SYSTEM DEVELOPMENT FOR PLACEMENT MODULES

(POST START OF PLACEMENT)

Final Report

Author: Matthew Trimby
Supervisor: Dr Martin Caminada
Moderator: Steven Arthur

Word count: 23,432

Module code: CM3203
Module Title: One Semester Individual Project
Student Number: C1525379

Abstract

Like many other university schools, the Cardiff School of Computer Science runs a one-year placement module for its students to spend a year working in an external company. This project delivers the implementation of a centralised web platform to monitor and track students' progress, and to facilitate collaboration between students, staff and employers during the placement year.

Acknowledgements

I would like to express my gratitude to Dr Martin Caminada and Dr Catherine Teehan for their continued support and guidance throughout this project. They have provided invaluable advice and feedback throughout the process and supported me greatly through difficult circumstances.

I'd also like to thank my family and close friends who have continually motivated me to work hard, even when the going was tough.

Table of Contents

Table of Figures

1   Introduction

The Computer Science BSc/MSc with a Year In Industry degree programs at Cardiff University allow students to spend a year of their degree working in an external organisation in order to gain experience of applied computer science practices within a professional environment. Having spent a year working in industry as part of this programme, I have direct experience of the numerous benefits it can have on students learning experience, as well as its shortcomings.

One such shortcoming, which this project aims to address, is that the processes for tracking a placement year are inconsistent, split across different mediums, and heavily reliant on students and staff acting on their own initiative. The application should provide users with a single platform to handle all of the core administrative tasks they will need to undertake throughout the year, whilst also providing the School of Computer Science Placement Team with the ability to customise and adapt the process year on year.

1.1   Project Aims

The aim of this project is to design, develop, document and test an industry-standard, bespoke web application to be used by the Cardiff University Placement Team. The application should provide an interface for students and university staff to create, view and complete tasks, upload assignments, and schedule meetings. The overall goal being to make administrative processes easier and more intuitive for all stakeholders.

Further to this, the application should look and feel like it is part of the Cardiff University 'ecosystem' and should allow users to quickly and securely access data via their pre-existing university credentials.

It is integral that the final system is well implemented, using good software engineering practices in order to facilitate the future maintenance and possible expansion of the software by the school in years to come.

1.2   Personal Aims

This project presents an opportunity for me to learn and develop by going through the process of fully designing, implementing and deploying a full-stack web application. With most of my current experience being in front-end development, it's my aim to become more familiar with the full web stack and learn about areas such as RESTful APIs and database design and implementation.

1.3   Beneficiaries

The placement management system stands to benefit several groups once implemented:
- Students will be able to use the system to easily view and carry out tasks associated with a placement year in an intuitive way, allowing them to focus on making the most of their placement
- Supervisors will be able to track all of their students' progress throughout their placement year in one place, making it easier to organise and track students alongside their other responsibilities as university staff
- Placement team staff will be able to use the system to centralise and automate tasks that were previously time-consuming and labour intensive, decreasing their workload

1.4   Project Scope

An important factor in determining the scope of the project was that my application would make up part of a larger system covering the entire placement process - from looking for and applying to placements to their return for their final year of university. The project should cover the processes and procedures necessary to be undertaken during the year the student is on placement whilst another project, with a different set of requirements, would cover the 'pre-placement' part of the application. From the outset it was determined that the two parts of the application would be developed independently, but with the consideration that both parts would be eventually part of the same system. This affected the requirements of the project and influenced some of the design choices made from the beginning.

Due to the nature of the software development process, and uncontrollable external circumstances, the scope of this project evolved during its duration. Whilst the overarching goals of the project stayed largely aligned with those set out at the start, some of the more specific aims and requirements were adjusted during the design and implementation process. Throughout this report I will endeavour to highlight, explain and justify these decisions.

1.5   Assumptions

One of the main assumptions of the project was that it is to be utilised by users that are already part of the Cardiff University 'ecosystem', i.e. users would be existing students and staff who already have a footprint within the various online university systems. The project also assumes the University's continued association with Office365, as the login and authorisation are done via the user's Cardiff University Office365 account – if the university were to move away from its integration with Microsoft then changes would need to be implemented to the application for its continued use.

Also, it is assumed that when the user is interacting with the system, their user data such as their student number and placement provider is already stored in the database. The assumption is made that the user has already been using the independently developed pre-placement part of the system, and that the data collected in that part of the platform is carried over to the during-placement section of the application.

1.6   Approach

During the initial stages of the project, research was carried out into existing solutions and their potential strengths and shortcomings, as well as extensive discussions with members of the School of Computer Science Placement Team to understand the details of the problem. From this, an initial set of requirements was defined in order to detail how the application should work upon completion, which influenced the technologies used to implement the solution. These requirements also helped to influence initial interface designs and use cases for different user types, which provided further detail as to how the application should look and how users should interact with it.

Once the majority of the design work was done, implementation started. This began with the core functionality, such as the user dashboard and integration with Microsoft's Office365 APIs, followed by further features such as document upload and supervisor assignment.

Finally, the application was evaluated against the initial requirements and tested in alignment with the cases outlined in the design stage in order to determine the effectiveness of the implemented solution.

1.7   Outcomes

By the end of the project, a platform was created and deployed that fulfilled a large majority of the requirements set out in its inception. With some further work and refinement, the system will be able to be utilised by real students and staff in the Cardiff University School of Computer Science. The implemented solution is written and documented in such a way so as to make it as maintainable and extendable as possible, allowing for future work to continue in order for the placement application to be fully realised as part of the Cardiff University online ecosystem.

2 Background

2.1 The Problem

With an increasing number of students opting to take a 'Year in Industry' as part of their degree, it's become more and more difficult for the School of Computer Science to process and track student data.

The aim of this project will be to create a fully operational, well-tested web-based platform that allows students, supervisors and staff to carry out their tasks and track their progress during the time that students are on placement. The goal is to create a bespoke, purpose-driven system that is designed to make collaboration as easy as possible for all stakeholders.

From this problem, the following research questions can be derived:
- How can we improve the experience for all stakeholders?
- What parts of the Placement Team's work stack can be optimised by the application?
- How can we track each student's placement in a standardised, uniform way, whilst customising the experience based on their unique parameters?

2.2 Stakeholders

Placement Team Staff
The team of staff within the School of Computer Science (henceforth in this report referred to as the 'Placement Team') which, amongst their other duties, are responsible for managing the placement year programme within the school. The Placement Team set out the structure and schedule of tasks and assessments that will be completed by the student throughout the year.

Academic Supervisors
Academic staff from within the School of Computer Science that monitor and assess students on an individual basis. Supervisors will have several students assigned to them each year and act as a point of contact between the student and the university, as well as marking the academic work produced by the student during the year.

Students
The individuals that are undertaking the placement year. Students will have to complete various administrative and academic tasks throughout the year in order to be assessed on their placement by the School.

2.3 Additional Complexities

Having three distinct stakeholders in the problem increases the complexity of finding an appropriate solution. The final application must balance the needs of all three groups in

order to improve the experience for all parties. An ideal solution would decrease the workload for the Placement Team staff and academic supervisors, whilst also improving the experience for students.

This task's complexity is also compounded due to the variations in the schedule of students' placement years. With each student working at a different external company (with varying placement start dates), each student requires their own personalised timeline of due dates and assessment deadlines.

The wider context also must be taken into account – as an academic institution the University is expected to make sure that all practices are fair and don't give any one student an advantage over another. Naturally, the current processes that the school has in place have been carefully put together in order to account for this. As such, the solution should endeavour not to change the existing processes, but rather to facilitate them and optimise them where possible.

Finally, the process of tracking a placement year is likely to evolve year on year – the implemented solution should have the flexibility to handle these changes without requiring any extensive changes to the code base.

2.4   Existing Solutions

2.4.1   Current Solution

The current solution used to monitor and track placement years is becoming less and less fit for purpose. Currently, students' placement data is manually entered and stored in a Microsoft Excel Spreadsheet. Whenever this data is needed it must be manually accessed. This method worked when the programme began, and there were only single-figure numbers of students on placement, but is now inefficient and time consuming for the Placement Team.

There is also no formal way for supervisors to track their students' progress. Supervisors must maintain their own records about the details of their students and keep updated on their progress by emailing them on an individual basis.

The process for students managing and submitting work is also inefficient – at the beginning of the year students are emailed a collection of documents containing details about the placement processes and forms to be filled out and uploaded at various points during the year. Students are expected to be individually responsible for working out when tasks are due based on their unique start date, meaning there can be a lack of clarity about deadlines for certain documents.

A viable solution should consequently improve on the current implementation by providing a single platform where data is readily accessible for users who require it, and all placement related tasks can be accessed and completed by students and supervisors.

2.4.2 Related Existing Solutions

Name: InPlace
Summary: InPlace [1] is a placement management solution developed by QuantumIT for universities to manage their entire placement programme, from creating connections with employers to tracking students on placement. InPlace currently provides placement management to over 100 academic organisations worldwide.
Positives: • Flexible system for tracking placements of all types • Separate views and functionality for students, supervisors and administrative staff • Bulk notifications and email functionality • Create and manage student schedules • Functionality to flexibly assign supervisors
Drawbacks: • Heavyweight – ships with a huge catalogue of functionality such as analytics, report building, logbooks/timesheets and financial management - much of which is outside the scope of this project • Would require large amounts of student data to be stored by a 3rd party • Only small subsection of software covers the actual functionality required by the School of Computer Science • Current school processes would likely have to be adjusted to fit the format of assessment and tracking provided by InPlace
Conclusions: InPlace is without a doubt the closest existing solution to the problem presented. It provides a large library of features and functionality for placement management across an entire organisation and would likely be a strong choice if Cardiff University was looking to manage all university placements through a single centralised system. However, for the needs of the School of Computer Science it is probably too heavyweight. My solution will seek to implement the features covered by a sub-section of this software, in a way that is directly catered to the needs of the Placement Team.

Name: Creatrix Campus
Summary: Creatrix campus [2] is a cloud-based campus management system that seeks to help higher education organisations organise and centralise their administrate processes. As part of their services they facilitate 'placement management' for institutions.
Positives: • Mobile integration • Create and manage student accounts • Synchronisation with events and calendar • Enables student sign-in through third party services such as Google and LinkedIn
Drawbacks:

- Within the placement management aspect of the service, the main focus is on matching students with placements rather than tracking their placements once they start
- Limited functionality – allows admins to view student and employer data but there is no functionality for tracking and assessing students on placement
- Possible conflict/overlap with systems the university currently has in place to organise administrative work

Conclusions:

Creatrix campus' placement management service provides some interesting ideas for mobile device support, and integration with other login systems. However, as it is part of a larger higher education administration package it lacks the depth of features required by the School of Computer Science.

Name: Sonia

Summary:

Sonia [3] is a placement management software solution package that helps institutions to manage students who are seeking and enrolled on placements. Sonia provides bespoke instances of its software, tailored to the institution's requirements.

Positives:
- Built in integration with existing student record management systems
- Package built around institutions requirements
- Customise documents and reports upload
- Mobile app
- Automated email workflows

Drawbacks:
- Mainly targeted for health/social-care/education placements
- Does not ship ready out of the box

Conclusions:

Because the features of Sonia are implemented based on requirements of the institution it is somewhat difficult to judge its overall effectiveness at solving the presented problem. The main points to take away are that certain features such as student login, document/form upload and ability to view student data are the core concepts that placement management applications are built around.

2.5   Methods and Tools

2.5.1   Potential Methods

Traditional Software Development Lifecycle (Waterfall Method)

The waterfall method involves breaking the project down into stages (requirements, design, implementation, verification and maintenance) and following these steps through linearly. The advantages of the waterfall method are that through the requirements and design stages there is a strongly defined goal state for the application, making progress measurable.

However, the waterfall method is inflexible (does not allow for evolving requirements) and produces a practical implementation late in the development cycle. For this project it would be more useful to be able to develop incrementally, receiving feedback from the intended users and shaping the application to suit their needs as it is developed.

Incremental Method

The incremental method involves developing new software features in small chunks – doing requirements, design, implementation and testing for each new increment. The advantages of this method are that it produces a working implementation early on in the development process, allowing for flexibility.

Iterative Method

The iterative method of software development promotes iteratively performing a scaled down version of the traditional Software Development lifecycle. This model involves repeated prototyping and review of the software product, then using this as the starting point for the next increment if it is accepted.

The idea of iterative prototyping would be useful in this context as it would allow for continuous reviewing of requirements as the software is developed. However, this method can be problematic as not all requirements are gathered upfront.

Agile Method

Agile [4] is an incremental development methodology that involves designing, implementing/testing and reviewing new features incrementally as they are required – it arguably combines the strengths of both the iterative and incremental models.

The incremental nature of agile makes it an attractive method for this project, as it would allow for regular changes of requirements. However, it also involves less view of the final goal state of the project when beginning its implementation. Because this is a short project,

with firmly set requirements with many overlapping dependencies, the lack of vision of the final goal state could cause issues.

The agile method also requires a large amount of collaboration with the 'client (in this case the Placement Team) in the form of regular reviews of progress and new features. If this kind of regular contact with the client is not available, then requirements can become misaligned with the needs of the customer.

2.5.2 Selected Method

For this project I have decided to follow the Agile method, with some small adaptations for the project's unique factors. Incremental prototyping and review by the client will be key to this, and the placement team agreed to a schedule of weekly review meetings. This will allow for continuous review of the product against the client's requirements, meaning the final solution should be suited to the client's real-world needs.

Because this is a relatively small project, to be completed in a short time frame, there are some areas where I will slightly deviate from the 'standard' Agile method. For example, I will be outlining the majority of the requirements for the application as a whole before beginning any implementation. This will allow me to work towards a desired goal state within the allotted time, without drifting from core requirements that are essential to the platform's viability. Additionally, I will be laying out a design for the system architecture before any development begins, this will save time during implementation of individual features as there will already be a view of how they fit into the overall architecture.

2.5.3 Potential Tools

2.5.3.1 Frontend

JavaScript

JavaScript is the obvious choice for creating modern, dynamic web applications. As of January 2019, it is the most popular programming language by GitHub pull requests [5] and is a staple in web development.

As this project is to develop a web application, the choice was made early on to use JavaScript as the base for the front-end – it was also logical to then look into JavaScript frameworks that facilitate the functionality required for the project.

ReactJS

ReactJS [6] is a lightweight JavaScript framework for user interface (UI) development, created and managed by Facebook.

The advantages of ReactJS are that it is lightweight, easy to learn, and uses a Virtual Document Object Model (DOM) in order to reduce the load on the browser. As it is created

and managed by Facebook, React is an up-to-date, industry standard framework with a strong community of developers.

Disadvantages of React include that because its primary purpose is for UI development, it only provides the 'View' layer and requires a multitude of additional libraries in order to build a fully-fledged web application. There is also no in-built application structure, so this must be designed before development begins which can mean that the initial stages of getting the application up and running take longer.

Angular

Angular [7] is an open source JavaScript framework, created and managed by Google, for building mobile and web applications. It is TypeScript (a type-safe superset of JavaScript) based and provides functionality such as data binding, routing and dependency injection without the need for any additional libraries.

An additional advantage of Angular is the Angular Command Line Interface (CLI), allowing the developer to easily create angular applications, generate components, and build projects ready for deployment. Angular projects created through the CLI are also automatically set up for unit testing using the Jasmine testing framework and Karma test-runner.

The disadvantage of Angular is the steep learning curve, due to its deep built-in functionality there is a lot to learn and it can take longer to get to grips with than more lightweight frameworks such as React.

2.5.3.2   Backend

Due to the functionality required for the application (such as displaying and updating data from a database, uploading documents, and user authentication) the system requires a server-side client to process and return data.

The decision was taken to implement a RESTful API[1] to serve JSON data to the front-end client.

Java Spring Boot

Java Spring Boot [8] is an open-source Java framework for creating RESTful microservices.

Spring Boot, being Java-based, provides type safety and support for multi-threading – it's a powerful framework that can handle large amounts of computing at scale. However, Spring Boot applications are heavy: they include a large quantity of dependencies that may go unused in a small API and contain a large amount of boilerplate code which can make debugging difficult.

---

[1] REST(Representational State Transfer) Application Programming Interfaces (APIs) are web services that follow a defined set of constraints to provide interoperability between components on the internet via a request system

Node.js (+ Express)

Node.js [9] is an event-driven JavaScript runtime environment for building and running server-side network applications. Express is a framework for Node.js specifically intended for creating web applications and APIs. Express [10] is a commonly used framework for developing efficient, light-weight RESTful APIs in JavaScript.

Advantages of Node.js Express are that it lightweight, has low memory utilization and gives the developer access to a constantly growing library of additional frameworks through the node package manager (npm). Express is designed to make creating robust APIs quick and easy.

The disadvantage of building RESTful APIs in Node.js is that because it is single-threaded, performance can suffer if performing very heavy-duty computational tasks at scale – but for this project this should not be of great consequence.

2.5.3.3   Database

MongoDB

MongoDB [11] is a schema-less document-based database that stores and returns data in JSON. As the application would be using JavaScript, and the backend API would be returning JSON data MongoDB became a clear choice for storing the applications data as it would store the data in a format close to how it was being used. For this reason, MongoDB was preferred over an SQL-based database.

2.5.3.4   Other tools considered

Microsoft PowerApps

One potential tool that could be used for the implementation of the system is Microsoft PowerApps [12]– a tool developed by Microsoft for rapid, low-code cloud application development tied in with the Microsoft 365 suite.

This tool would allow rapid development of a placement management interface that students could use via their Cardiff University Office365 subscription. One of the biggest advantages of this tool is that all user authentication and data security would be handled by Microsoft, all the developer needs to do is build the interface through the provided tool.

Through this, a solution could be developed that allowed form uploads/submission and some of the other core requirements of the project. However, because the apps must be built from the pre-existing building blocks that the tool provides there is not much flexibility to implement the more bespoke, complex features required. Another disadvantage is that the application developed would be entirely dependent on the University's subscription to Microsoft's software packages – if the University were to move away from the Microsoft ecosystem, the application and all its functionality would be lost.

With this being the case, it was decided that in order to implement an application that satisfied the full user requirements, the tools provided by Microsoft Power Apps weren't powerful enough.

2.5.4 Selected Tools

After researching and comparing the various choices of potential tools, I opted to implement the application using the MEAN (MongoDB, Express, Angular, Node.js) stack: a widely used solution stack for development of web applications.

