with the appropriate values.
- `processProjects`: This function processes project data from various CSVs and generates a list of `NecessaryProjectData` containing information about each project. It handles duplication and allocation requests. It also returns a list of supervisors who supervise the projects and a list of allocations that came from allocation requests.
- `processSelfProposals`: This function processes self-proposal data from the CSV and updates the supervisors and allocations lists accordingly. It adds new supervisors who only have self-proposals and updates existing supervisors if they are supervising a self-proposal.

- `processStudentPrefsCSV`: This function processes student preferences data from the CSV and generates a list of `NecessaryStudentData`. It removes any preferences for projects that are not available to the students and reorders the preferences. It also identifies students whose preferences were altered or removed due to opting into a doc allocation, already being allocated a project, or having an accepted self-proposal. It tackles the tricky issue or reordering students preferences to give them the best chance of allocation when some of their preferences are found to be inappropriate or have gaps.

- `generatePseudoStudents`: This function moves through the supervisors list detecting thoses you are not at max capacity and have more project copies available than max loading. For each of these supervisors a pseudoStudentGroup is created which holds the projects that the pseudo students need to have preferences for and the number of the pseudo students that are neccessary in order to stop potentially overloading the supervisors.

These functions handle tasks such as shuffling lists, parsing strings using regular expressions, processing project data, updating supervisor information, handling student preferences, and identifying where pseudo students are needed.

5.3.3 HyperParams Module

In this subsection, we will discuss the default values chosen for the hyperparameters in the `HyperParams` module. These default values were carefully selected based on considerations and objectives specific to our student-to-project allocation process. Below, we provide a rationale for each default hyperparameter value:

- **Suitability Multiplier and Preference Multiplier** (`SUITABILITY_MULTIPLIER` & `PREFERENCE_MULTIPLIER`): These values where chosen to create a 3:2 ratio between the importance of preferences and suitability ranking. This was done to try to lessen the impact of supervisors submitting disingenuous suitability rankings.

- **Disabled Levels of Suitability Cost** (`DISABLED_SUITABILITY_LEVELS_COST`): A cost value of 1.0 is assigned to projects when suitability costs are disabled. This default treats projects with disabled levels of suitability as having all students signed off as highly suitable.

- **Preference Cutoff** (`PREFERENCE_CUTOFF`): The preference cutoff value of 5 was chosen this mirrors 2023s allocation system.

- **Required Preferences** (`REQUIRED_PREFS`): The minimum required number of preferences is set to 5 to mirror the 2023 allocation system.

- **Less than Required Preferences Cost Multiplier** (`LESS_THAN_REQUIRED_PREFS_COST_MULTIPLIER`): A value of 1.5 was selected to moderately increase the allocation cost for students who have provided fewer preferences than required. This encourages students to meet the minimum preference requirement and provides a fair penalty for those who don't.

- **Supervisor Cost Weights** (*not in the provided code*): These are set on the supervisor loading list and should be set to 1 and adjusted when necessary.

- **Unallocation Cost** (`UNALLOCATION_COST`): The value of 100.0 was chosen as the required and cutoff for preferences is the same so the system should be maximally against leaving students unallocated.

The default costs for the different suitability ranks and preferences ranks is 1 through 3 and 1 through 8 respectively.

Although these are the default hyperparameter values, it is important to note that these values are not fixed and can be adjusted by the FYP Coordinator according to the specific needs and constraints of our allocation process.

5.3.4 Main

The main of the program is responsible for reading in the CSVs and applying the processing functions too them. Once done, the following steps are applied.

- Removing redundant projects before using the list for cost matrix generation and adjusting the supervisors number of projects accordingly
- Generating the pseudoStudentGroups that are necessary
- Looking at every combo of student and project and calculating the cost in a matrix form
- Adding pseudoStudent costs to the bottom of the matrix
- Exporting the costs to the allocation algorithm
- Importing and processing the resulting allocation

This main code runs twice once to generate the costs and the second to process the allocation results of the Hungarian algorithm implementation.

Chapter 6

Evaluation

6.1 Gameability of the System

During the requirement capture phase, the system's gameability was a significant concern. Both students and staff expressed concerns that students might act disingenuously with their preferences in order to manipulate the system and secure a specific project. As discussed in section 3.2.2, a certain level of strategising from students is not inherently negative. However, it is crucial to prevent students from maliciously manipulating the system.

To address this concern, the system underwent an analysis to ensure that students cannot obtain direct numerical feedback on their preference combinations. This prevents them from calculating their allocation likelihood and using multiple preference combinations to achieve a desired outcome. Additionally, measures were implemented to prevent students from accessing the preferences of their peers within the same cohort. Such access could potentially enable students to conduct their own trial allocations if they understand how costs are calculated.

Tactics like those discussed in Section 5.1.3 were considered by the students, and corresponding countermeasures were implemented to mitigate potential instances of foul play. While these measures were designed to address known risks, a more comprehensive investigation in the future may be required to ensure complete assurance against any possible manipulation or misconduct. By proactively considering and implementing these measures, the system aims to maintain fairness and integrity in the allocation process.

6.2 Interfacing Code Performance

As analysed in Section 2.2.3, the current allocation system has execution times that, if this project is successful, the new system should be able to significantly outperform.

Tests were conducted to evaluate the execution time of the new allocation algorithm as the number of students and projects increased while keeping them equal. The results, presented in Figure 6.1, indicate that the execution time remains below 30 seconds for allocations involving up to 1000 students and projects. Furthermore, the resulting allocation is confirmed to be optimal, providing a significant improvement compared to the previous allocation method, which took approximately half an hour and did not guarantee optimal allocations.

Interestingly, there was a spike observed in the results at 500 students/projects. This anomaly was not a random error, as the tests were repeated five times, and the average was taken. Although the cause of this spike remains unknown, it does not affect the overall conclusion that the system performs well within the expected requirements. The system is designed to handle approximately 200 students and 250 projects, and the graph clearly indicates ample room for scalability as the department grows.

However, it should be noted that as the number of students/projects increases further, the execution time exhibits a rapid increase. This suggests that the algorithm may not be suitable for significant expansion of the system. Additionally, Power Apps itself has limitations on expansion, which will be discussed in detail later in this section.

The improvement in execution time achieved by the new allocation algorithm will allow the FYP Coordinator to run trial allocations much faster than in previous years. This enhanced efficiency will greatly facilitate the overall process of generating improved final allocations.

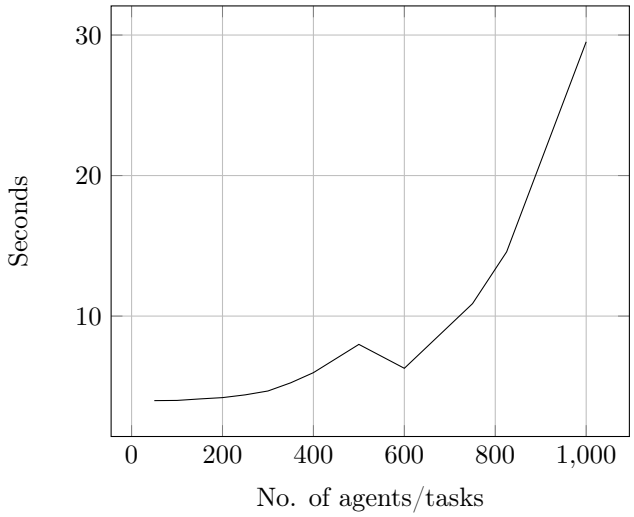

Figure 6.1: Graph showing the relationship between number of students/projects and execution time in the final interfacing code

6.3 User Testing

In order to evaluate the effectiveness of the project in improving the speed and efficiency of user actions in the FYP (Final Year Project) process, both students and the FYP Coordinator were asked to perform actions on both the old system and the new system. Subsequently, they provided feedback on their experience with each process. Their feedback is accessed in the following subsections.