One of the largest advantages of using this combination of technologies is that all of the components are based in JavaScript. For a project with a relatively short timescale such as this one, this is attractive; it decreases the time overhead spent learning new languages for each part of the stack. Added to this, because this is a fairly commonly used solution stack, there are a variety of middleware packages that are already built to help integrate the components. There is also a wealth of resources on the internet that would help to guide me through the implementation steps.

2.5.5 Additional Tools Utilised

Amazon Elastic Compute Cloud (AWS EC2)

Amazon EC2 [13] is a web service that provides secure cloud computing resource as part of the Amazon Web Services (AWS) ecosystem. In this project EC2 is used to deploy the web application on a virtual Linux Ubuntu machine in the cloud, allowing it to be accessed from anywhere.

Amazon Route 53

Amazon Route 53 [14] is a cloud Domain Name System (DNS) web service that facilitates the routing of users to internet applications. In this project Route 53 is used to connect user requests to the application, deployed on an Amazon EC2 instance.

Figma

Figma is a design tool for creating application prototypes/designs. In this project Figma was used to create visual design mock-ups of the system.

GitLab (Git)

GitLab is a CI/CD (Continuous Integration/Continuous Deployment) tool for managing software development projects using Git. In this project GitLab is primarily used for version control, facilitating incremental development methodologies.

Mongoose

Mongoose [15] is an open source MongoDB object modelling tool for NodeJS. It allows the user to easily write schemas and queries for interacting with MongoDB databases in an asynchronous environment.

Microsoft Graph REST API

The Microsoft Graph API [16] provides a single endpoint for access to a wide variety of data associated with the Microsoft 365 ecosystem. In this project the API was used to access data stored about the user in their Office365 account, such as their 'job title' and display picture.

Microsoft Identity Platform

The Microsoft Identity Platform [17] is a platform that allows developers to build applications that use Microsoft identities for sign-in, authorisation, and the calling of Microsoft APIs. This application uses the Microsoft Authentication Library (MSAL) [18], an open source library for acquiring and caching authorisation tokens, in order to facilitate sign-in and data access with the user's Cardiff University linked Microsoft Office 365 account.

Postman

Postman [19] is a software package designed to aid API development. In this project Postman was used to quickly create and send Create, Read, Update, Delete (CRUD) requests to the backend API for design and testing purposes.

Wireframe.cc

Wireframe.cc is an online wireframing tool for creating interactive application wireframes. This tool was used to create all the wireframes for this project.

Additional Third-party Packages

Third party Angular packages used will be discussed in the implementation section of this report.

2.6   Constraints & Considerations

There are several factors that need to be considered in this project that may affect the outcomes.

Firstly, the nature and scale of this project is not one that I have previously undertaken. It is therefore reasonable to assume that I may experience problems such as unexpected delays due to unforeseen circumstances or problems. Added to this, although I have some web development experience, I have no prior experience with the Angular or NodeJS frameworks, so time will have to be taken out of the project in order to learn these frameworks as I progress through the implementation.

Additionally, there is a strict time constraint on the project, impacting the scale and range of features I will be able to implement. For this reason, it was decided not to implement the platform for mobile users: given the activities that will be undertaken within the application it is likely that most users will be using the system from a desktop computer. Taking this into account it was decided that having to implement and test all features for mobile as well as for desktop would take up a large amount of time compared to the benefit delivered.

3   Specification

This section will endeavour to lay out the requirements and specification of the project in greater detail, breaking down and analysing the problem in order to identify the key features of an implemented solution. These requirements and use cases will also serve as a framework to evaluate the final software implementation against in test cases.

3.1   User Personas

In order gain an understanding of the potential user base and identify specific requirements, user personas were created. These personas were developed based on conversations with students, supervisors and Placement Team staff, research conducted into existing solutions, and personal experience of the placement year process.

The users this application is designed for can be broken down into 3 categories:
1. Computer Science students
2. Placement Team staff
3. Supervisors

As all three groups are part of the School of Computer Science, it can be assumed that the large majority of the userbase will have at least moderate technical experience: using the internet and basic software packages on a regular basis. In some ways, this makes things easier as the design will not have to cater as much to complete novices. However, regular internet/software users will also expect the application to behave in a manner that they are used to, so it becomes more important to design an application that adheres to best practices.

Persona 1

Name: Jessica Bradbury

User Group: 1 - Student

Quote: "I want to focus on learning as much as possible from my placement and impressing my employer"

image source:
https://www.pexels.com/photo/woman-wearing-black-eyeglasses-1239291/

Description:
Jessica is a Computer Science student that has recently secured a lucrative placement at IBM. She knows that students that perform well on the placement year are often invited back to the graduate scheme, so she wants to focus her efforts on impressing her employer.

Goals:
- Quickly and easily view outstanding administrative tasks to be completed, and the dates they are due
- Easily find the contact information of her supervisor
- Be alerted when due dates are coming up
- Be able to upload her work as and when she completes it

Technical Knowledge: Strong

Persona 2

Name: James Anderson

User Group: 2 – Placement Team

Quote: "I want an application that will help me make sure every student is on track to get the most out of their placement year"

https://www.pexels.com/photo/man-wearing-black-zip-up-jacket-near-beach-smiling-at-the-photo-736716/ image source:

Description:

James is a member of the Cardiff Computer Science Placement Team, but this is just one of his many administrative roles within the school. He wants to make sure every student is on track and feels connected to the school during their placement, but is also flooded with work, meaning sometimes there just aren't enough hours in the day. If he could spend less time doing boring administrative tasks, he could spend more time helping students.

Goals:
- Easily be able to view all students enrolled in the placement programme
- Be able to assign students to their supervisors
- Be able to set up tasks for students to complete according to their personal placement year schedule

Technical Knowledge: Moderate

Persona 3

Name: Karen Stone

User Group: 3 – Supervisor

Quote: "I need an organised way to stay up to dates with my students progress"

Description:

Karen, amongst her other responsibilities of researching and teaching, is an academic supervisor for placement students. Each year she supervises 5 students in various locations and wants a way to make sure they're all on track. Often she's assigned students she hasn't yet met so she'd like a way to find out a bit more about them. She has a lot going on, so she finds herself having to often dig back into her email archive to find past communications with her students to remind herself what they've done so far. Karen is also a technical expert; she finds herself frustrated when she uses websites that behave in unexpected ways.

Goals:

- View relevant information about the students she is supervising
- Easily see and download the work her students have submitted
- Mark her students' submissions and give them feedback

Technical Knowledge: Expert

Conclusions

Based on the personas, it was inferred that most users will want quick and easy access to the information they need, when they need it. For the majority of users, this system will facilitate work that diverts their time away from their primary interests - because of this, design decisions should be made so that the user does not have to spend a lot of time looking for the functionality they require. The application should have some sort of dashboard page where the user is provided with the information that is most useful to them. Based on this we can conclude that each user group should see slightly different information and have different access to resources based on what they are using the application for.

3.2   Requirements

This section contains a detailed rundown of the requirements set out for this project.

3.2.1   Functional Requirements

3.2.1.1   Must Have

FR-1: User must be able to log in to the application
Justification: This will allow users to see personalised information relating to them
Acceptance Criteria: • User will enter some credentials • Application will assess these credentials, and if legitimate allow the user to use the application

FR-2: Student must be able to view list of tasks to complete
Justification: This will allow student to see the tasks they need to complete throughout the year
Acceptance Criteria: • When user is logged in as a student they should be presented with 'My tasks' section containing a list of tasks they need to complete

FR-3: Student must be able to see deadline of tasks they are to complete
Justification: This will allow students to find out when certain work is required to be carried out by
Acceptance Criteria: • When user is logged in as a student, they will see the tasks they need to complete • When the task is clicked, a task details screen will be shown which includes a deadline date

FR-4: Students must be able to upload files for tasks
Justification: This will allow the user to submit work pertaining to a particular task
Acceptance Criteria: • When user is logged in as a student and they click on an upload task they should be taken to the file upload page • The file upload page should allow the user to select a file from their computer and upload it

FR-5: Students must be able to see a mark for a piece of work they have submitted
Justification: This will allow students to find out their mark for pieces of work
Acceptance Criteria: • When a user is logged in as a student, they should be able to see a link to a feedback page • When a student clicks this link, they should be taken to a page that shows them any feedback they have received on work they have uploaded

FR-6: Supervisors must be able to see a list of students they are supervising

Justification: This allows supervisors to easily find out which students they are supervising

Acceptance Criteria:
- When a user is logged in as a supervisor, they should see a list of 'my students' which contains a list of the students they are supervising

---

FR-7: Supervisors must be able to see and download students file submissions for a task

Justification: This will allow supervisors to have access to the work of the students they are supervising which will allow them to assess the student on their placement year

Acceptance Criteria:
- When user is logged in as a supervisor, they should be able to click on the name of a student they are supervising and be taken to a page with details about that student
- The student details page should contain a list of the work a student has submitted
- The supervisor should be able to click on a submission to view it and should have the option to download it to their device

---

FR-8: Supervisor must be able to submit a mark for students work

Justification: This will allow the supervisor to mark work through the system

Acceptance Criteria:
- When logged in as a supervisor, user can see the work a student they are supervising has submitted
- User should have an option to submit a mark that allows them to assign a mark to that piece of work

---

FR-9: Placement Team staff must be able to view all students enrolled in the placement program

Justification: This will allow the Placement Team staff to have an overview of all the current students on placement years

Acceptance Criteria:
- When logged in as an admin, the user should see a link to view students
- This should take the user to a page containing a table of all the students currently enrolled in the program

---

FR-10: Placement Team staff must be able to assign a student to a supervisor

Justification: This allows admin staff to be able to assign a supervisor to a particular student in order to track and assess them

Acceptance Criteria:
- When logged in as an admin and viewing all students the user should have the option to assign a supervisor
- The user should be able to select a supervisor from the list and assign this to the student
- The student should now have this supervisor assigned to them

---

FR-11: Placement Team Staff must be able to view tasks set for students to complete

Justification: This allows admins to keep track of the tasks they are asking students to complete
Acceptance Criteria:
• When the user is logged in as an admin, they should see an option to view tasks • On the view tasks page, they should be able to see a list of tasks names that shows the tasks that students are asked to complete

FR-12: Placement Team staff must be able to create new tasks
Justification: This will allow the placement team staff to set up tasks for the student users to complete during their placement year
Acceptance Criteria:
• When logged in as an admin and on the view tasks page, the user should have an option to create a new task • The user should then be presented with a form to fill out task details • On submission of this form, the new task should be created and viewable by students

FR-13: Placement team staff must be able to delete a task
Justification: This ensures that if a task is created incorrectly, or is no longer part of the required process it can be deleted
Acceptance Criteria:
• When user is logged in as an admin and on the view tasks page they should have the option to delete a task • The system should ask them to confirm they want to delete the task to prevent errors • Once deletion is confirmed, the task should no longer be show in the list of tasks

3.2.1.2   Should Have

FR-14: User should be able to log out of the application
Justification: This will allow the user to make sure their account is secure when they are finished using the application on a device used by more than one person
Acceptance Criteria:
• The user should have an option to log out when on a page • When the log out process is completed, when the user tries to use the application, they should be re-directed back to the log in page • Once the user is logged out they will have to log in again to use the application

FR-15: Students should be able to view their previous submissions for a particular task
Justification: This will allow the user to see the work they have already submitted for a task to verify they have submitted correctly
Acceptance Criteria:
• When user is logged in as a student, tasks they have completed should still be displayed in the list of tasks

- When the user clicks on a task that they have already completed they will be taken to a page where they can see their previous submissions
- User should be able to preview the submitted document

FR-16: Student should be able to see information about their assigned supervisor

Justification: This will allow the student to be able to contact the supervisor if needed

Acceptance Criteria:
- When logged in as a student, the user should be able to see the supervisor they have been assigned
- User should have the option to view contact information for the supervisor
- When this option is clicked, a pop-up modal should appear with the information about the student's assigned supervisor

---

FR-17: Student should be able to arrange a meeting with their supervisor

Justification: This will allow for students to organise a meeting within the application instead of by email

Acceptance Criteria:
- When logged in as a student, the user should see tasks for arranging the required supervisor meetings in their list of tasks
- When the user clicks on the task they should be taken to a page where they can submit a prospective meeting time and date for the task
- When this is submitted the supervisor should be able to see this prospective meeting when they log in

---

FR-18: Students should be able to view profile information about themselves

Justification: This allows the student to validate that any information stored within the system about them is correct

Acceptance Criteria:
- When the user is logged in as a student, they should see a summary of their information including name, picture, supervisor, placement provider and start date

---

FR-19: Student should be able to distinguish tasks they have already completed from those they have not

Justification: This allows the student to keep track of which tasks they have completed and which they are yet to complete

Acceptance Criteria:
- When logged in as a student, the user should see a list of tasks
- In the list of tasks, if the task has already been completed the task should be coloured/formatted differently to indicate it has been previously completed

---

FR-20: Tasks should be marked as done once they have been completed

Justification: This allows the system to automatically mark tasks as completed without the user having to keep track manually

Acceptance Criteria:

- When the user is logged in and they complete the process for a task this task should be marked as done by the system
- When the user navigates back to the list of tasks, that task should be formatted differently to indicate that it has been completed

---

FR-21: Supervisors should optionally be able to submit feedback comments alongside a mark for a student submission

Justification: This allows the supervisors to give the student feedback on their work through the placement management system

Acceptance Criteria:
- When logged in as a supervisor, the user should be able to see a list of submissions for a given student and have the option to submit a mark
- When submitting a mark, the user should be given the option to add feedback comments if they desire
- When the user submits the mark and comments, this feedback should then be available to the student

---

FR-22: Supervisors should be able to upload documents relating to a student they are supervising

Justification: This allows the supervisor to upload supporting documents related to the student as evidence of their progress

Acceptance Criteria:
- When logged in as a supervisor and viewing details about a student, the user should have an option to upload supporting documents
- The user should be able to select a document to upload and submit it
- When the document is submitted, it should now appear in a list of supporting documents for the student

---

FR-23: Supervisors should be able to view relevant details about students they are supervising

Justification: This will allow the supervisor to see information about the student which will allow them find out details about the student's placement or to contact the student

Acceptance Criteria:
- When logged in as a supervisor, user should see a list of students they are supervising
- When clicking on one of the students, the supervisor should be taken to a page presenting a summary of information about the student
- The students id, name, email and placement provider should be displayed

---

FR-24: Placement Staff should be able to edit the details of current tasks

Justification: This will allow admins to change details such as the description or deadline date of a task without having to delete the task and create a new one

Acceptance Criteria:
- When logged in as an admin and on the view tasks page, the user should be presented with an option to edit a task

- When this is clicked, a form should appear with the task's details filled in, the user should then be able to change these details and submit the form
- When this has been submitted, if the user clicks the edit task button again the new details should be displayed

---

FR-25: Placement Team staff should be able to change student details

Justification: This will mean that if any of the students' data is incorrect, it can be edited by an admin through the application

Acceptance Criteria:
- When the user is logged in as a supervisor and on the view students page, they should have an option to edit student's details
- When this is clicked the user should be presented with a screen that shows the students details in a form
- The details can be changed and then the form submitted
- When the for is submitted, the user's details are updated and the new values are visible in the table

---

FR-26: Placement Team staff should also be able to see students they are supervising

Justification: Some members of the placement team will also act as supervisors, so they should also be able to see the details etc of the students they are supervising in the same way was any other supervisor

Acceptance Criteria:
- When logged in as an admin the user should be able to see a list of students they are supervising, if they are assigned to supervise students

---

FR-27: Placement Team staff should be able to sort the list of students by field

Justification: This will allow staff to more quickly find the students they are looking for in the view students page

Acceptance Criteria:
- When logged in as an admin, user should be able to see a table of all students enrolled in the placement program
- Next to some table headings there should be a button for sorting
- When this button is clicked, the table should be re-organised in descending order according to the field the button was clicked on
- When clicked a second time, the table should be sorted in descending order according to that field

---

FR-28: Placement Team staff should be able to search for students by key identifiers

Justification: This will mean that if the staff need to see details about a particular student that they already know some information about they can find them quickly

Acceptance Criteria:

- When logged in as an admin and on the view students page, user should see a search box
- When a word is typed in the search box, data entries in the table of students should be filtered by students whose name, id or supervisor match that string
- Only students matching the string should now be displayed in the table

---

FR-29: Placement Team Staff should be able to give tasks deadlines

Justification: This will mean that tasks will be able to have an associated deadline for the students to complete them by

Acceptance Criteria:
- When logged in as a supervisor and on the create task page the user should have the option to set a deadline date and time for the task
- When this is set and the task is submitted, this deadline should appear to this and other users in the task details

3.2.1.3 Could Have

FR-30: System could get and display data about users from the Microsoft Office365 API

Justification: Some useful data about users is already stored by Microsoft as part of the University's Office365 subscription, this would allow this data to be used in the application

Acceptance Criteria:
- When user is logged in, they can see data about themselves
- Some of this data should have come from the Microsoft365 API

FR-31: Student could receive an email when a deadline is approaching

Justification: Allows the student to be reminded about upcoming deadlines so they are less likely to miss them and be penalised

Acceptance Criteria:
- When the user is a given time away from a particular deadline, they should receive an email at the address associated with their account
- This email should contain the name of the task the deadline is for and encourage the user to log in and complete it

---

FR-32: Student could be able to edit their own profile information

Justification: This would allow students to edit any information stored by the system about it if they knew it to be wrong without having to contact an admin

Acceptance Criteria:
- When logged in as a student, the user should see an option to edit their profile information
- When clicked, the user should then be presented with a editable form containing their current information

- On submission this data should be updated to the new values from the form for all users

---

FR-33: Students could see a progress bar which shows them the percentage of tasks they have completed

Justification: This allows the student to, at a glance, get an idea of how far they have progressed through the tasks they need to complete

Acceptance Criteria:
- When logged in as a student, the user should be able to see a list of the tasks they need to complete
- Above this list there should be a progress bar that is filled to a certain level dependent on how many tasks they have completed
- The level the bar is filtered to should be relative to the number of tasks completed divided by the total number of tasks

---

FR-34: Placement Team Staff could create scheduled emails to send to all students at certain points during their placement

Justification: This would allow admins to send automated emails to students throughout the year

Acceptance Criteria:
- When logged in as an admin, user should see an option to create a scheduled email
- User should be able to enter a subject, body and date relative to start date
- User should be able to submit this and then see it in a list of scheduled emails

---

FR-35: Placement Team Staff could be able to save a task as a draft rather than publish it to students

Justification: This would allow admins to create tasks and then decide when they wanted to publish them – this might mean that throughout the year staff could add new tasks for next year's placement students and then only publish them when needed

Acceptance Criteria:
- When logged in as an admin and creating or editing a task, the user is given the option to publish the task or save it as a draft
- If the user opts to save as a draft, the task data should be saved but it should appear in a list of draft tasks
- Draft tasks should not be visible to students

3.2.2 Non-functional Requirements

NFR-1: System should work on all main browsers
Justification: This will allow users to use the system from whichever browser they prefer
Acceptance Criteria: • Site pages should be correctly displayed for IE, Chrome, Firefox and Safari

NFR-2: System should load user data within a reasonable amount of time
Justification: So that the user does not have to wait a long time while the app is loading data
Acceptance Criteria: • When on pages such as the dashboard that load data from the API, data should take no longer than 5 seconds to appear

NFR-3: System should be intuitive and easy to use
Justification: This will allow users to have a good experience using the system
Acceptance Criteria: • System should be reviewed against Nielsen's usability heuristics

NFR-4: System should not allow users to change other user's data without correct permissions
Justification: This is to decrease risk of malicious users changing data
Acceptance Criteria: • When logged in as a student or supervisor, user should not have access to pages that allow them to change data

NFR-5: System should update data in database on action
Justification: All changes should be updated in the database when an action is made, not just in the UI – this means changes are always saved in the event of page refresh
Acceptance Criteria: • When action is performed such as 'create task', the data should be pushed and saved to database

3.3 Use Cases

The following section outlines the main use cases for the application, based on the requirements. The diagram in figure 1 gives an overview of the use cases in relation to each type of user.
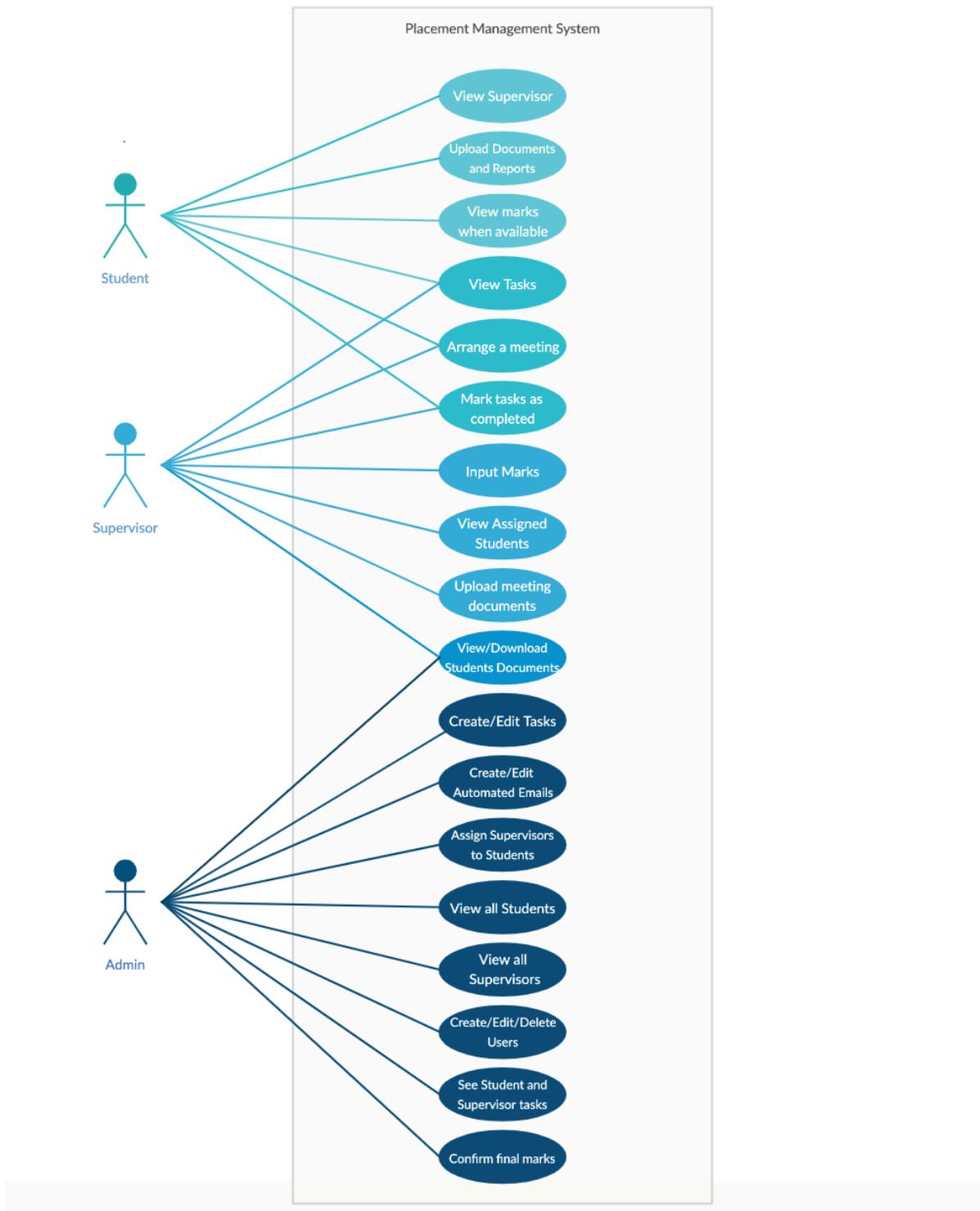

Figure 1: Use Case Diagram

Detailed below are some of the main use cases for each user

Use Case 1
Name: View Supervisor
User: Student
Preconditions: • User account exists • User type is student • User has been assigned a supervisor
Main Flow: 1. User presented with login page 2. User enters credentials 3. Credentials verified by backend 4. User presented with dashboard page - In profile card on dashboard page, supervisor's name is displayed as a link 5. User clicks link 6. User presented with pop-up modal containing contact information about their supervisor
Alternative Flow: 1A. User navigates to system and is already logged in, user taken straight to dashboard page

Use Case 2
Name: Uploading a document
User: Student
Preconditions: • User is logged in • User type is student • Tasks are available to students
Main Flow: 1. User is presented with dashboard page, containing list of links to tasks to be completed 2. User clicks on link to upload file for uncompleted task 3. User taken to task page 4. System retrieves task details from database 5. User presented with list of task details and a form 6. User clicks to add file 7. User selects file from file browser 8. System validates file is correct type 9. (Optional) User can choose to add a comment to upload 10. User clicks button to submit file 11. System displays loading spinner while file upload attempted 12. On successful file upload user is taken to review submissions screen where all submissions for that task are shown
Alternative Flow: 1A. 1. User clicks on link for upload task they have already completed

1A. 2. User taken to review submissions page and presented with list of previous files uploaded for this task

1A. 3. User clicks on link to upload another file for this task

4A. User drags and drops file into file area from open file browser window

10A. File upload fails, user is presented with an error message

Use Case 3
Name: View Marks
User: Student
Preconditions: • User is logged in • User is of type student
Main Flow: 1. User is presented with dashboard page 2. User clicks link to view marks 3. User taken to feedback page 4. System retrieves user feedback from database and displays list of tasks the user has received feedback for 5. User clicks task they would like to view feedback for 6. Mark and comments for the feedback for this task are displayed on screen
Alternative Flow: 4A1. User does not have any feedback available 4A2. Page displays message informing the student they do not have any feedback to view yet

Use Case 4
Name: Arrange a meeting
User: Student
Preconditions: • User is logged in • User type is student • Task of type meeting is available to students
Main Flow: 1. User is presented with dashboard page, including list of tasks to be completed 2. User clicks link to task of type meeting 3. User is taken to a schedule meeting page 4. Details about task are loaded from the database and displayed on page along with a form 5. Student inputs date and time to form 6. Student clicks submit 7. System attempts to add meeting to database 8. On success, message is shown to user saying their proposed meeting time has been submitted

Alternative Flow:
8A1. Attempt to add meeting to database fails
8A2. User presented with an error message describing the failure

Use Case 5
Name: View Assigned students and details about student
User: Supervisor

Preconditions:
  • User is logged in
  • User is of type supervisor
  • User has been assigned at least one student to supervise

Main Flow:
  1. User is presented with dashboard page
  2. List of students assigned to supervisor is retrieved from database and displayed on page
  3. User clicks on the name of a student they are supervising
  4. User taken to student details page
  5. System retrieves details for this student from database, if successful information about the student is displayed on page

Alternative Flow: 5A. Student details cannot be loaded from database; error message is displayed

Use Case 6
Name: View/Download students work
User: Supervisor

Preconditions:
  • User is logged in
  • User type is Supervisor
  • User has followed steps from Use Case 5 to get to student details page

Main Flow:
  1. User selects submissions tab
  2. List of tasks student has completed is shown
  3. User clicks on a task
  4. Page directed to view submissions page
  5. Submissions for this student for this task are loaded from the database
  6. List of submissions displayed
  7. (Optional) user clicks a submission to view a preview
  8. (Optional) user clicks download button to download submission

Alternative Flow: N/A
Related Use Cases: Use Case 5

Use Case 7
Name: Mark Assignment

User: Supervisor
Preconditions: • User is logged in • User type is Supervisor • User has followed steps from Use Case 5 to get to student details page
Main Flow: 1. User selects submissions tab 2. List of tasks the student has completed is shown 3. User clicks button to mark work 4. Form displayed 5. User inputs mark 6. (Optional) User inputs feedback comments 7. User clicks submit 8. Feedback is uploaded to database 9. On success, list of tasks is displayed again with success notification
Alternative Flow: 3A1. Student has not uploaded any work 3A2. In place of list, message is displayed indicating the user has not submitted any work for marking
Related Use Cases: Use Case 5

Use Case 8
Name: View Tasks
User: Admin
Preconditions: • User is logged in • User type is admin
Main Flow: 1. User presented with dashboard page 2. User clicks view tasks link 3. Redirect to view tasks page 4. Tasks are loaded from database and displayed in list
Alternative Flow: 4A1. No tasks have been created yet 4A2. Message is displayed informing user no tasks have been created yet

Use Case 9
Name: Create new task
User: Admin
Preconditions: • User is logged in • User type is Admin • User has followed steps from Use Case 8 and is on view tasks page
Main Flow: 1. User clicks button to create new task 2. Redirect to create task page

3. User presented with form
    a. User inputs task name
    b. User inputs task deadline date
    c. User inputs task deadline time
    d. User inputs task description
4. User clicks button to submit form
5. Task is uploaded to database
6. On success user is redirected to view tasks page

Alternative Flow:
5A1. Task upload fails
5A2. User presented with error message informing them of task creation failure

Related Use Cases: Use Case 9

Use Case 10
Name: Edit task
User: Admin

Preconditions:
- User is logged in
- User type is Admin
- User has followed steps from Use Case 8 and is on view tasks page
- There is at least one task already created

Main Flow:
1. User clicks edit button for one of the displayed tasks
2. User is taken to edit task page
3. Task details are loaded from database
4. Task details are loaded into form inputs
5. (Optional) User edits form fields
6. User clicks submit button
7. Task update uploaded to database
8. Redirect back to view tasks page
9. Notification displayed confirming task successfully edited

Alternative Flow:
7A1. Task update fails
7A2. Display notification of error

Use Case 11
Name: View All Students
User: Admin

Preconditions:
- User is logged in
- User type is Admin

Main Flow:
1. User presented with dashboard page
2. User clicks link to view all students
3. Redirect to view students page
4. System retrieves account data about all student accounts from database

5. Display table of students with some details such as name, student ID and placement company + start date
Alternative Flow:
4A1.   No student accounts present in database
4A2.   Display message informing user there are currently no students in system

Use Case 12
Name: Assign supervisor to student
User: Admin
Preconditions:

• User is logged in
   • User type is admin
   • User has followed steps from Use Case 11 and is on view students page

Main Flow:

1. User clicks button to assign selected student a supervisor
   2. List of currently available supervisors retrieved from database
   3. Display list of available supervisors
   4. User clicks on a supervisor to select
   5. User clicks button to submit
   6. Data updated in database
   7. On success, notification displayed informing user supervisor has been assigned

Alternative Flow:
1A. (Optional) User filters list of students by string to find student, then clicks button to assign this student a supervisor   3A. Use already has supervisor selected; this supervisor is selected in list by default

4 Design

This section of the report details the design of the application. Due to the nature of the Agile implementation method, this design was done incrementally – all design work done throughout the process is documented in this section.

4.1 User Interface Design

The design of the application's UI is centred around being able to quickly and easily carry out tasks with minimal errors. The design should give high visibility of key features, allowing the user to easily navigate to desired resources, and should provide visual confirmation of actions performed in order to increase clarity.

4.1.1 Wireframes

Wireframes are schematics designed to communicate the structure and functionality of an application. As such, they do not necessarily incorporate detailed visual design, but are focused more on layout and flow of the application.

These wireframes are designed based on the requirements and use cases.

The look of the application will vary based on the whether the user is a student, supervisor or admin, thus the wireframes are divided by user type.

All Users
Some screens will look the same for all users.

Login Screen
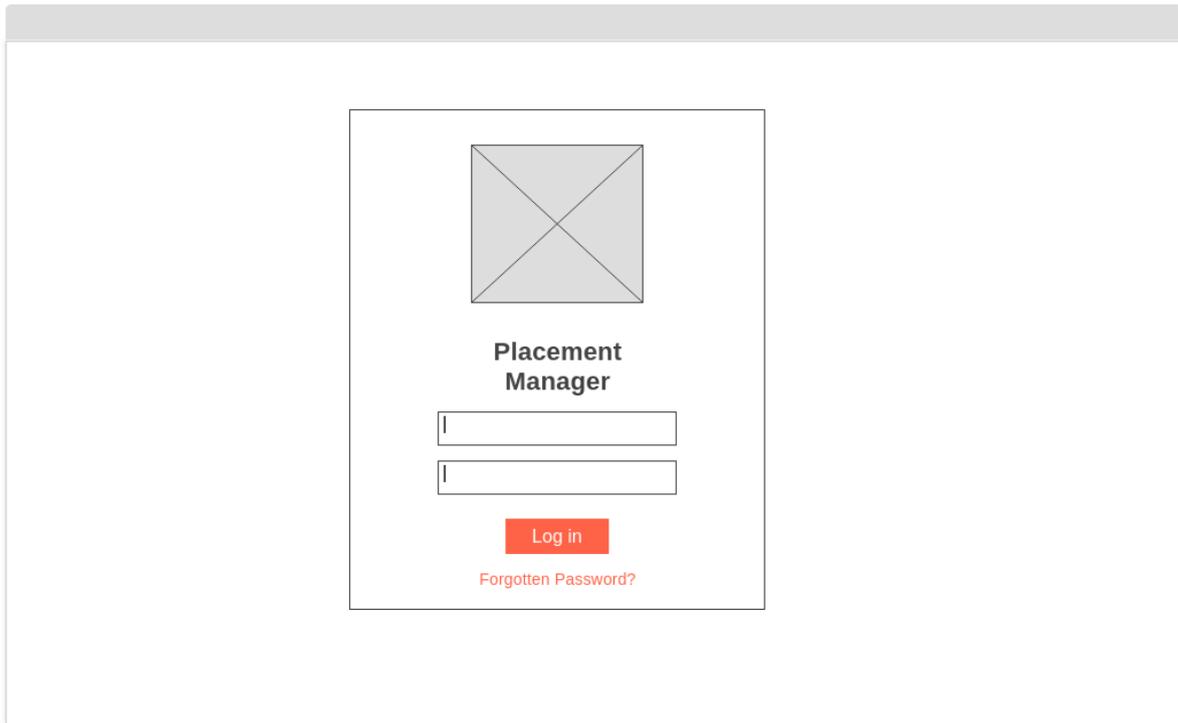
Figure 2: Wireframe - Log in screen

Features

Feature	Description/Justification
Image	Cardiff University logo – this makes it immediately clear to the user that this application is under the wider umbrella of Cardiff University web resources
Username/Password input boxes	Due to the standard structure of the form, the functionality of these input boxes should be obvious to the user. However, to make it explicit the boxes will be filled with 'placeholder' values 'username' and 'password'
Log In Button	This button is clicked to submit the log in details. The words 'log in' make the button's function clear to the user. The button will also be coloured to draw user's attention.
Forgotten Password link	This button will take the user to the Cardiff University password reset page

Student View

Wireframes for the application from the student view.