6.3.1 Student Feedback

Three different students were invited to provide feedback on the new system.

Project Discorvery Page

When discussing the project discovery, filtering, searching capabilities, and bookmarking feature, overall, all three students acknowledged the significant improvement in project discovery compared to the old system, attributing it to the enhanced filtering and searching functionalities. One student even emphasized the importance of these features and expressed their view that the old system appeared outdated without them. Another said the live status of the projects helped to not waste time on projects that weren't available anymore.

The bookmarking feature also received positive feedback, with students finding it helpful for saving and organizing projects of interest. However, one student mentioned bookmarking feeling a bit buggy. It was observed that there is a delay in updating the bookmark icon after clicking, which can lead to multiple clicks causing an error to be displayed. While this issue is unfortunate, it suggests a potential workaround for future development. One possible solution could involve handling the bookmarking process locally first, before passing the information to the lists, thereby mitigating the delay and reducing the occurrence of errors.

Self Proposal Page

Regarding the self proposal page, students expressed satisfaction with the quick and efficient process of creating proposals within the system. They appreciated the convenience of having the self proposal feature integrated into the system, as it facilitated easier tracking of the proposal's status. Furthermore, students highlighted the messaging functionality as a great addition for seamless communication.

However, one student pointed out that the new self proposal button was easily overlooked, prompting the relocation of the button to a more visible location.

Preference Submission Page

Once again, all students expressed their satisfaction with the messaging feature integrated into the preference submission page. They found it beneficial to have the preference review incorporated on the same page, as it saved time and reduced complexity by providing immediate information about their suitability for certain projects. One student also appreciated the feedback on competitiveness, as well as the removal of non-eligible projects from the available options.

Request Management

Students did not have extensive feedback on the request management section. However, they found the messaging feature to be helpful, and they appreciated the convenience of handling requests within the same application as other functionalities.

Accessibility

The new system being more accessible is a goal of the project. Because of this it was chosen that one of the surveyed students should have accessibility issues. This student provided additional feedback on the accessibility of the new system. They emphasised that the old system was overwhelming due to the display of all information at once in a single block. The improvements in the new system, such as expandable and segmented information, as well as clear visual separation, significantly enhanced accessibility in the project discovery feature. The use of different colors and the repeated segmentation of information were particularly appreciated by the student, as they made the system more visually distinct and easier to comprehend.

6.4 Framework Evaluation

6.4.1 Power Apps and Microsoft Lists

PowerApps in combination with Microsoft lists had positives and negatives about its selection.

Cons	Description
Small Code Editor	The Power Apps Code Editor initially appears as a single line but can be expanded to display seven complete lines simultaneously. This limited view posed significant challenges when writing intricate code, particularly considering the inherent difficulty of ensuring code readability in Power Apps. Oftentimes, copying and pasting code into external notes was resorted to, depriving access to essential features such as linting and autocompletion, thus further extending the development process.

Poor Linter	The Power Apps code editor incorporates a linter; however, it tends to be less precise in identifying the exact location of errors. Moreover, the error messages that it generates often lack helpful information, making troubleshooting more challenging. In situations where substantial changes are made, impacting numerous components, Power Apps can experience a breakdown, resulting in all components being marked as erroneous. Unfortunately, the only remedy for this situation is to reload the page, which can be time-consuming within the Power Apps platform.
Inconsistent Preview	During Power Apps development, it is possible to preview the app as the end user would see it. However, when previewing the supervisor app, it was observed that the preview window had a narrower width compared to the actual app. As a result, the text within the app would occupy less space in the preview, leading to misalignment issues when compared to the development window. This misalignment created challenges in achieving correct alignment, causing delays in the app development process. Aligning elements accurately became difficult and time-consuming due to the inconsistencies between the preview and development environments.
Limited Variable Assignment	In Power Apps, there is a limitation that prohibits the use of behavior functions in non-behavior attributes. These functions encompass actions such as setting variables or aggregating items together. This restriction poses significant challenges when implementing functionality like the preference review table in the students' app, as it requires numerous calls to different data sources and performing logic on the data within a non-behavior component. Consequently, this restriction leads to excessive data access to various columns of the same row, as it is not possible to save the intermediate results. The inability to abstract and encapsulate code logic becomes a hurdle in maintaining code readability. Although one workaround is to utilise components with visibility turned off, this solution is not ideal and can only be employed in limited scenarios where ForAll loops are not used. Furthermore, the use of ForAll loops imposes additional limitations on the utilisation of certain behavior functions, resulting in multiple unnecessary lookups of the same data source row. These limitations compound the challenges faced during development, further hindering efficiency and code optimisation.
Strange Typing	One of the drawbacks of Power Apps lies in its peculiar and inconsistent type system. Working with data types in Power Apps can sometimes be frustrating. The platform's type system lacks consistency and can behave unexpectedly in certain situations, leading to confusion and difficulties in data manipulation. One particular issue arises when dealing with conversions and comparisons between different data types. Power Apps may exhibit inconsistent behavior when attempting to convert or compare values of different types, resulting in unexpected errors or incorrect outcomes. This inconsistency can make it challenging to ensure reliable and accurate data processing within the application.

Unhelpful Lookup Columns	In Power Apps, when looking up rows in a List that contain lookup columns referencing items from another list, only limited information is available from that column, such as the ID and a few basic columns from the referenced list. This limitation makes the lookup columns less helpful, as additional lookups are often required to retrieve the desired information from the referenced items.
Poor Complexity Handling	Power Apps faced limitations when dealing with complex logic applied to datasource items. It struggled to enforce restrictions on student preferences due to difficulties in handling intricate logic and multiple datasource accesses. Improving code optimality was challenging, as Power Apps would enter an infinite loading state, disrupting linting and updates. These limitations impeded functionality and hindered the development process many times.
Lack of Features in Lists	Microsoft Lists, as Microsoft's new interface for SharePoint lists, offers a visually appealing UI. However, it lacks important features, such as batch updating of column values for multiple records or duplicating multiple records for testing purposes. This limited functionality significantly affected the efficiency of testing, making it a slow and tedious process.
Poor Export and Import Options of Lists	In Microsoft Lists, the available column export option for lookups and people columns is basic text columns containing only the item's name. This limitation resulted in the supervisor and student items lacking useful additional information when exported. To overcome this problem, additional columns had to be added to many lists to export people's emails as a reliable way to match items across different lists. Lists can be created by importing an excel file; however, there is no way to import data into an existing List and there is no way to import new items from a file even if they have the same schema forcing the use of repetitive one at a time copy pasting of records for testing.
Poor List Transferability	In Microsoft Lists, the absence of a direct method to copy lists between SharePoint sites poses a challenge. While it is possible to create lists on other sites based on existing ones, not all sites are available for selection, including personal sites where the original lists were created. An alternative option is to create list templates, export them, and upload them onto other sites for use. However, this process requires administrative privileges, making it a cumbersome and time-consuming procedure.
Lots of Annoying Nuances	Despite having a significant coding experience, it was observed that many implementations of functionality in Power Apps often required multiple attempts before they worked correctly. This recurring issue stemmed from the counterintuitive nuances of certain functions, making them less suitable for implementing more advanced requirements. Consequently, many challenges were faced, and multiple iterations were needed to achieve the desired results. The need for repeated implementation and fine-tuning underscored the complexity and intricacies involved in leveraging these functions effectively for more advanced needs.

Numerous Small Bugs	Throughout the development process, the progress was frequently hampered by various small bugs in Power Apps. These bugs manifested in different ways, such as component properties not unlocking or combo boxes randomly switching between being searchable or not. These issues may seem trivial at first glance, but they had a significant impact on the overall development timeline. Encountering bugs like these forced developers to invest additional time and effort in identifying and resolving them. In some cases, the only solution was to refresh the app, which interrupted the workflow and further prolonged the development process. The need for frequent refreshes to address these issues became an unwelcome disruption, disrupting the momentum and causing frustration.

Table 6.2: Cons of Power Apps in Combination with Microsoft Lists

The overall code quality within the apps can be described as sub optimal due to the inherent limitations of the framework when it comes to implementing complex logic. The Power Apps platform, while offering a low-code development environment, struggles to handle intricate logic and sophisticated programming requirements effectively. This limitation hampers the ability to write highly efficient and optimised code.

The constraints imposed by Power Apps result in workarounds and compromises when implementing complex functionality, often leading to convoluted and less maintainable code structures. Due to the platform's limited capabilities, it was challenging to employ best coding practices and adhere to standard software engineering principles. Consequently, the code suffers from reduced readability, maintainability, and extensibility.

Moreover, the absence of advanced debugging tools and comprehensive error handling mechanisms within the Power Apps framework further compounds the challenges of ensuring robust code quality. Identifying and resolving issues or bugs becomes more time-consuming and prone to error due to the platform's limitations in providing detailed debugging information.

PowerApps proved to be less suitable for this project considering the high complexity of required features. The platform's limitations hindered its effective handling of intricate functionalities, resulting in a significantly prolonged development process. As a consequence, valuable time that could have been dedicated to polishing the system and implementing planned features was instead consumed by overcoming PowerApps' constraints.

One such example is the inability to implement automated emails or seamless API integration between the interfacing code and lists. The later feature was initially planned to streamline processes and save time exporting lists. However, due to PowerApps' limitations, they were unable to be implemented within the given time frame.

The complexity of the project's requirements necessitated a more robust and flexible development platform that could accommodate the desired functionality seamlessly. While PowerApps offers a low-code environment that excels at rapid prototyping and simple applications, it falls short when confronted with intricate and demanding projects. The development software generally felt unfinished and meant it was often unclear if there was an error on the developers end or Power Apps. The poor permissions system of lists took further time away from the development process and does not feel like something that should have gone unchanged for so long.

To optimise future development efforts, it would be prudent to consider alternative platforms or technologies that align better with the project's complexity and feature requirements. Exploring more robust frameworks or custom development solutions could provide greater flexibility and efficiency, allowing for smoother implementation of advanced features and reducing the overall development timeline.

6.4.2 Interfacing Code

Table 6.3 has the evaluation of the code based of various aspects of it. The code would have benefited from more time to polish and improve it however in order to meet the deadline with the

Pros	Description
Drag and Drop Components	Power Apps made for quick and easy development of simple components without extensive coding. This was especially helpful due to lack of frontend experience.
Automatic Code Formatting	The Power Apps code editor offers a convenient button that enables instant code formatting. This functionality greatly simplifies the task of enhancing code readability and facilitates easier code editing. This feature proves to be particularly valuable, especially when considering the limited size of the code editor.
No-Code/Low-Code Development	Power Apps offers a no-code/low-code development environment, again this was helpful due to low frontend coding experience and made implementing simple functionality quick and easy.
Easy Hosting	Power Apps, being hosted by Microsoft, simplifies the deployment process. Once the apps are created and user permissions are granted, the hosting and distribution are taken care of. This streamlines the rollout of changes, making it quick and efficient. With Microsoft handling the hosting aspect, focus can be put on creating and improving the apps, knowing that the deployment process is well-managed.
Robust Power Apps and Lists Synchronisation	Power Apps, when connected with lists, effectively prevents patches from being applied if the Power App contains outdated data that it intends to patch. This functionality plays a crucial role in maintaining the integrity of the system and ensuring that data remains accurate and up to date. By preventing patches based on outdated information, Power Apps helps safeguard against potential data loss and maintains fairness in allocations or any other relevant processes. This ensures a safe and reliable system operation.
Quick Access To Datasources	Power Apps galleries offer convenient integration with Microsoft's lists, allowing for quick and straightforward retrieval and display of values. This functionality facilitated the creation of messengers and other components with simple logic in an efficient manner. By leveraging Power Apps galleries, it was easy to connect to lists and showcase the desired data.
Preview of Tables	In the Power Apps code editor, it was possible to preview the resulting table from a function, as long as the logic wasn't too complex. This feature made it easy to see if the code was being manipulated correctly at intermediate steps of long code snippets. By enabling code preview, Power Apps simplified the process of validating and debugging code, which contributed to making a smoother development experience.
Large Community Forums	Power Apps and Microsoft Lists had a relatively limited set of built-in functions and functionalities. As a result, developers often encountered similar various limitations and bugs during the development process. This led to a significant number of forum posts discussing workarounds for these platform constraints. The forums became a valuable resource for sharing insights, tips, and solutions to overcome the limitations and address the encountered bugs. These discussions helped to navigate through the challenges posed by the platforms.

Table 6.1: Pros of Power Apps in Combination with Microsoft Lists

delays that were faced this was not possible.

6.5 The Design

In hindsight, it may have been more prudent to limit the implementation of extensive functionality within PowerApps. The inclusion of features such as duplication and allocation requests introduced numerous complexities to the system, which proved challenging to implement. Additionally, a substantial amount of time was spent addressing edge cases that were unlikely to be encountered in practice.

By omitting these less essential functionalities to be handled externally, more time and resources could have been allocated to improving other aspects of the system. This would have allowed for a greater focus on refining existing features, enhancing user experience, and addressing any limitations or shortcomings identified during the development process.

The decision to prioritise certain functionalities over others is a common trade-off in software development projects. While it is important to aim for a comprehensive solution, it is equally vital to assess the potential impact and complexity of each feature. In this case, dedicating a significant portion of the development effort to handling duplication and allocation requests detracted from other areas that could have benefitted from more attention.

Readability and Maintainability	The code has been adequately formatted and follows consistent indentation, which enhances readability. Appropriate variable and function names have been used, making the code easier to understand. The code could benefit from additional comments or documentation to explain the purpose of certain functions or clarify complex logic.
Code Organisation	The code is organised into modules functions, which helps improve modularity and maintainability. The use of type aliases and record types improves code clarity and expressiveness.
Error Handling	The code includes error handling in some cases, such as using failwithf to raise exceptions when encountering certain conditions. However, there are areas where error handling could be improved. For example, error cases in CSV parsing or file loading are not explicitly handled, which may result in unhandled exceptions. This should be okay as there won't be errors if the setup guide is followed correctly. Also, in further work, changes to the way the CSV files are imported will be discussed that would render the additional code obsolete.
Code Efficiency	The code appears to be reasonably efficient in terms of execution time, as it processes data from CSV files and performs necessary operations. There is room for potential optimizations, but the code still executes in a satisfactory time.
Code Reusability	The code could benefit from better modularity and separation of concerns, allowing for improved reusability of individual components. However, the nature of the code means that it is unlikely much of it will be reused.
Adhering to Best Practices	The code follows some best practices, such as using type annotations, adhering to naming conventions, and utilizing library functions where appropriate. However, there are areas where certain best practices could be applied more consistently, such as handling nullability of values or using appropriate F# idioms for certain operations.
Testing	A good number of unit tests were used to test the main data manipulation functions. To add to this, during the development process, F# interactive and executing trial allocations were used with different difficult inputs to make sure the code was able to handle them. Ideally, more unit tests would be implemented to ensure tests are still passed after changes and updates to the code .