Dashboard
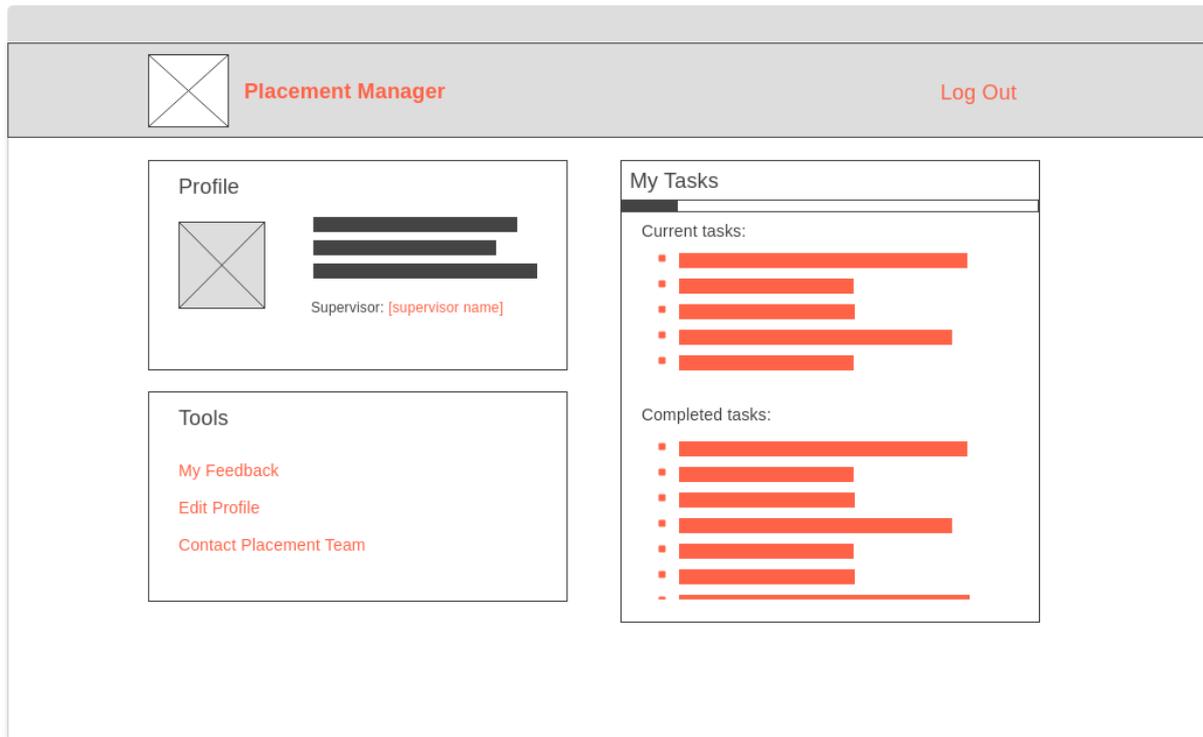
Figure 3: Wireframe - Student Dashboard

Feature	Description/Justification
Navigation Bar	Present on all pages. • Contains Cardiff University to visually tie the application into the university's other web resources. • 'Placement Manager' title text describes the application and also serves as a link back to the home/dashboard page from anywhere else in the application. • 'Log out' link allows user to log out quickly and easily from anywhere in the application.
Profile Card	• User avatar/picture – provides visual confirmation that user is logged in to their account • Profile text – will show information such as name, Student ID and placement provider, giving user quick overview of data stored about them in the system • Supervisor name – shows supervisor if assigned. Name is a link that can be clicked to display a modal giving user quick overview of details about their supervisor

Tools card	Gives user quick access to links to useful features of the application. Allows for fast navigation of the system if the user knows what they want to do.
Tasks Card	Gives user immediate overview of tasks they have completed and tasks they need to complete throughout the year • Progress bar – gives user quick visual representation of the percentage of tasks they have completed at the current time • Current tasks – list of tasks the user needs to complete, displayed as clickable links that will take user directly to page where they can complete that task • Completed tasks – list of tasks the user has already completed, clicking these links will take the user to the page where they can view their past submissions

Supervisor Details Modal

This modal appears over the top of the dashboard page when the user clicks on the supervisor name in the profile card.
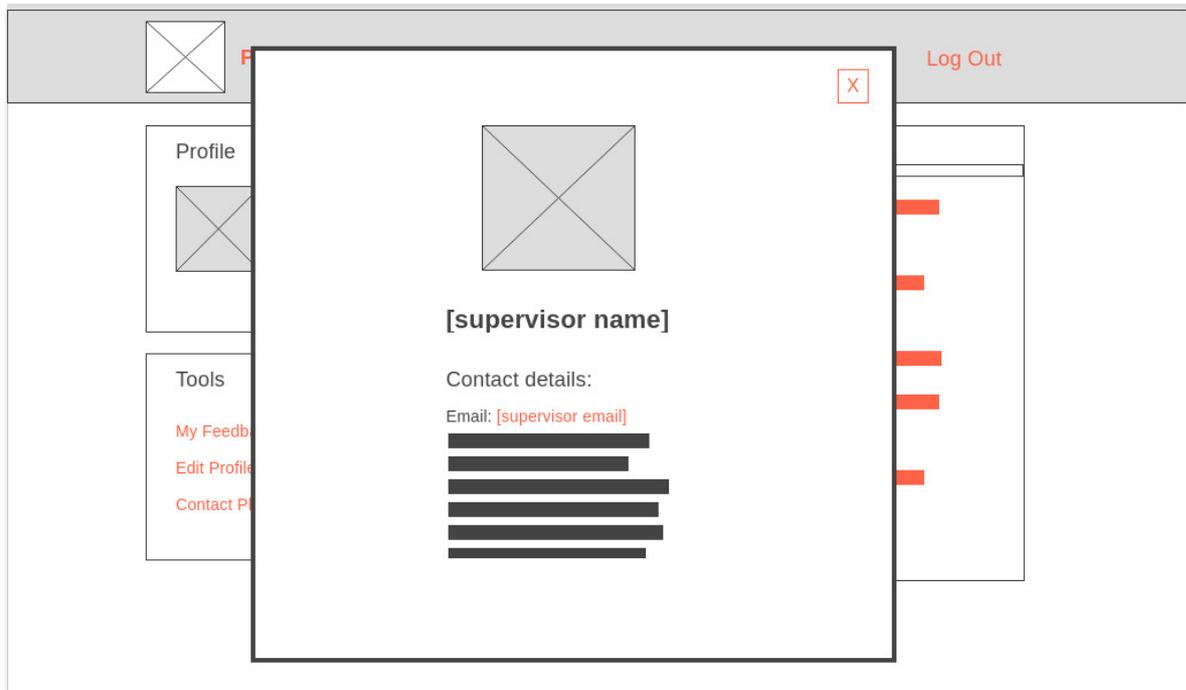
Figure 4: Wireframe - Student Dashboard - supervisor modal

Feature	Description/Justification
Supervisor picture	Shows picture of supervisor – this feature allows user to visually recognise supervisor, creating familiarity and strengthening relationship
Contact details section	Contains various contact details for the supervisor, allows user to quickly find a way to communicate with their supervisor if they need to. Email link will open new email composition page in users default email client.

File Upload Page

User will be taken to this page when they click a task on the dashboard of type 'upload'. This page is designed to facilitate easy upload of assessments and supporting documents throughout the placement year.

Figure 5: Wireframe - Task - Upload file

Feature	Description/Justification
Page heading	Display task name at the top of the page so user knows where they are in the application and has confirmation they are uploading the correct file.
Details boxes	Sets out key information about task to user
Preview window	Allows user to see a preview of the file they select for upload; this means user can visually confirm they have selected the correct file.
File box	Can be clicked to open default file explorer, or file can be dragged and dropped. File will be validated on add.
Upload button	Button will be disabled until a valid file had been selected to prevent accidental submission errors. Button will be coloured to highlight it to user.

View Submissions

This page will allow user to view documents they have previously submitted for an upload task.
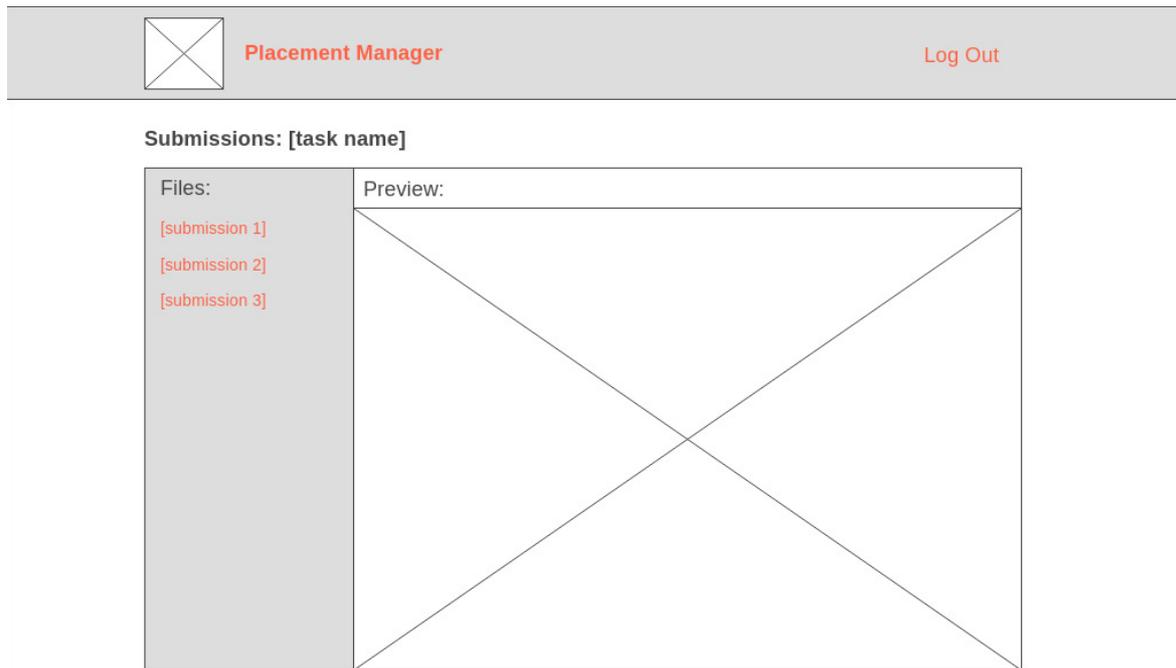
Figure 6:Wireframe - View Submissions Page

Feature	Description/Justification
Title	Display's task name to inform user which task they are viewing submissions for

File list side panel	User can see all the files uploaded for this task in a single list – list should be ordered chronologically, with the most recent upload at the top. Clicking the name of the task should display this file in the preview window.
Preview window	Allows user to see a preview of the file – if user has navigated from upload page, the file they just uploaded should be previewed by default

Feedback Page

This page allows user to view marks/feedback they have received for each upload task when their supervisor has marked it.
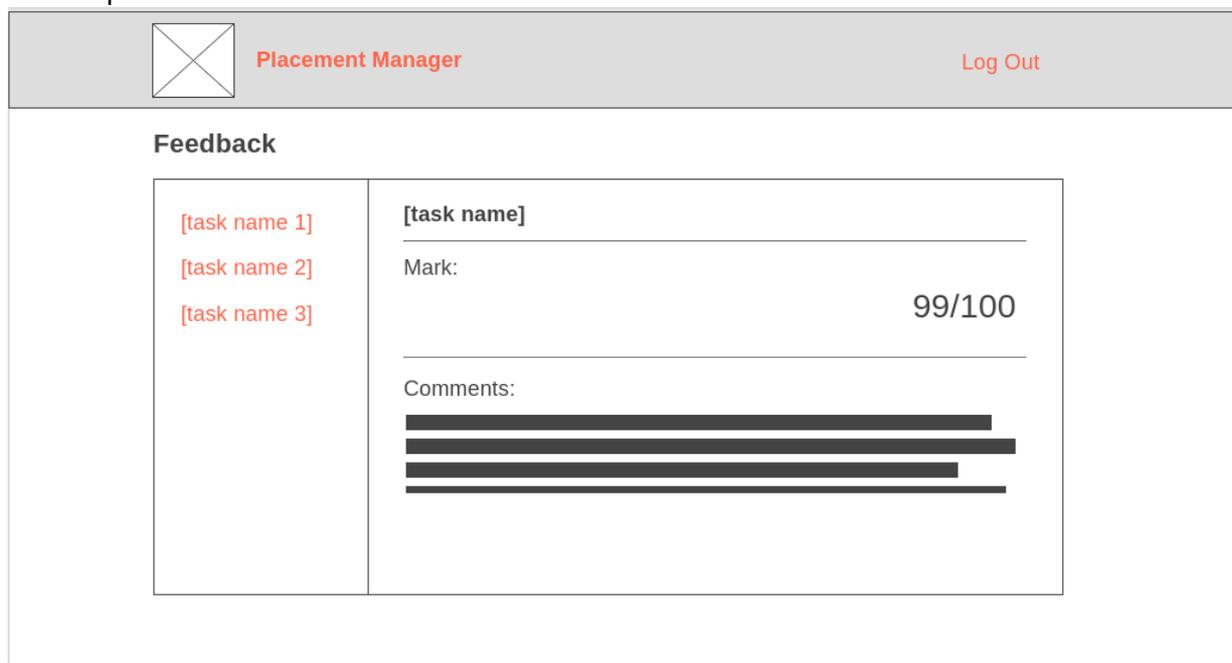
Figure 7: Wireframe - Feedback Page

Feature	Description/Justification
Task list side panel	User can see a list of all the tasks they have received feedback for. These tasks should be displayed in chronological order with the most recent feedback first. Clicking the task name will display the feedback for that task in the feedback section, allowing user to easily navigate between feedback tasks.
Feedback section	Contains feedback details for task • Task name – informs user which task they are viewing feedback for

	• Mark – Large font size because this is important to user
Preview window	Allows user to see a preview of the file – if user has navigated from upload page, the file they just uploaded should be previewed by default

Supervisor View

Wireframes detailing layout and functionality for pages from the supervisor view. Any shared features are explained in student view wireframes.

Supervisor Dashboard

The dashboard page for supervisors is largely the same as it is for students. However, rather than having user's tasks in the right-hand column, the user is instead presented with a list of students they are supervising/moderating.
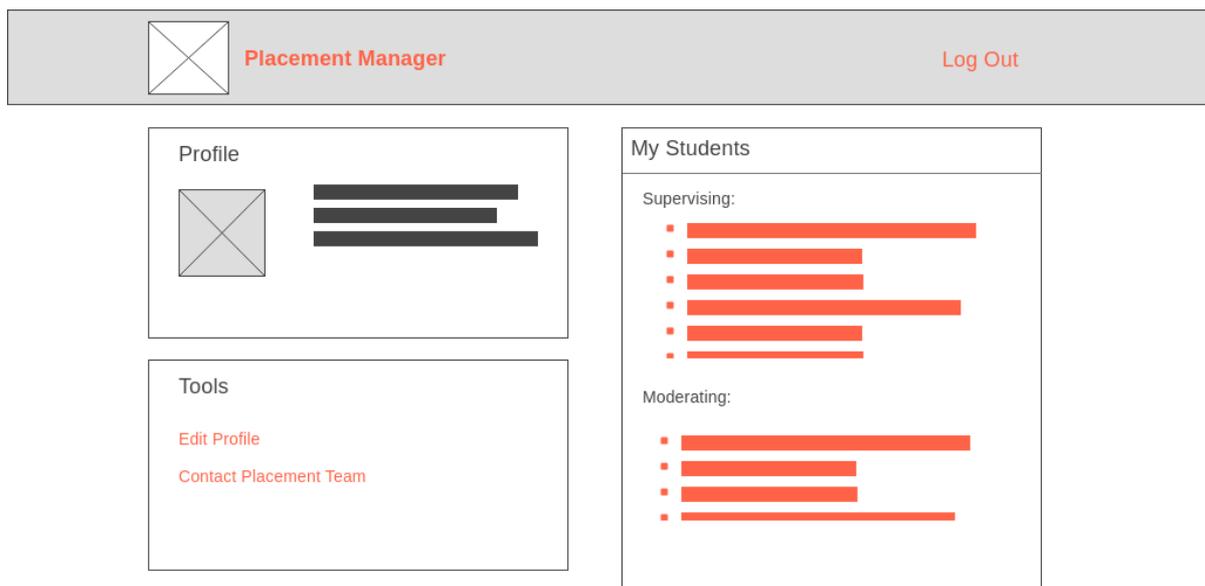
Figure 8: Wireframe - Supervisor Dashboard

Feature	Description/Justification
My Students card	Displays students that the user is supervising/moderating. • To distinguish between students the user is supervising and moderating, the lists are separated into two

	• Clicking the name of a student in this list will take user to details page for that student

Student Details

The student details page will have 4 distinct sections which will be shown depending on the tab selected in the left-hand panel. This allows the user to choose which specific information they would like to be displayed, rather than cluttering the UI trying to fit all of the information into a single view.

Student details - Details tab

This is the default tab if navigating from the dashboard page as it provides a generic overview of potentially useful information about the student.
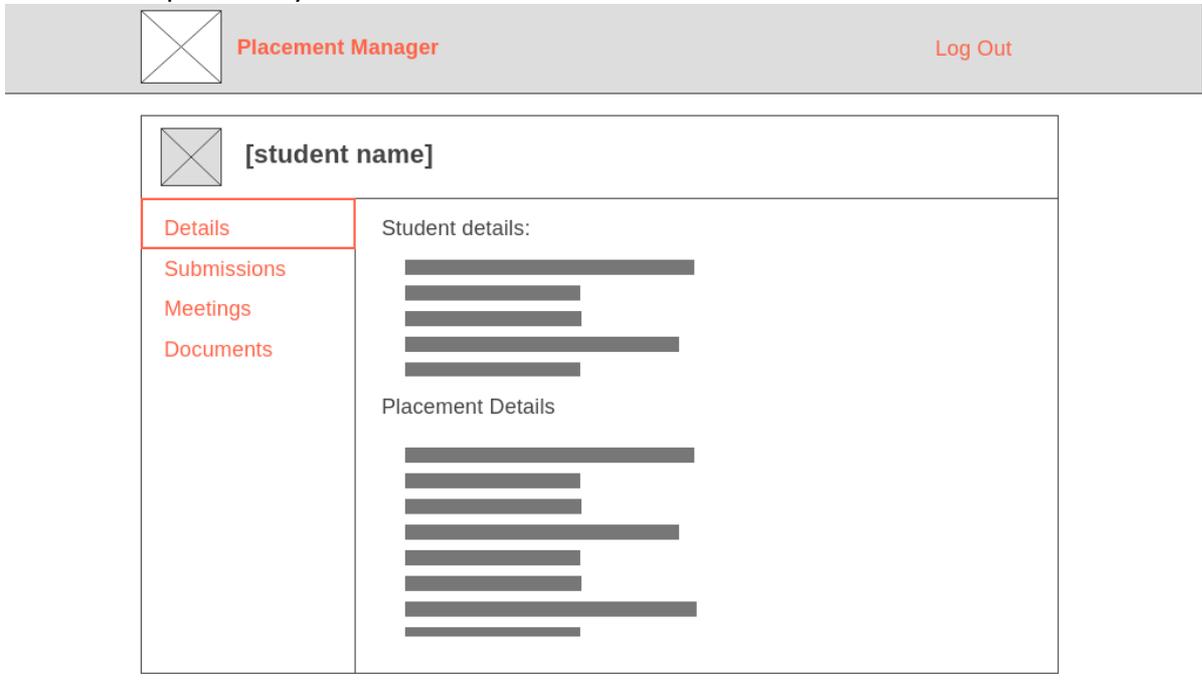
Figure 9: Wireframe - Student Details - Details Tab

Feature	Description/Justification
Student Picture	Student's display picture is shown in the top left of the card, providing the user with visual recognition of the Student. This can help user to personify student rather than just seeing data.
Tab selection panel	This side panel allows user to select the information they wish to display for the user
Student details	Displays details relating to the student in key-value pairs, such as: name, student id, email, supervisor

	name – allows user to quickly find information necessary to contact student
Placement details	Displays key-value list of placement-specific details such as: workplace supervisor, placement provider, location, start date