Table 6.3: Code Evaluation

Chapter 7

Testing

7.1 PowerApps and Lists Testing

Tables 7.1 7.1 7.1 shows the user requirements that the different apps had to meet, the test cases for those requirements and whether or not the tests where passed or failed. The tests were carried out on a test account that only had the permissions that the users would have for the different tests. This was to make sure that the permissions were set up correctly. Unfortunately, the built in Power Apps test suit did not have the necessary functionally to effectively make tests for these requirements.

Actor	**Requirement**	**Test Case**	**Pass/Fail**
Student	View, filter and bookmark projects with access to all necessary information	Use all different filters at the same time and bookmark multiple projects, making sure all information is accessible.	Pass
Student	Ability to submit preferences	Submission of both initial and updated preferences, making sure gaps and duplicate preferences are caught and that the student is placed in the project suitability rankings.	Pass
Student	Submit self-proposals for projects	Creation of self proposal, Attempt creation of proposal with same name to the same supervisor and see error.	Pass
Student	Consent to supervisors allocating them projects	View the project being offered. Approve the request. Unapprove the request. Reject the request.	Pass
Student	Assert whether they are doing DoC project allocation instead	Use the select DoC toggle on the home page and refresh to make sure change has taken place.	Pass

Table 7.1: Students Requirements

Actor	Requirement	Test Case	Pass/Fail
Supervisor	Create a project with all necessary information	Create projects assuring empty boxes error. Attempt to create a project with the same name and see error.	Pass
Supervisor	Manage existing projects by deleting them, editing them and ranking students	Delete project. Update project form fields and submit successfully. Move students between every suitability ranking on with all rankings including students and ensure no students are lost and all end in the correct column	Pass
Supervisor	Request allocation of a project to a specific student and request duplication of projects	Create allocation request to a student successfully. Attempt creation of the same allocation request and get error. Create duplication request successfully. Attempt a duplication request for the same project and see error	Pass
Supervisor	Manage allocation and duplication requests	Delete both types of requests. Update the number of copies on a duplication request	Pass
Supervisor	Respond to students self-proposing projects with them as the supervisor	Access attachments, approve, unapprove and reject a project	Pass

Table 7.2: Supervisors Requirements

Actor	Requirement	Test Case	Pass/Fail
Coordinator	View all student preferences	Enter students preferences section and select a student will submitted preferences. Preferences and review are shown	Pass
Coordinator	View all student self-proposals, give feedback, and approve/reject them	Veiw self proposal and access attachments successfully. Approve request, unapprove request, reject request. Message student with messenger and check it is received	Pass
Coordinator	View all existing projects and the interest on them	Access all information for a project in project list and view number of students with it in their top 3 and any preference rank at all	Pass
Coordinator	View all allocation/duplication requests from staff, give feedback, and accept/decline	Use messengers to send messages with them being successfully received. Approve request, unapprove request, reject request.	Pass

Table 7.3: Coordinator Requirements

Along with these test all of the apps additional functionalities were tested by simply observing if components behaved in the correct way based on different values in the Microsoft lists.

Table 7.1 presents the results of some resiliency tests conducted to validate the system's robustness before its implementation. Additional tests were performed to simulate concurrent access to the same data by multiple actors, ensuring that patches were appropriately blocked when necessary and no disruptions occurred in the operation of the applications.

These tests were crucial in assessing the system's ability to handle concurrent data access scenarios without compromising data integrity or causing any unintended malfunctions. By subjecting the system to realistic usage scenarios, it was confirmed that appropriate measures were in place to prevent conflicts and maintain the stability of the applications.

Actor	**Requirement**	**Test Case**	**Pass/Fail**
System	Handle a project being deleted without breaking any pages	Delete project proposal in backend and see that duplication requests and allocation request for it are also deleted	Pass
System	Handle a student being deleted without breaking any pages	Delete a student from the student info list and check no pages break	Pass
System	Handle a student changing streams	Change students stream in student info list and delete their preferences. See that their project preferences options have changed	Pass
System	Handle students joining the system after the initial opening	Add student to student info list, app access and permissions group and see that they can use the app as normal	Pass
System	Can handle preference submission including project at the same time as supervisor is ranking students for it	Submit a preference set that includes a project and quickly rank an unranked student for the project to cause a synchronisation error. Retry the action and when it goes through ensure that no students are lost from the suitability columns	Pass

Table 7.4: Resilience Requirements

The Tables show all necessary functionality that system users need have been implemented.

Tests were performed in which it was attempted with the different permission levels to access SharePoint Lists directly using their URLs. All attempts were met with access denied on the student and supervisor permission levels as desired. It is important that no students can view the backend lists and gain information about other students' preferences to game the system. Supervisors are also blocked from being able to gain access to the lists as they are not expected to have an understanding of how the system works and could end up breaking things.

7.1.1 The Interfacing Code

The interfacing code underwent rigorous testing using a combination of unit testing, F# interactive, and trial allocations to ensure its functionality and reliability.

Unit Testing

To validate the data manipulation functions, a set of 20 unit tests was created. These tests covered the main functions in the data manipulation module and were designed to verify the correctness

of the code. It is worth noting that all 20 unit tests successfully passed.

These unit tests can be executed by following the instructions provided in the User Guide (see Section 10). The tests helped identify and rectify several bugs, such as incorrectly identifying students who had no changes made to their preferences and the failure to update supervisors when removing redundant projects. Due to this, under certain conditions there was the creation of excessive pseudo students, which was also addressed thanks to the unit tests.

F# Interactive and Trial Allocations

During the development process, F# Interactive was extensively utilised for testing small functions with a variety of input scenarios. This interactive environment facilitated rapid experimentation and debugging, ensuring the correct behavior of individual code components.

Additionally, multiple trial allocations were performed with intermediate program printing to verify the code's ability to handle diverse CSV data. These trial allocations were instrumental in validating the generation of the cost matrix, ensuring that pseudo students were assigned the appropriate costs for specific projects, and verifying the impact of hyperparameters on the allocation process.

Future Testing Considerations

Due to time constraints, end-to-end tests for repeated execution on code changes and more extensive unit testing could not be implemented at this stage. However, it is recommended to incorporate these testing strategies in future iterations. End-to-end tests would provide valuable insights into the system's behavior over repeated executions, while enhanced unit testing would further validate the code's correctness and robustness.

Chapter 8

Deployment

8.1 Lists Set Up

To set up the necessary lists for the system, follow the steps below. Please note that as the SharePoint site used for the project will no longer be available, a new site must be set up to host the system.

There are two methods for obtaining the required lists:

Method 1

1. Use the "New list" button in Microsoft Lists and select the existing list templates provided.
2. From my personal site, Justice Konec Dominic, select and create the following lists:
    - ProjectProposals
    - DuplicationRequests
    - AllocationRequests
    - SelfProposals
    - StudentPrefs
    - Messages

Ensure that the newly created lists have the same names as the ones being copied from. It's important to note that the SharePoint site may need to be enabled for list access by an administrator account.