Student details – submissions tab

This tab allows supervisor to view and mark the files that the student has submitted.
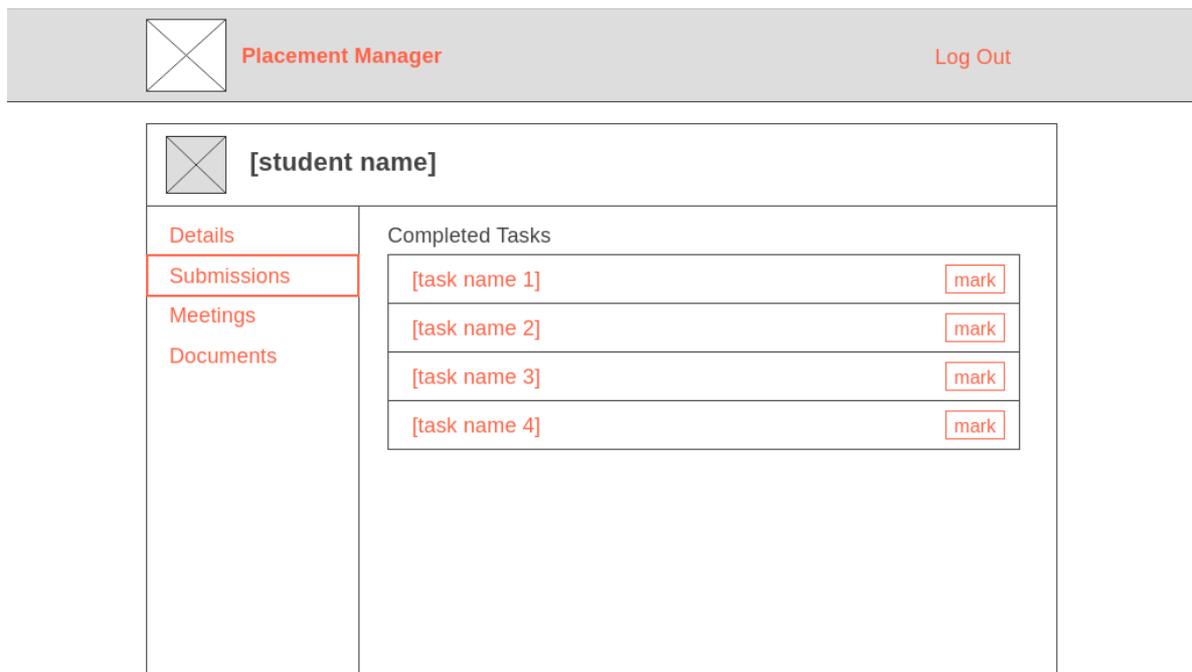
Figure 10: Wireframe - Student details - Submissions Tab

Feature	Description/Justification
Completed tasks list	Shows list of tasks the user has submitted work for • If task name is clicked, then user is taken to submissions page for that task for the student (as shown in figure 6) – allows fast access to students work • Mark button – when button is clicked a modal is displayed for the supervisor to input a mark and comments – this button will only be displayed if the task upload type is 'assessment' as supporting documents do not require marking

Student Details – submissions tab – mark modal
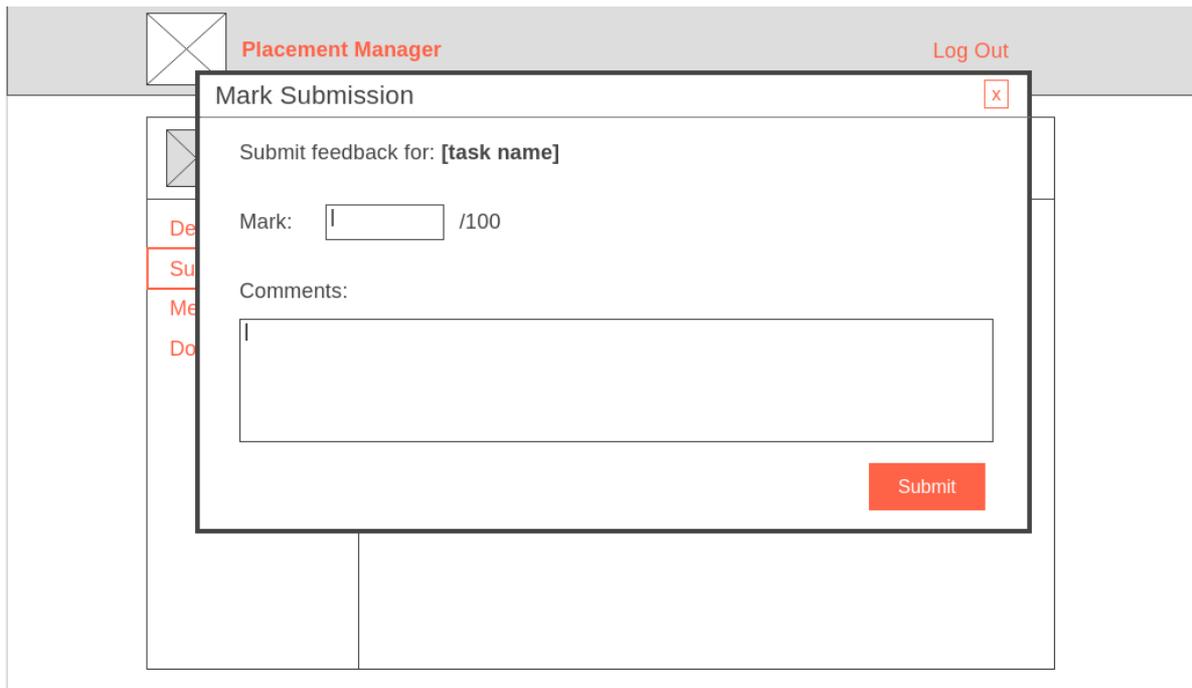
Figure 11: Wireframe - Student Details - Submissions - Mark Submission Modal

Feature	Description/Justification
Task name	Confirmation for user that they are submitting marks for the correct task
Mark	Allows user to input the mark for the task. Displays maximum mark available for task as reminder to user
Comments	Text box to allow for additional feedback comments, use textarea input to encourage longer comments for more constructive student feedback.
Submit button	Coloured to attract user attention

Student details - Meetings

Page displaying details about any meetings the user has scheduled with this student
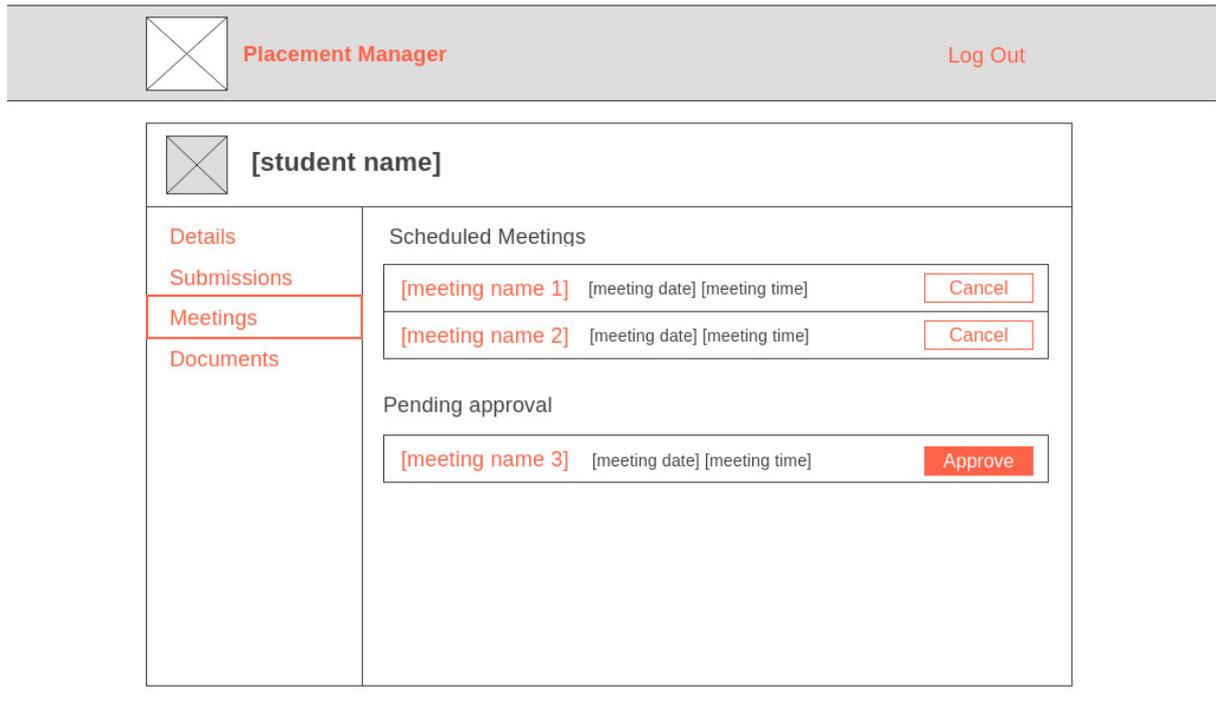
Figure 12: Wireframe - Student Details - Meetings Tab

Feature	Description/Justification
Scheduled meetings	Displays list of meetings currently scheduled with student. • Displays meeting name, date and time for quick access to information relating to meeting • Cancel button – allows user to cancel the meeting, this is not a primary action for the user on this page so secondary 'outline' button is used
Pending approval	List of meetings that need to be approved by supervisor. • Meetings must be approved by supervisor before being scheduled as this confirms that both student and supervisor are aware of meeting and confirmed they are available. • Approve button is primary action for this page so is coloured to attract user attention

Student Details – Documents

This tab allows the supervisor to view and upload documents from placement visits and any other material they want to upload related to this student.
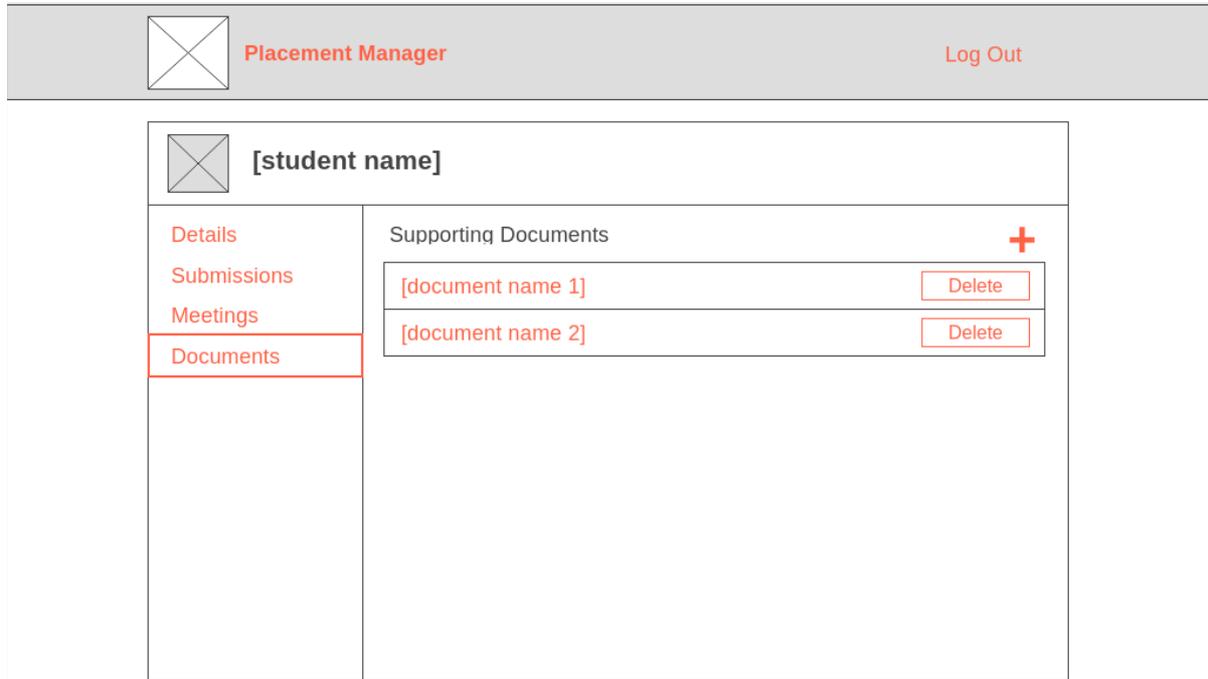
Figure 13: Wireframes - Student Details - Documents Tab

Feature	Description/Justification
Add documents '+' symbol	• '+' symbol is widely used across the web, so we can assume users will understand they can click this symbol to add documents • takes user to document upload page (as in figure 5)
Supporting documents list	List of documents uploaded related to this student. • Clicking on the document page will take user to document preview page similar to that in figure 6. • Delete button – allows user to delete document, this button should trigger a pop up window asking for confirmation that user wants to delete the document to prevent accidental deletion

Admin View

Wireframes for pages as viewed by a user with admin privileges – any shared features with other user types are detailed in previous wireframes.

Admin Dashboard

The admin dashboard is purposefully very similar to the supervisor dashboard, this is for two reasons:
1. Placement Staff can also supervise students, so they need the same supervisor features
2. Users who are admins should not have to learn a whole new site layout to use administrative features
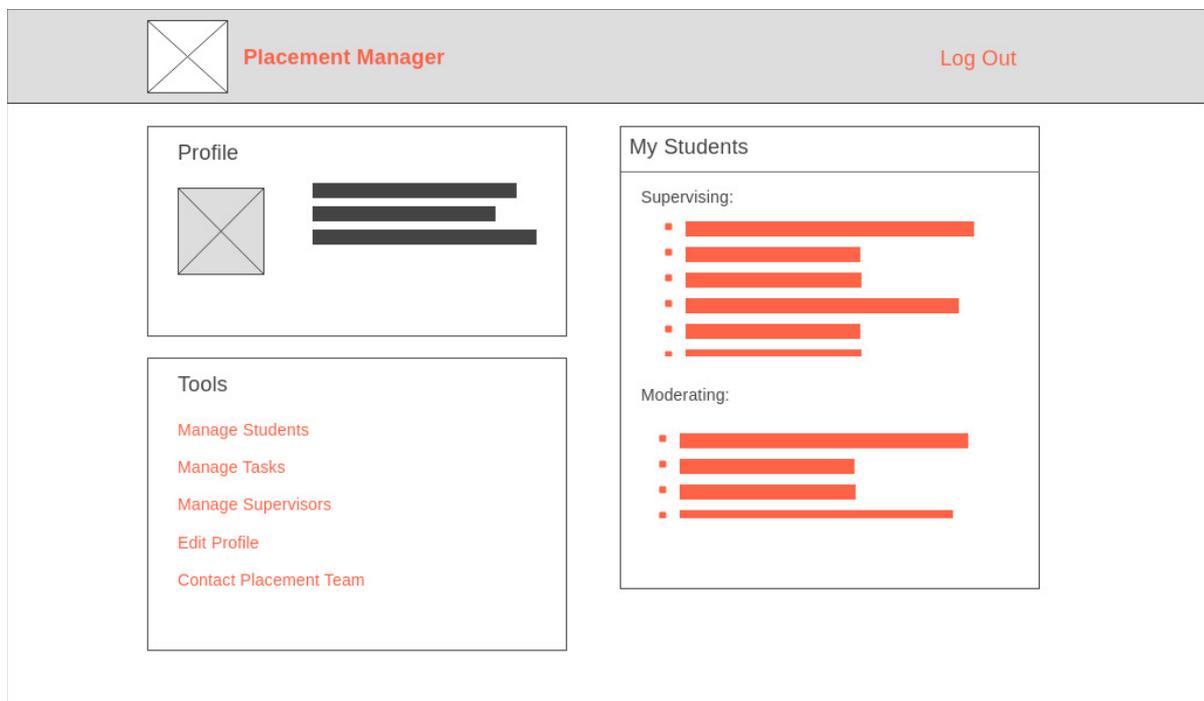
Figure 14: Wireframe - Admin Dashboard

Feature	Description/Justification
Tools	Admin view contains additional links to tools for managing tasks and users that are only available to users with admin privileges. Clinking the link to these tools takes the user to the page.