Method 2

1. Use the list templates available in the project repository.
2. Upload these templates onto the new site and enable them for usage with an administrator account.
3. When creating a list on the new site, the list templates should be available for selection. Create lists with the same names as the templates.

After creating the six lists, proceed to create a list of students using the app by importing data from Excel. The Excel list should include the following columns:

- Title: This column should contain the information by which the FYP Coordinator wants to search the students in the app (e.g., Full name).
- Email: Import the long-form student emails as a single-line text column with the name "Email."

- ERegyr: Import the student's stream code as a single-line text column with the name "ERegyr."
- Taking DOC Project: Import a column indicating whether the student is entering the DoC allocation instead of the EEE allocation as a choice column with the name "Taking DOC Project."

8.2 Power Apps Setup

Now that the lists are created, upload the Power Apps from the project repository to the new site. Access each app and update the data source connections, replacing the connections to SharePoint with connections to the new SharePoint lists. Since the lists have the same names, this step should not result in any errors. However, it is recommended to check the app doctor for any errors and correct them by providing the correct names. Please note that you may encounter delegation warnings in the app doctor, but these warnings are expected and explained in the User Guide.

8.3 Permissions

To set up the correct permissions on the new site, follow the method outlined in the comment by stevegeall [78]. However, instead of changing permissions for the "site name visitors," create three new groups: Students, Supervisors, and Coordinators. Alongside editing the read and contribute permission levels, create a "contribute and delete" permission level that includes the ability to delete list items.

For the following lists:

- ProjectProposals
- DuplicationRequests
- AllocationRequests
- StudentPrefs

Grant the Students group contribute permissions, as mentioned in the guide.
For the following lists:

- SelfProposals
- Messages

Grant the Students group contribute and delete permissions.
For the Supervisors group, grant contribute permissions on the following lists:

- ProjectProposals
- SelfProposals
- StudentPrefs

Grant the Supervisors group contribute and delete permissions on the following lists:

- DuplicationRequests
- AllocationRequests
- Messages

Lastly, give the Coordinators group full control over the entire site.

You can test the permissions by adding a test account to each group and granting them app access. Ensure that they can use all the features. For the Students and Supervisors groups, attempt to access any of the lists using a URL, and confirm that they receive an "access denied" response.

8.4 Adding Users

To add users to the system, follow these steps:

- Add students to the "Students" permission group, ensuring to not to email them all simultaneously.
- Add supervisors to the "Supervisors" group, ensuring to not to email them all simultaneously.
- Add coordinators to the "Coordinators" group, ensuring to not to email them all simultaneously.
- Share each Power App with its respective users, taking care once again to avoid emailing all users simultaneously.

8.5 Conclusion

The deployment process is unfortunately very complex and specific. This is due to a lack of Power Apps and SharePoint UI functionality. The process should be improved by writing C# code that should automate the process; for example, here [79] is how SharePoint permission levels can be created and changed programatically. Once on the correct site, the deployment for future years becomes a lot quicker and easier, simply adding students and staff to the correct permission groups and giving them access to the apps. This means a programmatic migration method may take more time to implement than it saves, though perhaps the lists will have to be relocated in the future to another site and it would again be useful.

Chapter 9

Conclusions

In this section, I will delve into the evaluation of the project's overall success, examining the effectiveness of specific design decisions. We explore the positive outcomes resulting from these choices and delve into the challenges encountered during different phases of the project and how they were overcome. Finally, the section will identify potential areas for future work and enhancements that can further improve the system's capabilities and performance.

9.0.1 Requirement Capture

The process of requirement capture and gaining a deep understanding of the project's intricacies played a pivotal role in ensuring its success. Extensive efforts were made to comprehend not only the typical behaviors of students and supervisors during the allocation process but also the nuances of the project itself. However, striking a balance between thorough discussions and timely implementation posed a challenge. It was essential to invest sufficient time in exploring various aspects to ensure the system met its requirements effectively.

9.0.2 The Frontend System

The decision to develop three distinct Power Apps, each containing carefully crafted pages, was thoughtfully considered. These apps successfully addressed the necessary requirements, providing a clear and efficient user experience. However, in retrospect, the choice to use Power Apps may not have been ideal. Although research indicated its effectiveness, it was underestimated how complex and interdependent the logic within the system would be, which Power Apps struggled to accommodate. While Power Apps met initial expectations with its user-friendly drag-and-drop interface and quick development of initial interfaces, it proved inadequate for handling the complexity of the allocation system. Fine-tuning interfaces became challenging, with the drag-and-drop functionality feeling clunky and imprecise, and the code controls proving inefficient to use. It is evident why Power Apps is highly regarded for prototyping, where perfect interfaces and intricate logic are not essential. The issues encountered during Power Apps development were not immediately apparent, as significant effort was invested in building various app components before encountering the need for more complex logic. By then, it was too late to switch to another platform. It was a learning experience to realise that a Microsoft-developed platform might not be as powerful and polished as older, well-established languages. Overcoming these difficulties involved drawing upon my programming experience and resilience developed throughout the challenging aspects of the EIE course. This aspect of the project further strengthened my ability to persevere and remain focused while working on solo projects over an extended period.

Another challenge during Power Apps development was to prioritise user-friendliness and accessibility, as these aspects had received limited emphasis during my time at Imperial. Leveraging my own experience with dyslexia and conducting research on creating more accessible webpages, I believe I have implemented a set of apps that surpass the previous implementation significantly in terms of accessibility. The apps effectively gather the necessary information from supervisors and students and communicate well-formatted and relevant information to the respective parties.

Considerable time and effort were dedicated to permissions and app migrations within SharePoint. In hindsight, these challenges could have been mitigated by better understanding the limitations and intricacies from the project's outset and setting up a dedicated SharePoint site with robust permissions early in the development process. Unfortunately, these limitations did not become apparent during the research phase.

9.0.3 The Allocation Code

The discussion on optimisation algorithms and the subsequent investigation into the suitability of the Hungarian Algorithm were conducted meticulously to ensure the selection of a suitable allocation algorithm for the system. The design of necessary adaptations for the Hungarian Algorithm required careful thought and consideration of various system aspects.

The development of the interfacing code between the Power Apps and the Hungarian algorithm was the most enjoyable part of the project. However, due to time constraints, there was limited opportunity to optimise and refine the code as desired. Nonetheless, it presented an engaging challenge that drew upon my understanding of the allocation process to integrate the different pieces of information. F# proved to be a powerful tool for data manipulation, and my experience in the HLP module significantly contributed to successfully implementing the code. However, given the speed at which I had to develop it, some of the coding followed more object-oriented programming (OOP) style, which was not highly efficient within the F# paradigm.

The work done to optimise the allocation code yielded significant success. The choice of algorithm and the interfacing code allowed for fully optimal allocations to be performed in just a few seconds, depending on the chosen hyperparameters. This represents a substantial improvement over the previous code. However, limitations arose from the inefficient data retrieval process from Power Apps, requiring manual exporting and copy-pasting of lists. This limitation will be addressed in the future work.

Finally, some of the biggest challenges came from the report writing which as my first long solo report as a dyslexic I found extremely difficult. I felt like it detracted from the time I could spend on my deliverable and the overall quality of them.

Overall, the project exhibits strong and successful aspects but falls short in certain areas. While theoretically deployable for use, further work would enhance its effectiveness. The project's scope and magnitude, particularly considering the unforeseen challenges with Power Apps, exceeded initial expectations. Additional resources in the form of more team members or extended time would have been beneficial to address these complexities effectively.

9.0.4 Future Work

The system would greatly benefit from additional improvements and enhancements, which could be focused on the following areas:

- The Power Apps can be further refined to enhance their visual appeal and optimise their performance. This could involve fine-tuning the user interface, improving the layout and design, and implementing more intuitive navigation and interaction features. Certain functions of the apps would benefit greatly from further filtering features that were not implementable within the time limit. The apps also have space for more time relevant information to be added to make sure users perform as desired.

- Currently, the system has a limitation in terms of handling a larger number of students, which is around 2000, due to delegation issues within Power Apps. To overcome this limitation, it is recommended to consult the app doctor to identify the specific delegation issues. Workarounds can be implemented based on tactics shared in resources such as the forum post cited [80].

- To simplify the deployment procedure, it would be beneficial to develop programmatic code for automating the migration and setup of lists and apps. This would streamline the precise and intricate deployment process, reducing the need for manual intervention and potential errors.

- In the interfacing code, there is an opportunity to improve efficiency by utilising more optimised F# code for reading CSV files. Alternatively, direct calls to SharePoint can be incorporated to fetch list values, taking advantage of the fact that F# is based on .NET.
- End to end tests should be implemented on the interfacing code to ensure that after any changes the code still performs correctly and allocations are happening as they should be.
- Consider transforming the Hungarian algorithm implementation into a DLL (dynamic-link library) to enable the removal of file transfer between programs or explore the possibility of creating an F# port of the algorithm. This would enhance interoperability and simplify integration with other components of the system.
- Currently, the allocation code directory only supports the Windows operating system. To broaden its compatibility, it would be valuable to create run files for other operating systems, ensuring that the system can be deployed and utilised across various platforms.
- The system is not currently equipped to deal with the masters streams but this should not be too hard to update just consult the app doctor in the apps.
- Automated emails were not implementable in the time scale given but using power automate these could be added to keep students and supervisors up to date.

By addressing these areas of improvement, the system can be further enhanced, offering a more visually appealing, scalable, and efficient solution for allocation processes.

Chapter 10

User Guide

This guide assumes that the system has been deployed successfully and is functioning.

10.1 The Coordinator App

Project Management

The project list can be viewed and filtered with the competitiveness displayed and for each project the suitability rankings can be viewed and disabled if it is thought that supervisors are using the system dubiously. This makes all students in the low medium and high suitability sections count as the same suitability.

Requests

Duplication requests, allocation requests and self proposals can be handled in there respective pages with feedback being provided to participants in the messenger. The number in the top right of the galleries show how many unread messages there are for that item and to read the messages just click on them.

Preferences

The student list has all the students on the student info list. Each students preferences can be reviewed and a messenger with the same unread system as requests can be used to communicate preference suitability to students.

10.2 Student Updates

Stream Change

If a student changes streams during the course of the allocation then their stream in the student info list should be updated to mirror this change.

New Student

If a new student appears in the allocation then it should be made sure they are added to the correct permission group and given an entry on the student info list before they are given access to the app.

Remove Student

If a student is removed from the system then they should have there entries in the student prefs list and student info list deleted and any of there allocation requests or self proposals deleted also.

10.3 System Running Advice

Be very careful directly modifying backend lists when users could be changing the item you are editing. There patches will be overwritten when you save if they make them during the time you are editing the same record.

10.4 Running Allocation

Export the following lists as CSVs:

- AllocationRequests
- DuplicationRequests
- ProjectProposal
- SelfProposals
- StudentInfo
- StudentPrefs

Add these CSVs to the second InterfacingCode directories csvs file along with a supervisor loading CSV that needs to contain all the supervisors in the system and have the columns:

- Supervisor Email: This is the long form email of the supervisor
- Max Projects: This is the maximum number of allocated project copies the supervisor can handle
- Cost Weight: This is a hyperparameter that modifies how much it costs to allocate students to projects of this supervisor and can be edited to aid in supervisor loading.

With these in place refer to the FYP Allocation Code repository. For information on hyperparams go to sections 4.10.5 and 5.3.3.

Bibliography

[1] W. W. A. Initiative (WAI). "Avoid too much content," Web Accessibility Initiative (WAI). (Apr. 28, 2023), [Online]. Available: https://www.w3.org/WAI/WCAG2/supplemental/patterns/o5p03-manageable-quantity/.

[2] R. Jutaviriya. "A perfect project selection and allocation system for imperial college." (2021).

[3] H. R. Lourenço, O. Martin, and T. Stützle, "A beginner's introduction to iterated local search," in *Proceedings of MIC*, vol. 2, 2001, pp. 1–6.

[4] S. J. Russell, *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[5] J. Pearl, *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., 1984.

[6] "Hyperparameter," Techopedia. (Aug. 8, 2022), [Online]. Available: https://www.techopedia.com/definition/34625/hyperparameter-ml-hyperparameter.

[7] "Build apps for your business | microsoft power apps." (), [Online]. Available: https://powerapps.microsoft.com/en-gb/landing/developer-plan/.

[8] "Introduction to lists - microsoft support." (), [Online]. Available: https://support.microsoft.com/en-us/office/introduction-to-lists-0a1c3ace-def0-44af-b225-cfa8d92c52d7.

[9] Z. Paracha. "Top 9 reasons to use power apps," Encore Business Solutions. (Oct. 26, 2020), [Online]. Available: https://www.encorebusiness.com/blog/top-9-reasons-to-use-power-apps/.

[10] "PowerApps vs react," TrustRadius. (), [Online]. Available: https://www.trustradius.com/compare-products/powerapps-vs-react.

[11] "React." (), [Online]. Available: https://react.dev/.

[12] "JavaScript | MDN." (Apr. 5, 2023), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[13] J. Compton. "Business benefits of microsoft power apps," Pragmatiq. (Oct. 29, 2020), [Online]. Available: https://www.pragmatiq.co.uk/business-benefits-of-microsoft-power-apps/.

[14] "React.js security guide: Threats, vulnerabilities, and ways to fix them," Relevant Software. (Dec. 6, 2022), [Online]. Available: https://relevant.software/blog/react-js-security-guide/.

[15] "SQL injection." (), [Online]. Available: https://www.w3schools.com/sql/sql_injection.asp.

[16] joel-lindstrom. "Create a canvas app that can trigger a power automate flow (contains video) - power apps." (Dec. 16, 2022), [Online]. Available: https://learn.microsoft.com/en-us/power-apps/maker/canvas-apps/how-to/trigger-flow.

[17] ".NET | build. test. deploy.," Microsoft. (), [Online]. Available: https://dotnet.microsoft.com/en-us/.

[18] "Power automate | microsoft power platform." (), [Online]. Available: https://powerautomate.microsoft.com/en-gb/.

[19] MSFTMan. "Create flows for popular email scenarios in power automate - power automate." (Feb. 9, 2023), [Online]. Available: https://learn.microsoft.com/en-us/power-automate/email-top-scenarios.

[20] W. W. A. Initiative (WAI). "Provide reminders," Web Accessibility Initiative (WAI). (Apr. 11, 2023), [Online]. Available: https://www.w3.org/WAI/WCAG2/supplemental/patterns/o7p07-reminders/.

[21] "Why millions of developers use javascript for frontend web development?" Section: Web Development. (Jul. 28, 2021), [Online]. Available: https://itechnolabs.ca/millions-of-developers-use-javascript-for-frontend/.

[22] "Why JavaScript is the programming language of the future," freeCodeCamp.org. (Apr. 4, 2020), [Online]. Available: https://www.freecodecamp.org/news/future-of-javascript/.