Manage Students

This page will be where admin users can view and manage students registered within the system. The primary goal of this page is for Placement Team staff to be able to find individual students easily and to assign students a supervisor.
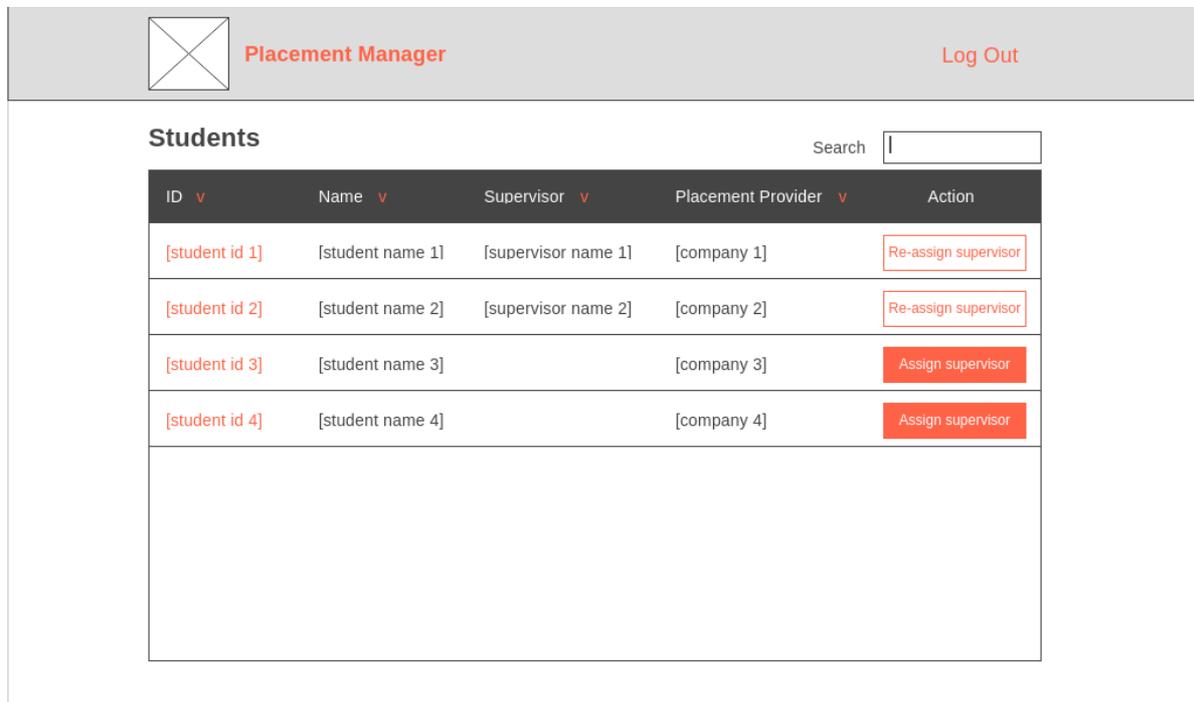
Figure 15: Wireframe - Manage Students

Feature	Description/Justification
Search box	This allows user to input a string to filter students by. Filter should be applied to key data such as Id, Name, supervisor name, or placement provider
Table	• Table should contain key data that helps to identify student – any non-essential data can be found on student details page • Clicking on the row containing a student will take user to student details page for that student – this means more data about the student is available to the user if they desire it
Header row	• Any sortable table columns will contain a symbol in the header that can be clicked to sort the table by this row – allowing the user to organise the data in order to help them find a particular student
Action buttons	• Assign supervisor – 'primary' button, coloured to draw user attention as this is the primary functionality for this page – pressing this button opens the supervisor selection modal

	• Re-assign supervisor – 'secondary' button, can be used to re-assign a student that already has a supervisor

Manage Students – supervisor selection modal

This is the pop-up modal shown when the user clicks 'assign supervisor' button. A modal was chosen as this doesn't navigate away from the page, meaning the user can quickly assign students one at a time without going from page to page.
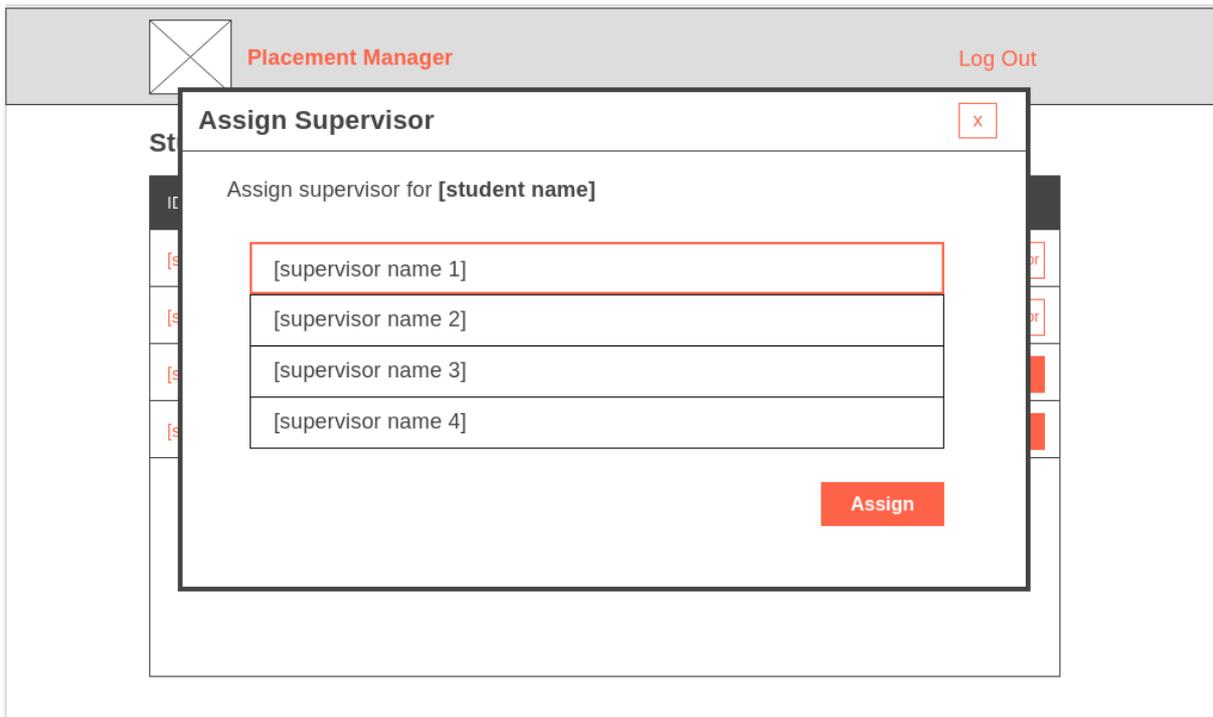
Figure 16: Wireframe - Manage Students - Assign Supervisor Modal

Feature	Description/Justification
Subheading	Provides confirmation to user that they are assigning supervisor for the named student
Supervisor list	List of selectable supervisors to assign to the student. • When clicked, the supervisor will be highlighted in a different colour to show selection to make system status clear to user • If student already has a supervisor, this supervisor should already be selected by default
Assign button	• Button text makes functionality clear to user • 'Primary' button using colour to draw user's attention • Should be disabled until a selection is made

Manage Tasks

This page allows users with admin privileges to manage the tasks for students to complete.
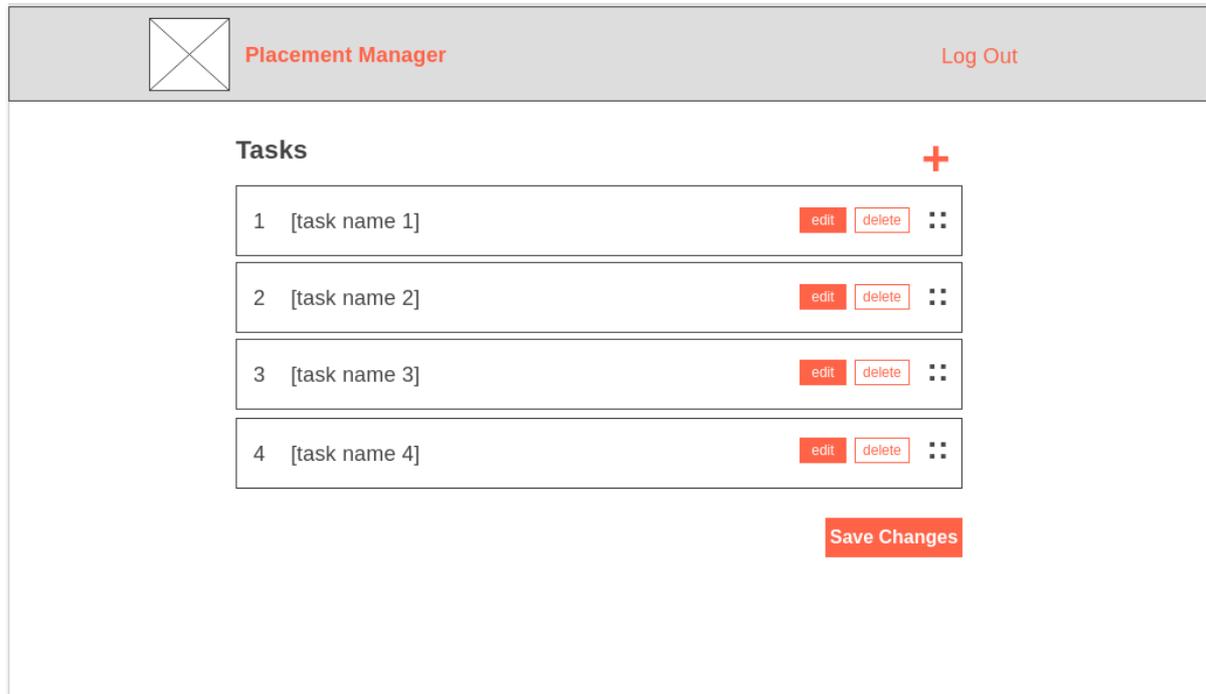
Figure 17: Wireframe - Manage Tasks Page

Feature	Description/Justification
Plus icon	Clicking this icon will take the user to page to create a new task – this is a commonly used symbol so purpose should be clear to user, but on-hover tooltip will also be used in order to make function obvious
Tasks	- Number on far left indicated the position of this task in the order – the idea is that students will complete the tasks in order - Edit button takes user to page where they can edit the details of the task – this means users will not have to make a new task if one or two details need to be changed - Delete button – allows user to delete task, should trigger a modal that asks for confirmation to prevent accidental deletion - Drag dots – these dots can be clicked to drag and drop tasks into desired order – this should be much more intuitive to user than having to adjust the order index manually
Save changes button	- Should be disabled until a change is made 'primary' button colours used

Create Task

This page is designed for the admin to be able to create new tasks for students to be able to complete.

Figure 18: Wireframe - Create Task Page

Feature	Description/Justification
Task Name	Text input, should be a required field for the form
Task Id	Id will be displayed but should be auto-assigned by the system and not editable by the user to maintain uniqueness
Task type	Select box to for user to choose between 'upload' and 'meeting' task types – should be empty by default and should be required field
File Type	Select box input – should be disabled unless task type is specified as 'upload' as file type is not relevant for meeting tasks
Marks available	Number input – should be disabled unless task type is 'upload' and file type is 'assessment' as otherwise this field is not relevant to task
Deadline date	Should allow user to type date for deadline or select from calendar date picker for more intuitive date

	selection – also user should not be able to select dates in the past
Deadline time	Time input – should allow user to type time in 24h clock
Create task button	'primary' button to attract user attention. Button text makes functionality clear to user.

4.1.2   Colour Scheme

Colour is an important tool in being able to improve user experience. The colour scheme used is designed to create a pleasing experience for the user, visually link the system to other Cardiff University Web Resources, and most importantly to maintain a good accessibility standard.

In order to ensure a professional look at feel, the colour design is modelled from the Material Design colour system principles [20] produced by Google. The scheme is centred around a primary and secondary colour and several harmonious variants.
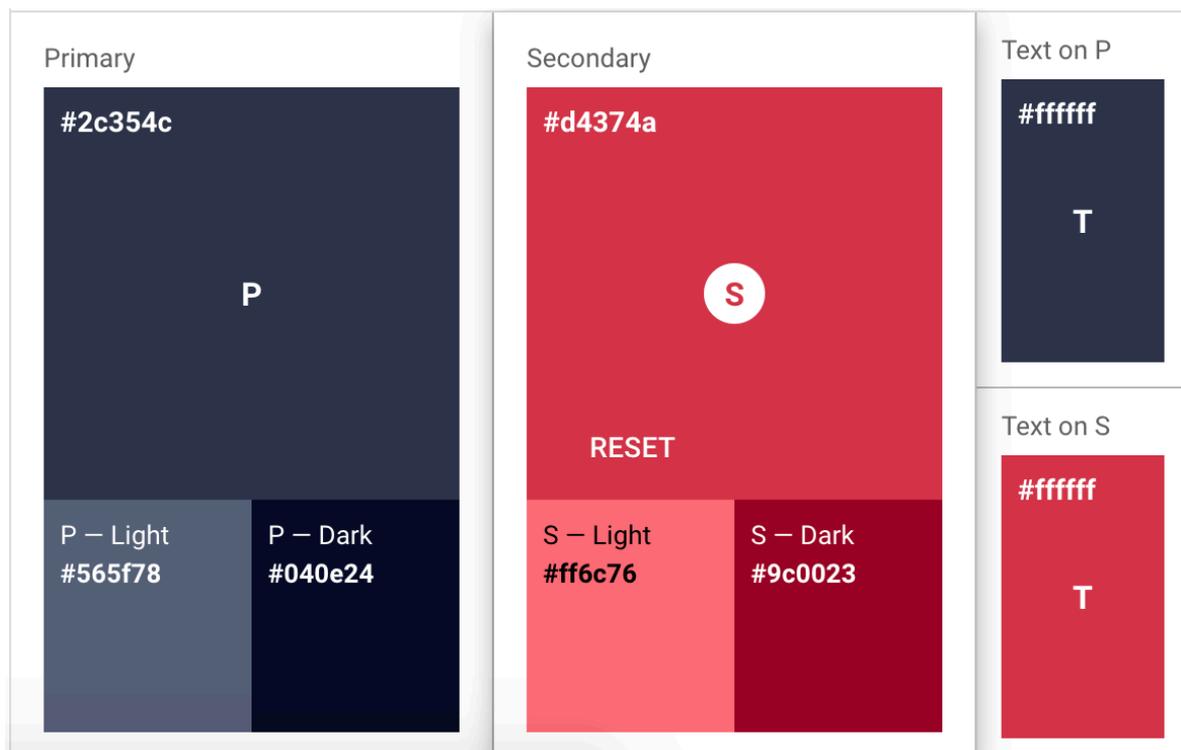
Figure 19: Colour Scheme

To visually tie the application in to the Cardiff University ecosystem, the primary colour (#2b354c) was pulled from the navbar of the Cardiff University Intranet site. This colour is used for the application's primary navigation bar, as well as to accentuate certain features such as tables. Similarly, the secondary colour (#d4374a) is pulled from the Cardiff University intranet's buttons and headings, which is in turn modelled from the Cardiff University logo. Within the system this colour is used to draw the user's attention to

actionable elements such as buttons, as well as being used to highlight 'active' members of lists etc.

The exceptions to this colour scheme will be for:

- Links – will be default blue colour as is widely standardised across the internet, this makes links clear and obvious to the user
- Confirmation notifications, symbols and progress bars will be green to indicate success – industry standard across may web applications
- Warning notifications and symbols will be yellow – industry standard across many web applications
- Error notifications/symbols and dangerous actions such as 'delete' will be coloured in red – universal error/danger colour, industry standard across many web applications

Accessibility

The Material colour tool also provides insight into the text colour/opacity required for all the colour variants in order to maintain accessibility. These standards will be abided by throughout the application and are available here:
https://material.io/resources/color/#!/?view.left=1&view.right=1&primary.color=2c354c&secondary.color= d4374a

4.1.3   Visual Design Mock-up

Visual Designs are mock-ups designed to demonstrate what the app will actually look like to the user by including accurate fonts, copy, colours and structure. These designs are time-consuming to produce, so due to the time constraints of the project a small selection of visual designs were created to be used as a style guide when creating pages.

Dashboard Visual Design

Figure 20: Visual Design - Dashboard

Notable Features

Feature	Description/Justification
Cards	Cards are based off default Bootstrap 4 cards – clean, rounded, minimal colour to distract from functionality
Display picture	Rounded, compliments rounded borders of cards and is pleasing to the eye
Tasks	The format of tasks changed slightly from original wireframes: the tasks are displayed in their order rather than being separated into 'to do' and 'completed'. Tasks are also separated by horizontal rules so each task is more distinct. • Completed tasks – muted text colour as they are less important, green checkmark to visually reinforce completed status • Tasks to be done – default blue url colour to indicate user will be taken to page to complete task

5   Implementation

This section of the report details the process and code involved in implementing particularly key or interesting features of the application - as such, much of the code is not covered in detail.

5.1   System Architecture
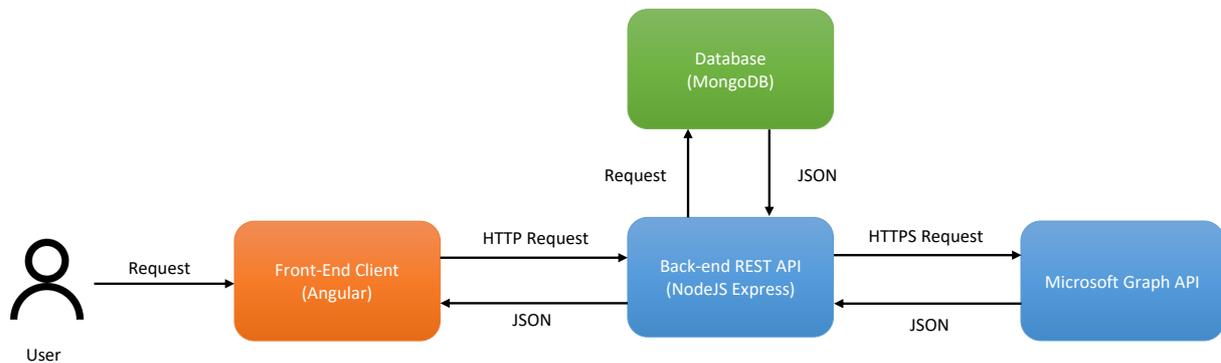
Figure 21: System Architecture Diagram

Figure 21 describes the system architecture and technologies used in the final implementation of the solution. The user interacts with the front-end angular client, the client then makes requests to the back-end REST API. The API (built with NodeJS Express) then makes appropriate requests to the MongoDB database and/or the Microsoft Graph API, it then does any necessary data processing with the JSON data returned and then passes data back to the client in a JSON response. The front-end client will then present this data to the user.

The majority of the data manipulation should be done in the back-end. Any display logic can be done in the front-end, but the back-end should pass data to the client ready to use and data that needs processing from the client should be passed to the API.

5.2   Database

It was decided that the database for the application should contain 4 collections in order to store the different types of data required to implement the requirements:
- Users – store data relating to the system users
- Tasks – store data about the tasks created and managed within the system
- Files – store files uploaded by users of the system
- Meetings – store data relating to any meetings organised by the users

Complications arose when thinking about how these different types of document should relate to each other. The information relating to a task must be available globally to all users, however each user also needs to be able to have data that relates to their own 'instance' of the task, such as whether they have completed that task or not, and their mark for that task.

At first, it was conceived that each task document could contain an array of nested documents describing user submissions/data relating to that task. However, if there were a large number of users then the task documents would become too large and this may affect retrieval time.

In the end, it was decided that each user document would contain an array of id's associated with the tasks they have completed. If further information about the task was required by the client, it could then be retrieved based on this task id. Similarly, files and meetings will have an associated task Id, meaning that they can be retrieved based on the task they are associated with.

It could also be argued that meetings and files documents should be embedded in the user document, as they are closely tied to the user – meaning all the documents and meetings related to a user would be sent when the user's data was requested. This would mean that a separate request would not have to be made to the API in order to retrieve a user's documents/meetings when they wanted to view them. This was decided against, as if the functionality was to be added to view all the files or meetings currently stored by the system then it would require a laborious process of querying each student record individually to retrieve their meetings and documents.

The three user types are implemented through the userType and accessLevel fields of the user documents. As the Placement Team should be able to act as supervisors, it was decided that Placement Team staff would use supervisor accounts with a set of extended privileges. To implement this there are two values for userType: 'student' and 'supervisor' – Placement Team staff accounts are those of userType 'supervisor' where accessLevel.isAdmin has the value true.

The schemas for each collection are managed by the Mongoose object data modelling tool in the back-end API. By defining the schemas in mongoose, it means that the schemas are formatted in a way that reflects the JavaScript objects that will be used by the front-end client when displaying the data.

Collection	users
| Schema | ```js
const UserSchema = mongoose.Schema({
    universityId: {
        type: String,
        required: true,
        unique: true
    },
    name: {
        type: String,
        required: true
    },
    email: {
        type: String
    },
    userType: {
        type: String,
        required: true
    },
    tasksCompleted: [String],
    studentData: {
        supervisorName: String,
        supervisorId: String,
        workplaceSupervisor: String,
        placementProvider: String,
        startDate: Date
    },
    accessLevel: {
        isAdmin: {
            type: Boolean,
            required: true
        }
    },
    feedback: {
        type: Array
    }
});
``` |
| Example | ```
_id: ObjectId("5e8159039296373b4d170bf5")
universityId: "C1525379"
name: "Matthew Trimby"
email: "TrimbyM@cardiff.ac.uk"
studentData: Object
    supervisorName: "Carl Jones"
    placementProvider: "BT"
    workplaceSupervisor: "Clare Lewis"
    supervisorId: "S0003"
    startDate: 2020-04-20T21:00:00.000+00:00
tasksCompleted: Array
    0: "1"
    1: "2"
    2: "36"
    3: "3"
    4: "05cc477a-1b95-1a1d-11ad-1f6236e56f9e"
    5: "4"
    6: "3a387c6b-42e9-c032-f856-a98cd2be5b04"
    7: "bb7284fb-5a59-bf22-07d6-2eafd7c56892"
__v: 15
accessLevel: Object
    isAdmin: true
userType: "student"
feedback: Array
    0: Object
        taskId: 2
        mark: 99
        comments: "Keep up the great work!"
    1: Object
        taskId: 3
        mark: 39
        comments: "You need to add more references to defined practices"
``` |

Collection	tasks
| Schema | ```js
const TaskSchema = mongoose.Schema(
    {
        taskId: {
            type: String,
            required: true,
            unique: true
        },
        name: {
            type: String,
            required: true,
        },
        type: {
            type: String,
            required: true
        },
        uploadInfo: {
            uploadType: String,
            marksAvailable: Number
        },
        displayName: {
            type: String,
            required: true
        },
        description: {
            type: String,
            required: true
        },
        dueDateTime: {
            type: Date,
            required: true
        },
        orderIndex: {
            type: Number
        },
        isPublished: {
            type: Boolean,
            required: true
        }
    }
``` |
| Example | ```
_id: ObjectId("5ed54125599281f62d7498c6")
taskId: "d2273e15-ecdc-34cf-4150-0ab4de8c6522"
name: "upload-final-report"
type: "upload"
∨ uploadInfo: Object
    uploadType: "assessment"
    marksAvailable: 100
displayName: "Upload Final Report"
description: "Upload your final report for marking."
dueDateTime: 2021-05-31T08:00:00.000+00:00
isPublished: true
__v: 0
orderIndex: 12
``` |

Collection	Files
| Schema | ```js
const FileSchema = mongoose.Schema(
    {
        filename: {
            type: String,
            required: true,
        },
        fileId: {
            type: String,
            required: true,
        },
        taskId: {
            type: String,
            required: true
        },
        universityId: {
            type: String,
            required: true
        },
        data: {
            type: String,
            required: true,
        },
        fileType: {
            type: String,
            required: true
        },
        comments: {
            type: String
        },
        date: {
            type: Date,
            required: true
        }
    }
);
``` |
| Example | ```
_id: ObjectId("5e8dd93e44df236c2f85fc3e")
filename: "C1525379-Report.pdf"
fileId: "edkefgyyc"
fileType: "application/pdf"
taskId: "4"
universityId: "C1525379"
data: "JVBERi0xLjMKJCTl8uXrp/Og0MTGCjQgMCBvYmoKPDwgL0xlbmd0aCA1IDAgUiAvRmlsdG..."
comments: "Report"
date: 2020-04-08T14:01:34.252+00:00
__v: 0
``` |

Collection	Meetings
| Schema | ```js
const MeetingSchema = mongoose.Schema(
    {
        name: {
            type: String,
            required: true
        },
        taskId: {
            type: String,
            required: true
        },
        studentId: {
            type: String,
            required: true
        },
        supervisorId: {
            type: String,
            required: true
        },
        location: {
            type: String
        },
        time: {
            type: Date,
            required: true
        },
        approved: {
            type: Boolean,
            required: true
        }
    }
);
``` |

Example	
	          _id: ObjectId("5ed546c59322f7f68e5bbb10")
	          name: "First Supervisor Visit"
	          taskId: "4"
	          studentId: "C1525379"
	          supervisorId: "S0003"
	          time: 2020-10-31T15:00:00.000+00:00
	          location: "BT Data Centre, Cardiff"
	          approved: true
	          __v: 0

5.3　Front-end

5.3.1　Coding principles

As the project is intended to be used by the School of Computer Science, it was important to ensure good coding standards are maintained throughout the code for the purpose of extendibility and maintainability.

The project was generated using the Angular CLI, meaning the file structure of the code conforms to Angular standard practice. Added to this, wherever possible, the code endeavours to conform to the Angular Style Guide [21], a set of coding principles created by the Google Angular team.

Added to this, within the IDE used to build this application I opted to install TSLint, a third-party linter intended to help maintain standard coding style in TypeScript. Further, I have endeavoured to include comments in the code in particularly hard to understand areas, and use intuitive variable names for readability.

Finally, all code endeavours to conform to the Single Responsibility Principle (SRP) [22] in order to maintain code modularity for easier extendibility and maintainability by myself and other developers.

5.3.2　Page Flow

Figure 22: Page Flow Diagram

Figure 22 shows an overview of how the user will interact with various pages and features of the application. Paths shown in orange represent paths that only users with admin privileges will be able to access such as managing tasks and students. Green lines show the page flow for Student users, where they can view and complete the required tasks. Blue lines indicate paths taken by Supervisor users that do not require admin privileges, and red lines indicate paths in the application available to all users. It should also be noted that on pages such as the student details page, students will be able to view data but not able to perform some supervisor/admin actions such as marking submissions or approving meetings.

This page flow is designed to try and create a user experience that is similar for all user types, and where some functionality can be shared by all users to reduce code duplication.

5.3.3   Log in

In the original requirements and designs, it was assumed the user would sign in using their designated Cardiff University credentials (username/password). After conferring with the IT department in the School, it was decided that this would not be possible for the project for security reasons.

Instead, the application uses the Microsoft Identity Platform to sign in and authenticate users via their Cardiff University Office365 account. After some research, I discovered the Microsoft Authentication Library (MSAL)– a library allowing client-side applications to easily authenticate Microsoft work and school accounts. Fortunately, Microsoft maintain a MSAL package for Angular, and I was able to use this to set up authentication in the application.

By setting up and configuring the application in Microsoft Azure and installing this package, a service called MsalService is made available to the application, which exposes methods for log in and log out tasks among other things. The component login.component.ts contains the logic for calling these methods.

```
export class LoginComponent {

  constructor(private router: Router, private authService: MsalService, ) {
  }

  // function called when user clicks log in button
  login() {
    // check if user is on Internet Explorer
    const isIE = window.navigator.userAgent.indexOf('MSIE ') > -1 || window.navigator.userAgent.indexOf('Trident/') > -1;

    if (isIE) {
      // if on internet explorer, log in with redirect
      this.authService.loginRedirect();
      this.loginSuccess();
    } else {
      // if not IE, use popup - on success call loginSuccess function
      this.authService.loginPopup().then(() => {
        this.loginSuccess();
      });
    }
  }

  loginSuccess() {
    // navigate to dashboard on login
    this.router.navigate( commands: ['/dashboard']);
  }

}
```

Code Snippet 1: login.component.ts

When the log in button is clicked by the user, the login method is called. This method then calls the appropriate methods in the MsalService, which either redirects the page or brings up a pop-up window where the user can log in with their Microsoft Office 365 account. If this authentication is successful, then the application will go to the dashboard page.

5.3.4   Dashboard

Figure 23: Application Screenshot - Dashboard - Student View

The dashboard component is rendered slightly differently based on the user, to handle this behaviour the dashboard is made up of other custom Angular components which are displayed or hidden based on the user type.

```html
<!-- navbar -->
<app-navbar></app-navbar>

<!-- spinner - shown while loading data -->
<div *ngIf="loading" class="spinner-container">
  <mat-spinner></mat-spinner>
</div>

<div *ngIf="!loading" class="container">
  <!-- notification/error banner -->
  <div *ngIf="errorMessage" class="alert alert-danger topError" [innerHTML]="errorMessage | translate"> </div>

  <div class="row">
    <!-- first column -->
    <div class="col-md-6">
      <br/>
      <app-dashboard-profile [user]="user"></app-dashboard-profile>
      <br/>
      <app-dashboard-tools [user]="user"></app-dashboard-tools>
    </div>
    <!-- second column-->
    <div class="col-md-6">
      <br/>
      <app-dashboard-my-tasks *ngIf="user.userType === 'student'" [user]="user"></app-dashboard-my-tasks>
      <br>
      <app-dashboard-my-students *ngIf="user.userType === 'supervisor'" [user]="user"></app-dashboard-my-students>
    </div>
  </div>
  <br>
</div>
```

Code Snippet 2: dashboard.component.html

If user.userType is student then tasks will be displayed, if user.userType is supervisor then the my-students card will be displayed.

As the dashboard screen is the home page for the application, it is here we handle retrieving and saving the current users data. This functionality is handled in dashboard.component.ts.

As some of the data we retrieve will be coming from the Microsoft Graph API, we use the MSAL service again (see log in), this time to get an authentication token we can use to call the Microsoft API.

```
getCurrentUser() {
  // get user data from user service
  this.userService.getCurrentUser(this.userToken)
    .subscribe( next: data => {
      // set user data
      this.user = data;
      // set loading to false
      this.loading = false;
    },
      error: (error: AppError) => {
        // if user not found, redirect to error page
        if (error instanceof NotFoundError) {
          this.router.navigate( commands: ['error'], extras: { queryParams: { errorCode: 'userNotFound' }});
        } else if (error instanceof InvalidMsalTokenError) {
          // if token retrieved does not have required permissions, we need to explicitly ask for users consent for app to get their data
          this.getUserConsent();
        } else {
          throw error;
        }
    });
}

getUserToken() {
  // get token to use to read user data from Microsoft Graph Authentication from MsalService
  this.authService.acquireTokenSilent( request: {scopes: ['user.read']})
    .then(response => {
      this.userToken = response.accessToken;
      this.getCurrentUser();
    });
}

getUserConsent() {
  this.authService.acquireTokenPopup( request: {scopes: ['user.read']})
    .then( response => {
      this.userToken = response.accessToken;
      this.getCurrentUser();
    }).catch( err => {
      if (err instanceof ClientAuthError) {
        // displays error message to user informing them they need to disable pop-ups to grant permissions
        this.errorMessage = 'dashboard.errors.needPermissions';
      }
    });
}
```

Code Snippet 3: dashboard.component.ts

The getUserToken method is called on the initialisation of the dashboard component. This method calls the MsalService to acquire an authorization token in the background. When the token has been retrieved, the getCurrentUser method is called which calls the userService – if data is successfully returned then the user object is set. This data object can be passed to the other components in the dashboard to be used.

This method worked well for my own student account, as my account set up the Microsoft API integration through the Cardiff University Azure AD platform. However, when I tested log in for other users, an error occurred as their access tokens did not have the correct permissions. To solve this issue, if this error is encountered then the aquireTokenPopup method of the msalService is called – this method uses a pop-up window to ask for the user's consent for the application to read their Microsoft data, and acquires a token with the correct permissions. These permissions are stored in the users Microsoft Office365 account, so the user will only have to grant permissions once.

When the userService.getCurrentUser method is called by the dashboard, the user's data is retrieved using a get request to the backend.

```
getCurrentUser(token: string) {
  return this.http.get( url: this.url + '/me',  options: {
    headers: new HttpHeaders()
      .set('Authorization', token)
  })
    .pipe(
      map( project res => {
        localStorage.setItem('currentUser', JSON.stringify(res));
        this.currentUserSubject.next(res as User);
        return res as User;
      }),
      catchError(this.handleError)
    );
}
```

Code Snippet 4: user.service.ts - getCurrentUser()

The getCurrentUser method in user.service.ts performs a get request to the backend API's URL + '/users/me' – how this request is handled by the backend is detailed in the Back-end section of the implementation. This method sets the user's token as the authorization header and saves the JSON data response to local storage for quick access from other parts of the application while the user is logged in. This is cleared when the user logs out.

5.3.5   File Upload
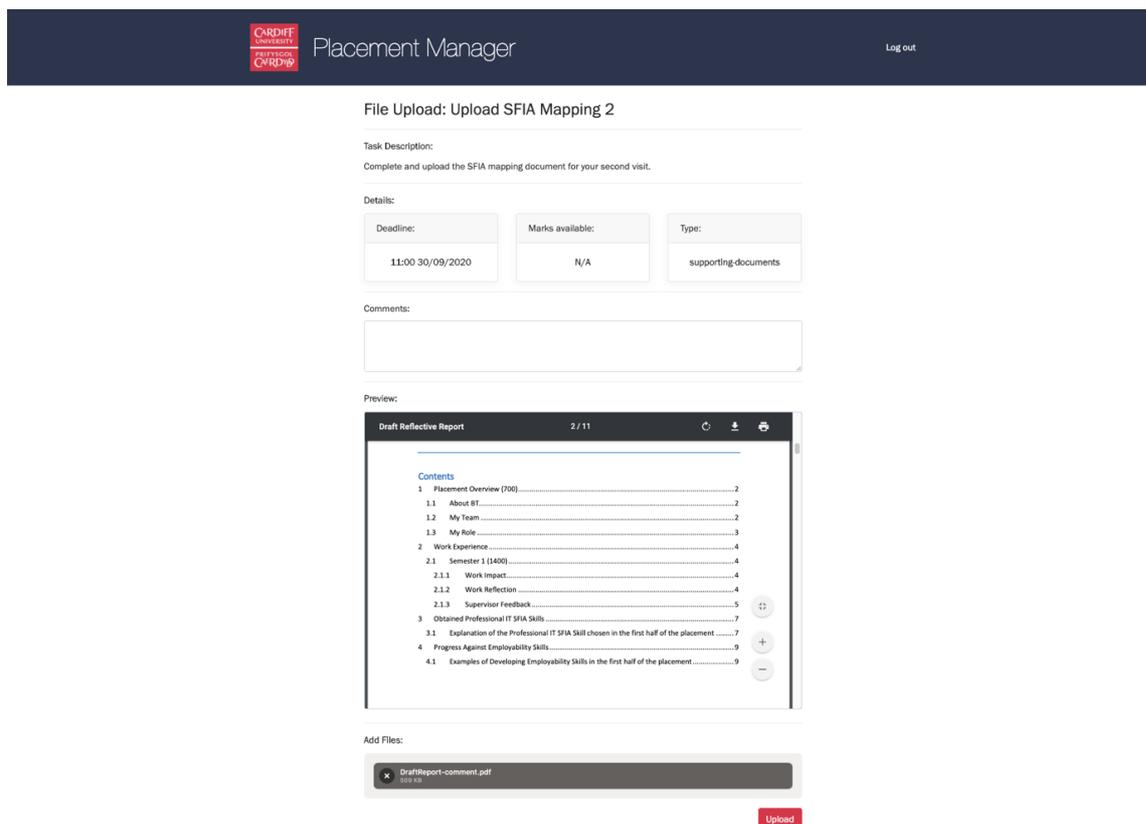
Figure 24: Application Screenshot - Upload File

One of the most key requirements of the project is to be able to facilitate file upload. As file upload is such a key feature, I wanted it to feel slick and intuitive, so instead of using the

default HTML file input I opted to use the filepond npm package. This package allows the user to drag and drop files, has sleek animations while the file is processed, and facilitates file validation.

The filepond component has built in lifecycle hooks that can be used to call methods in the parent component (in this case the file upload component) on events such as when the filepond element is initialised and when files are added or removed from the pond.

```typescript
// method called when file added to the filepond
pondHandleAddFile(event) {
  // build file object
  const fileObject = this.buildFileObject(event.file);
  // set pdfData for preview
  this.previewType = event.file.fileType;
  if (this.previewType === 'application/pdf') {
    this.pdfData = fileObject.data;
  }

  // set the value of the 'files' form control to file object
  this.uploadForm.get('files').setValue(fileObject);
}

// creates file data object
buildFileObject(file) {
  return {
    fileId: file.id,
    filename: file.filename,
    fileType: file.fileType,
    taskId: this.task.taskId,
    universityId: this.userId,
    data: file.getFileEncodeBase64String(),
    metadata: file.getMetadata()
  };
}
```

Code Snippet 5: file-upload.component.ts – on add file

When a file is added to the pond, the buildFileObject method is called that builds a file object which can be later posted to the database. The actual file data is encoded into a base 64 string and stored along with the other data such as the filename and id – this allows the files to be easily stored in the MongoDB database. If the file is a pdf, then the file data is set to the pdfData variable in the component, this variable is then read by the pdf previewer component to show the user. For now, the preview window only shows a preview for pdf files, as browsers don't natively support preview for docx files.

```
markTaskCompleted() {
  this.userService.markTaskCompleteForUser(this.userId, this.task.taskId)
    .subscribe( next response => {
      console.log(response);
    });
}
// set preview pdfData to null if file removed from filepond
onFileRemove() {
  this.pdfData = null;
}

// method called when user clicks upload button
onFileUpload() {
  // show user spinner while file being uploaded
  this.showSpinner = true;

  this.filesService.uploadFile(this.uploadForm.value)
    .subscribe( next (response: any) => {
      // mark task as completed
      this.markTaskCompleted();
      // file uploaded so stop showing spinner
      this.showSpinner = false;
      const uploadId = response._id;
      // set navigation url based on userId, taskId and file uploadId
      const navigationUrl = '/tasks/submitted/' + this.userId + '/' + this.task.taskId + '/' + this.task.name;
      // navigate to submissions page for user to view submissions for this task
      this.router.navigate( commands: [navigationUrl], extras: {queryParams : { idSelected: uploadId, isUploadRedirect: true}});
    });
}
```

Code Snippet 6: file-upload.component.ts - on upload

When the user clicks the upload button, the onFileUpload method is called. This method calls the filesService which executes a simple post request with the file object data to the back-end API which then stores it in the database.

If this is successful, then then the markTaskCompleted method is called which calls the markTaskCompletedForUser method in the userService. This method sends an UPDATE request to the backend, which then appends the taskId to the array of completed tasks for this user in the databsae.

The application then navigates to the submissions page so the user can view their submissions for this task. The file's id is passed as a query parameter, this will be used by the preview page to automatically preview the file just submitted by the user.

5.3.6   View/Assign students

Figure 25: Application Screenshot - Manage Students

Features of the manage students table include being able to sort by header, filter by search string and display a desired number of data entries per page (pagination). Added to this, rows are highlighted when hovered over, and the action button to assign/re-assign supervisor is only shown when hovering over the row in order decrease the likelihood of accidentally assigning a supervisor to the wrong student.

The implementation of the table utilises the MatTable and MatDataSource components provided in the Angular Material [23] package, an open source library of UI elements created and maintained by Google's Angular development team. This simplified the process of creating a table with complex functionality such as server-side pagination and filtering to enhance the user experience, as these libraries are well documented.

This table, and much of the other code within the front-end utilises RxJS [24], an open-source library for implementing asynchronous behaviour in JavaScript. RxJS observables work similarly to JavaScript promises, in that a function returns an observable which will then emit the desired object once some asynchronous behaviour has been performed.

The table subscribes to an RxJS Observable exposed by the MatDataSource, this data source acts as an interface between the table and the services that compile the data. The Observable emits the ordered list of student data objects as it changes based on user behaviour, and the table displays it.