[23] "Accessibility – react." (), [Online]. Available: https://legacy.reactjs.org/docs/accessibility.html.

[24] "Understanding what is MS power apps and its benefits | updated," NetCom Learning. (), [Online]. Available: https://www.netcomlearning.com/blogs/281/understanding-what-is-microsoft-power-apps-and-its-benefits.html.

[25] "How long does it take to learn JavaScript? (with 5 ways)," Indeed Career Guide. (), [Online]. Available: https://ca.indeed.com/career-advice/career-development/how-long-does-it-take-to-learn-javascript.

[26] "F# software foundation." (), [Online]. Available: https://fsharp.org/.

[27] "Why use f#? | f# for fun and profit." (), [Online]. Available: https://fsharpforfunandprofit.com/why-use-fsharp/.

[28] cartermp. "Pattern matching - f#." (Nov. 5, 2021), [Online]. Available: https://learn.microsoft.com/en-us/dotnet/fsharp/language-reference/pattern-matching.

[29] "Exploring data with f# type providers," TECHCOMMUNITY.MICROSOFT.COM. Section: Educator Developer Blog. (Mar. 21, 2019), [Online]. Available: https://techcommunity.microsoft.com/t5/educator-developer-blog/exploring-data-with-f-type-providers/ba-p/378900.

[30] "Other interesting ways of using f# at work · f# for fun and profit." (), [Online]. Available: https://swlaschin.gitbooks.io/fsharpforfunandprofit/content/posts/low-risk-ways-to-use-fsharp-at-work-5.html.

[31] Deland-Han. "Dynamic link library (DLL) - windows client." (Apr. 12, 2022), [Online]. Available: https://learn.microsoft.com/en-us/troubleshoot/windows-client/deployment/dynamic-link-library.

[32] "C++ introduction." (), [Online]. Available: https://www.w3schools.com/cpp/cpp_intro.asp.

[33] "Repository search results for hungarian algorithms in c++· GitHub." (), [Online]. Available: https://github.com/search?q=hungarian+algorithm+language%5C%3AC%5C%2B%5C%2B&type=repositories&l=C%5C%2B%5C%2B.

[34] TylerMSFT. "Walkthrough: Create and use your own dynamic link library (c++)." (Dec. 10, 2021), [Online]. Available: https://learn.microsoft.com/en-us/cpp/build/walkthrough-creating-and-using-a-dynamic-link-library-cpp.

[35] "Seamless interoperation with .NET libraries | f# for fun and profit." (), [Online]. Available: https://fsharpforfunandprofit.com/posts/completeness-seamless-dotnet-interop/.

[36] dotnet, *Integrating PowerApps with .NET web APIs*, Apr. 1, 2021. [Online]. Available: https://www.youtube.com/watch?v=VLqdRpwniwk.

[37] "TIOBE index," TIOBE. (), [Online]. Available: https://www.tiobe.com/tiobe-index/.

[38] "Tomcl - overview," GitHub. (), [Online]. Available: https://github.com/tomcl.

[39] "Welcome to python.org," Python.org. (May 5, 2023), [Online]. Available: https://www.python.org/.

[40] "PyPI · the python package index," PyPI. (), [Online]. Available: https://pypi.org/.

[41] "Repository search results for hungarian algorithms in python· GitHub." (), [Online]. Available: https://github.com/search?q=hungarian+algorithm+language%5C%3APython&type=repositories&l=Python.

[42] "Pandas - python data analysis library." (), [Online]. Available: https://pandas.pydata.org/.

[43] "Pandas documentation — pandas 2.0.1 documentation." (), [Online]. Available: https://pandas.pydata.org/docs/.

[44] "Matplotlib — visualization with python." (), [Online]. Available: https://matplotlib.org/.

[45] M. Waskom, "Seaborn: Statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, Apr. 6, 2021, ISSN: 2475-9066. DOI: 10.21105/joss.03021. [Online]. Available: https://joss.theoj.org/papers/10.21105/joss.03021.

[46] S. P. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004, p. 127, ISBN: 978-0-521-83378-3. [Online]. Available: https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf.

[47] A. Agnetis, "Introduction to matching problems,"

[48] Y. Okumura, "A one-sided many-to-many matching problem," *Journal of Mathematical Economics*, vol. 72, pp. 104–111, 2017, ISSN: 0304-4068. DOI: https://doi.org/10.1016/j.jmateco.2017.07.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304406816302865.

[49] D. J. Abraham, *Algorithmics of two-sided matching problems*. University of Glasgow (United Kingdom), 2003.

[50] R. E. Burkard, U. Derigs, R. E. Burkard, and U. Derigs, "The linear sum assignment problem," *Assignment and Matching Problems: Solution Methods with FORTRAN-Programs*, pp. 1–15, 1980.

[51] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[52] Z. Takhirov. "Hungarian algorithm," z-Scale. (Jul. 19, 2017), [Online]. Available: http://zafar.cc/2017/7/19/hungarian-algorithm/.

[53] "Assignment problem and hungarian algorithm." (), [Online]. Available: https://www.topcoder.com/thrive/articles/Assignment%5C%20Problem%5C%20and%5C%20Hungarian%5C%20Algorithm.

[54] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems*. USA: Society for Industrial and Applied Mathematics, 2009, ch. 6, pp. 172–191, ISBN: 0898716632.

[55] D. J. Abraham, R. W. Irving, and D. F. Manlove, "The student-project allocation problem," in *Algorithms and Computation*, T. Ibaraki, N. Katoh, and H. Ono, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 474–484, ISBN: 978-3-540-24587-2.

[56] S. Olaosebikan and D. Manlove, "Super-stability in the student-project allocation problem with ties," in *Combinatorial Optimization and Applications: 12th International Conference, COCOA 2018, Atlanta, GA, USA, December 15-17, 2018, Proceedings 12*, Springer, 2018, pp. 357–371.

[57] J. Hartmanis, "Computers and intractability: A guide to the theory of np-completeness (michael r. garey and david s. johnson)," *Siam Review*, vol. 24, no. 1, p. 90, 1982.

[58] E. A. "(39) creating a messenger in power apps | LinkedIn." (), [Online]. Available: https://www.linkedin.com/pulse/creating-messenger-power-apps-yelyzaveta-akinfiieva/ (visited on 06/20/2023).

[59] Y. Yang, *HungarianAssignment*, original-date: 2016-01-06T13:11:59Z, Apr. 25, 2016. [Online]. Available: https://github.com/yangyutu/HungarianAssignment.

[60] J. Payor, *Jamespayor/weighted-bipartite-perfect-matching*, original-date: 2017-12-13T02:12:26Z, Apr. 6, 2023. [Online]. Available: https://github.com/jamespayor/weighted-bipartite-perfect-matching.

[61] *Assignment problem*, in *Wikipedia*, Page Version ID: 1144928595, Mar. 16, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Assignment_problem&oldid=1144928595#Unbalanced_assignment.

[62] L. Ramshaw and R. E. Tarjan, "On minimum-cost assignments in unbalanced bipartite graphs," *HP Labs, Palo Alto, CA, USA, Tech. Rep. HPL-2012-40R1*, vol. 20, pp. 4–6, 2012. [Online]. Available: https://www.hpl.hp.com/techreports/2012/HPL-2012-40R1.pdf.

[63] "Mcximing/hungarian-algorithm-cpp: A c++ wrapper for a hungarian algorithm implementation." (), [Online]. Available: https://github.com/mcximing/hungarian-algorithm-cpp.

[64] "Github," Build software better, together. (), [Online]. Available: https://github.com.

[65] M. Buehren, *Functions for the rectangular assignment problem*, MATLAB Central File Exchange, Retrieved April 28, 2023, 2014. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/6543-functions-for-the-rectangular-assignment-problem.

[66] cartermp. "Collection types - f#." (Mar. 25, 2023), [Online]. Available: https://learn.microsoft.com/en-us/dotnet/fsharp/language-reference/fsharp-collection-types (visited on 06/04/2023).

[67] S. Liang, *The Java native interface: programmer's guide and specification*. Addison-Wesley Professional, 1999.

[68] Reza Dorrani, *Power apps gallery design tutorial | gallery UI styles*, Jan. 9, 2023. [Online]. Available: https://www.youtube.com/watch?v=Jj4ZYztnUQ0.

[69] Reza Dorrani, *Multi select filters in power apps with combo box, checkboxes & no delegation*, Apr. 25, 2022. [Online]. Available: https://www.youtube.com/watch?v=Wr42PeW6_wU.

[70] gregli-msft. "Filter, search, and LookUp functions in power apps (contains video) - power platform." (Feb. 23, 2023), [Online]. Available: https://learn.microsoft.com/en-us/power-platform/power-fx/reference/function-filter-lookup.

[71] Matthew Devaney. "SharePoint delegation cheat sheet for power apps." (Feb. 6, 2023), [Online]. Available: https://www.matthewdevaney.com/sharepoint-delegation-cheat-sheet-for-power-apps/.

[72] gregli-msft. "ForAll function in power apps - power platform." (Feb. 23, 2023), [Online]. Available: https://learn.microsoft.com/en-us/power-platform/power-fx/reference/function-forall.

[73] Matthew Devaney. "Power apps custom functions & reusable code." (Feb. 21, 2021), [Online]. Available: https://www.matthewdevaney.com/power-apps-custom-functions-reusable-code/.

[74] "Components - work around no data source access." (Sep. 20, 2019), [Online]. Available: https://powerusers.microsoft.com/t5/Building-Power-Apps/Components-work-around-no-data-source-access/td-p/365848.

[75] gregli-msft. "Collect, clear, and ClearCollect functions in power apps - power platform." (Feb. 23, 2023), [Online]. Available: https://learn.microsoft.com/en-us/power-platform/power-fx/reference/function-clear-collect-clearcollect.

[76] D. D. (S. MVP). "How to clear a collection inside ForAll in PowerApps canvas apps," Debajit's Power Apps & Dynamics 365 Blog. (Feb. 3, 2022), [Online]. Available: https://debajmecrm.com/how-to-clear-a-collection-inside-forall-in-powerapps-canvas-apps/.

[77] gregli-msft. "Remove and RemoveIf functions in power apps - power platform." (Jun. 8, 2023), [Online]. Available: https://learn.microsoft.com/en-us/power-platform/power-fx/reference/function-remove-removeif.

[78] https://powerusers.microsoft.com/t5/user/viewprofilepage/user-id/39272. "Prevent people from accessing SharePoint list that the PowerApp is connected to." Section: Building Power Apps. (Sep. 12, 2018), [Online]. Available: https://powerusers.microsoft.com/t5/Building-Power-Apps/Prevent-people-from-accessing-SharePoint-list-that-the-PowerApp/td-p/156033/page/2 (visited on 06/02/2023).

[79] S. Rajack. "Create permission level programmatically in SharePoint," SharePoint Diary. (May 9, 2013), [Online]. Available: https://www.sharepointdiary.com/2013/05/create-permission-level-programmatically-in-sharepoint-2010.html.

[80] https://powerusers.microsoft.com/t5/user/viewprofilepage/user-id/412285. "Delegation warning CountRows(filter(datasource, status="completed"))." Section: Building Power Apps. (Jul. 7, 2022), [Online]. Available: https://powerusers.microsoft.com/t5/Building-Power-Apps/Delegation-warning-CountRows-Filter-datasource-status-quot/td-p/1659202.

Appendix A

Appendix

A.1 Form Outlines

Below are the planed outlines for each form in the system:

A.1.1 Supervisor Forms

Home

- Link to current personal project list page
- Link to view requests for action and statuses of project allocation/duplication requests

Personal project list page
The page will contain a block for each of the supervisors projects, each block will include.

- The project title that will link to the proposal
- A status, Open or Allocated to <student name>
- A button to edit the proposal that links to the filled out proposal form
- A button to go to request project duplication form
- A button to go to form for requesting project allocation to a specific student
- The number of students waiting to be ranked
- The number of students already ranked
- A link to the student sign off page

There will also be a button that links to a new project proposal page

Student sign off page

- 5 buttons to filter the students, Unranked, Low Suitability, Medium Suitability, High Suitability, Unsuitable
- List of students that reacts to whichever rank is selected
- Box with info on selected student and options on how to rank them

Project proposal page

- Project title box
- Multiple selection box for each stream for whether they are applicable for the project
- Project description box
- Project focus drop down allowing multiple selections e.g. machine learning
- Required skills info box
- Desired skills info box
- Flexibility scale
- If interested instruction box

Action page

- List of supervisors requests and their statuses
- List of self proposals from students that have requested the supervisor as their supervisor
- Link to download the full proposal
- Button to accept the proposal or reject it

A.1.2 Student Forms

Home

- Link to preference page
- Link to project list
- Link to self proposals
- Button to identify as wanting to do a DoC project with explanation of process

Preference page
The page will consist of a project preference section:

- 5 default drop down boxes for preferences labelled
- An add preference button
- For each preference a button to make a preference equal to the one above it
- A value TBD next to each preference that informs how competitive the project is
- A section showing if the student has met the preference requirements and how to do so if not
- Comments that the students want to be taken in to account by the FYP Coordinator
- A response section from the FYP Coordinator

Project list page

- Search bar for projects
- Filtering options for supervisor, project focus, favorites, time released and stream
- List of projects with all information given in the supervisor proposal page
- Button on each project to favorite it
- Button on each project to email supervisor

Self proposal page

- List of projects supervisors have proposed to student
- Button to accept supervisor proposed project
- List of ongoing submitted proposals with approval status
- Button to add a proposal
- Button to attach proposal spec to a new proposal
- Drop down to assign a supervisor to a new proposal
- Button to delete a proposal

A.1.3 FYP Coordinator Forms

Admin home page

- Link to student list page
- Link to supervisor proposal list page
- Link to projects page

Student list page

- List of all student names
- Can be filtered by pending action, number of preferences given, allocation likely hood
- Each name is accompanied by a allocation likely hood value and number of projects given as preferences
- Each name will have a link to the students page

Student page

- List of student preferences
- Student allocation likely hood
- Student name and email hyperlink
- List of projects proposals student is involved in, supervisor proposed or self proposed
- Each project has the option to accept or decline approval

Supervisor list page

- List of all supervisor new project proposal titles and their supervisor
- Each proposal has a link to the full proposal
- Each proposal has a status accepted, declined or pending

Supervisor proposal page

- The full proposal
- If it is a proposal to a specific student then the status of their approval
- Button to approve or reject proposal

Project list page

- List of all projects
- Filter by supervisor, number of students who have the project in there preferences, pending action
- Each project has a link to its specific page
- Each project has an indicator of whether staff preferences are being considered

Project page

- Project description
- Supervisor name and email hyperlink
- Supervisor rankings for the project
- Number of people with project in there preferences by rank
- Requests section which allows for approval of project duplication or allocation of project to specific student or both