```
      // function called after view initialised (all children paginator, sort etc are loaded)
ngAfterViewInit() {

  this.table.dataSource = this.dataSource;
  this.sort.sortChange.subscribe( generatorOrNext: () => this.paginator.pageIndex = 0);

  // function executed whenever user types in filter input box
  fromEvent(this.input.nativeElement,  eventName: 'keyup')
    .pipe(
      // limits calls to load students to every 200 milliseconds, prevents unnecessary calls while typing
      debounceTime( dueTime: 200),
      // only calls if filter input has changed
      distinctUntilChanged(),
      tap( next: () => {
        // reset page index to 0 for paginator
        this.paginator.pageIndex = 0;
        // load student data from datasource
        this.loadStudentsPage();
      })
    )
    .subscribe();

  // merge function waits for observables to emit from paginator and sort, then emits them concurrently
  // in effect this function waits for a change in either paginator or sort (or both) and calls the load students function
  merge(this.paginator.page, this.sort.sortChange) Observable<PageEvent|Sort>
    .pipe(
      tap( next: () => this.loadStudentsPage())
    ) Observable<unknown>
    .subscribe();
}
```

Code Snippet 7: student-table.component.ts

For filtering, sorting, and pagination the table listens for user input on the filter box, sort buttons and paginator – when something changes, the loadStudents method of the dataSource is called, which in turn calls the back-end API with query parameters filter, pageIndex, pageSize, sortBy, sortOrder and userType. These parameters are processed by the API and an ordered list of students is returned. Details of the API processing are documented in the Back-end section of the implementation.

Assigning a supervisor
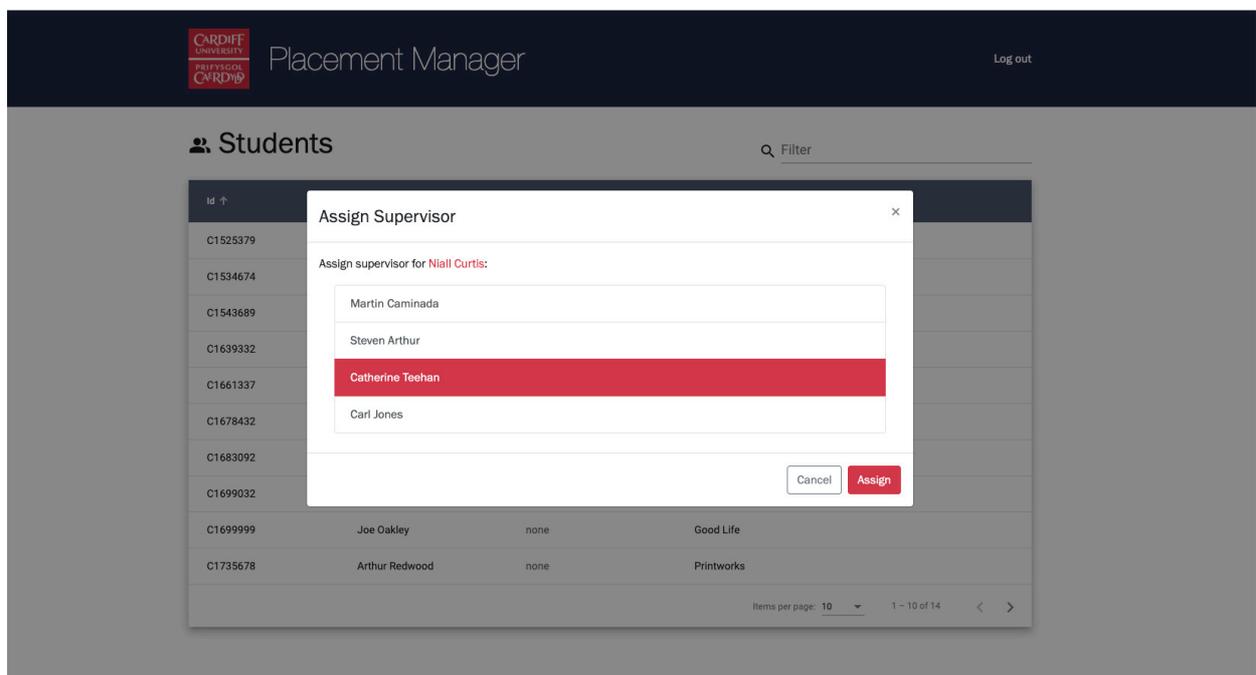
Figure 26: Application Screenshot - Assign Supervisor

Assigning a supervisor is facilitated through a pop-up modal. The modal itself is built using the ngBoostrap modal component. ngBootstrap [25] is a library of Angular UI components designed and built by Bootstrap for Angular.

```
// function called when 'assign' or 'reassign' button clicked, arguments are passed from the list item where the button is clicked
assignSupervisor(student, isReassign: boolean) {
  // open ngBootstrap modal for assigning supervisors and set input parameters
  const modalRef = this.modalService.open(AssignSupervisorModalComponent, options: {size: 'lg', windowClass: 'modal-holder', centered: true});
  modalRef.componentInstance.student = student;
  modalRef.componentInstance.isReassign = isReassign;

  // after modal closed
  modalRef.result.then( selectedSupervisor => {
    // if assign button was clicked, set supervisor name and id for student then push data to backend
    student.studentData.supervisorName = selectedSupervisor.name;
    student.studentData.supervisorId = selectedSupervisor.universityId;
    this.userService.updateUser(student).
      subscribe( next: response => {
        console.log(response);
    });
  }, () => {
    // modal dismissed, do nothing
  });
}
```

Code Snippet 8: student-table.component.ts - assignSupervisor

When the assign/re-assign button is clicked, the modalService (provided by ngBootstrap) is called and the modal is initialised. The necessary data is then passed to the modal from the parent StudentTableComponent based on the student selected and the available supervisors retrieved from the userService. Once the supervisor is selected and the assign button is clicked, the supervisor name and id are set for the student, and this data is posted to the database via the userService.

5.4   Back-end

The back-end of the application is a fairly standard REST API – its main function is to facilitate the Create, Read, Update Delete (CRUD) operations on the MongoDB database through the POST, GET, UPDATE, DELETE and PUT HTTP request methods.

5.4.1   Routes

The four main routes of the API reflect the 4 collections in the database:
- '/users'
- '/tasks'
- '/files'
- '/meetings'

Each route encapsulates the CRUD functionality for its respective collection, i.e. the 'users' collection is only queried by the endpoints in the '/users' route. This works well for a fairly basic API such as the one in this project as it clearly sets out where collections can be queried from – for an API with higher complexity it may not be possible to modularise in this way.

To keep things clean, all the endpoints for each route are contained within a file dedicated to that route. Additionally, controllers are used to execute the actual JavaScript functions related to the route – the route files endpoints simply receive a request and call the required function in the controller. This means each endpoint points to a self-descriptive method name, making it much easier for the developer to see an overview of the endpoints and their behaviour.

5.4.2   Retrieving user data

When the user logs in to the application and is taken to the dashboard, the endpoint for getting current user data is called. This endpoint calls the getUserFromToken method of the usersController.

```
getUserFromToken: async (req, res) => {
    try{
        // get Microsoft auth token from request header
        const token = req.get('authorization');
        //get data about user from Microsoft Graph
        const dataFromMsGraph = await msGraphController.getUserFromMsGraph(token);

        //if error getting the data from MS Graph with token, return 401 status request not authorized
        if( dataFromMsGraph.error){
            if(dataFromMsGraph.error.code === 'InvalidAuthenticationToken') {
                res.status(401).send('User not Authorized to access MS Graph data.');
                return;
            }
        }

        const userEmail = dataFromMsGraph.mail;
        //find user in database from unique university email
        await User.findOne({email: userEmail}) Query | void
            .then( resolve: userData => {
                //if user found, get call method to get profile photo then build and send user object as response
                if (userData) {
                    msGraphController.getProfilePhoto(token, userData.universityId).then();
                    res.send(buildUserObject(dataFromMsGraph, userData));
                //if user not found, send 404 not found error status
                } else {
                    res.status(404).send('No user data registered to this account.');
                }
            }) Promise
            .catch(err => {
                res.status(500).send(err);
            });

    }catch (err) {
        res.status(500).send(err);
    }
},
```
Code Snippet 9: usersController.js - getUserFromToken

This method uses the authorisation token received from the Microsoft Authentication Library in the front end to retrieve the data stored about the user in Microsoft's Graph API. If the authorisation token is invalid this is usually because the user hasn't granted permissions to the placement application yet. The API will return a 401 not authorised error which will then be processed by the client to take the necessary steps to get the user's permissions.

If the request to Microsoft Graph is successful, the email address from this data is then used to find the matching user in the placement application's MongoDB database using Mongoose. If the email matches a record in the database, then this data will be combined with the data from MS Graph in the buildUserObject function and then returned as JSON to the front-end client.

If the email does not match a record in the database then the API will return a 404 not found error, this will be processed by the front-end to inform the user no record exists for their account.

5.4.3   Retrieving data for manage students table

The API facilitates server-side filtering, sorting and pagination for the 'Manage Students' table in the front-end client in an attempt to move as much processing as possible away from the front-end. When the table in the client requires a page of data, it sends a request to an endpoint in the API which calls the searchUsersAndReturnPage method.

```
searchUsersAndReturnPage: async (req, res) => {
    try{
        let users;
        const queryParams = req.query;
        //set variables from query parameters
        const filter = queryParams.filter || '',
            pageSize = parseInt(queryParams.pageSize),
            pageIndex = parseInt(queryParams.pageIndex) || 0,
            sortBy = queryParams.sortBy,
            sortOrder = queryParams.sortOrder,
            userType = queryParams.userType;
        //if there is a filter string, query the database for users who's id, name, supervisor name or
        //placement provider match the string
        if (filter) {
            users = await User.aggregate(
                pipeline: [
                    {
                        $match: {
                            $and:[
                                {'userType': userType},
                                {$or: [
                                    {'universityId': {$regex: '.*' + filter + '.*', $options: 'i'}},
                                    {'name': {$regex: '.*' + filter + '.*', $options: 'i'}},
                                    {'studentData.supervisorName': {$regex: '.*' + filter + '.*', $options: 'i'}},
                                    {'studentData.placementProvider': {$regex: '.*' + filter + '.*', $options: 'i'}}
                                ]
                                },
                            ]
                        },
                    },
                ]);
        //if no filter, fetch list of all users matching userType
        } else {
            users = await User.find({'userType': userType});
        }
        //sort users using by sortBy field in ascending or descending order
        sortUsers(users, sortBy, sortOrder);
        //slice the data to the correct page size and position
        const initialPos = pageIndex * pageSize;
        const usersPage = users.slice(initialPos, initialPos + pageSize);
        //send data response
        res.send(usersPage);
    } catch(err) {
        res.status(500).send(err);
    }
},
```

Code Snippet 10: usersController.js - searchUsersAndReturnPage

This method takes the query parameters set in the request by the front-end client and uses them to query the database. If there is a filter string, the database is queried for user records which have a universityId, name, supervisorName or placementProvider that match the string using Mongoose – if not, all records matching the user type are returned.

From there, the records are then sorted by the field name from the sortBy parameter in either ascending or descending order according to the sortOrder. The data can then be sliced according to the page index and size from the table's paginator to return the desired page to the front-end client which then displays the data.

5.5   Deployment

As a proof of concept, it was decided that the application should be deployed on a web server in order to test the application would function when deployed by the university.

The applications front-end and back-end code were deployed on a Linux Ubuntu virtual machine, using NGINX on an Amazon EC2 instance (following an online guide[2]). After some time experimenting with configuring the server, the code was able to be accessed via the IP of the EC2 instance. However, it was discovered that once deployed on the cloud rather than running on a localhost server, the authentication using the Microsoft Identity Platform was no longer working.

After troubleshooting I worked out that this was because the URL for the EC2 instance was not whitelisted for the application in the Cardiff University Azure application configuration. This presented a further problem, in order to be whitelisted, a URL must be using HTTPS rather than HTTP (for security reasons).

In order to navigate this issue, I needed to obtain a security certificate for the URL in order to use HTTPS. Because security certificates can only be registered to domain names, I opted to use my personal domain and routed it to the EC2 instance through the Amazon Route 53 Domain Name Service. I then set up Certbot [26] on the server in order to automatically manage the HTTPS certificate for the domain.

Once this was done, I could then whitelist my personal URL in the Microsoft Azure configurations and the application could be accessed via the domain. It's worth noting that many of the issues encountered here would probably be avoided if the application is deployed on the university servers as they are already set up to use HTTPS.

The deployment of the application proved that the system would be able to be deployed and used by the university. However, the application has not been fully tested whilst deployed on the server – this should be done in order to make sure none of the functionality is adversely affected.

---

[2] Available at: https://jasonwatmore.com/post/2019/12/02/angular-nodejs-on-aws-how-to-deploy-a-mean-stack-app-to-amazon-ec2

6   Results and Evaluation

6.1   Testing the system

Throughout the implementation process, the system has been consistently tested whilst running on a local host, using mocked data that aimed to closely mimic the real data that would populate the system.

One challenge presented in terms of testing was the ability to be able to test a system that requires action from multiple users in order to represent a real-world scenario. Users can only log in to the system using real Cardiff University Microsoft Office 365 credentials and consequently the only test account available to me was my own.

To be able to test functionality for all user types, the data stored in MongoDB associated with my University email address was changed:
- Student user – userType: 'student', accessLevel.isAdmin: false
- Supervisor – userType: 'supervisor', accessLevel.isAdmin: false
- Placement team staff (admin) – userType: 'supervisor', accessLevel.isAdmin: true

This allowed the mocking of the user experience for each type of user. As the system requires interaction with other user data in order for testing, the MongoDB database was also populated with a set of mock users for the primary account to be able to interact with.

Mock data such as tasks, meetings and feedback were also inserted into the database for testing purposes – meaning that functionality such as creating and deleting tasks and files and approving and deleting meetings could be tested.

The main limitation of this testing is that each user's functionality had to be tested independently with mock data on a single account. This means that if and when the application is used by real users, bugs may appear that would not appear during testing with a single user account.

See Appendix A for test cases.

As shown in appendix A, the majority of test cases passed with the application behaving as expected for most scenarios. Further tests should be carried out in order to test a greater range of scenarios and data input.

Initially it was the intention for the project to have a fully functioning suite of automated unit tests. However, due to the time constraints of the project there was not time to fully implement this.

6.2   Assessing application against requirements

Requirement	Requirement Met (Y/N)	Justification/Comments
Must Have		
FR-1: User must be able to log in to the application	**Yes**	User is able to log in to the system through their Microsoft Office 365 credentials
FR-2: Student must be able to view list of tasks to complete	**Yes**	List of published tasks for students to complete is displayed on dashboard page to student users
FR-3: Student must be able to see deadline of tasks they are to complete	**Yes**	When user clicks on a upload/meeting task they are taken to upload file/schedule meeting page that displays the deadline date of task
FR-4: Students must be able to upload files for tasks	**Yes**	Students can upload pdf and docx files for a task via the file upload page
FR-5: Students must be able to see a mark for a piece of work they have submitted	**Yes**	Students can view marks for assessment tasks via the 'feedback' page linked from the dashboard
FR-6: Supervisors must be able to see a list of students they are supervising	**Yes**	Dashboard displays a list of students in the 'My Students' card on the dashboard for supervisor users
FR-7: Supervisors must be able to see and download students file submissions for a task	**Yes**	Supervisors can see students' submissions for tasks by going to the submissions tab for the 'student details' page for a student they are supervising.
FR-8: Supervisor must be able to submit a mark for students work	**Yes**	Supervisors can submit a mark for assessment type upload tasks via the same page described in FR-8
FR-9: Placement Team staff must be able to view all students enrolled in the placement program	**Yes**	Any user with admin privileges can see all students registered with the system via the table in the manage students page
FR-10: Placement Team staff must be able to assign a	**Yes**	Users with admin privileges can assign a supervisor to any student via the manage students page

student to a supervisor		
FR-11: Placement Team Staff must be able to view tasks set for students to complete	**Yes**	Users with admin privileges can view tasks via the manage tasks page
FR-12: Placement Team staff must be able to create new tasks	**Yes**	Users with admin privileges can create new upload/meeting tasks via the create task form accessed through the manage tasks page
FR-13: Placement team staff must be able to delete a task	**Yes**	Users with admin privileges can delete tasks via the delete button for each task on the manage tasks page
Should Have		
FR-14: User should be able to log out of the application	**Yes**	Users can log out of the application from any page via the 'log out' button on the navbar
FR-15: Students should be able to view their previous submissions for a particular task	**Yes**	When a student has submitted a submission for an upload task, if they click on the task from the 'My Tasks' panel in the dashboard they will be taken to the submissions page to view their previous submissions for that task
FR-16: Student should be able to see information about their assigned supervisor	**Yes**	Students can see information about their supervisor on by clicking their supervisors name in the 'profile' section of the dashboard to bring up the supervisor information modal
FR-17: Student should be able to arrange a meeting with their supervisor	**Yes**	Students can schedule a meeting with their supervisor for a particular time/location by clicking on any meeting tasks
FR-18: Students should be able to view profile information about themselves	**Yes**	Students can view details about themselves via the student details page by clicking on the 'view profile' link on the dashboard
FR-19: Student should be able to distinguish tasks they have already completed from those they have not	**Yes**	Tasks already completed on the dashboard are greyed out and a green checkmark is displayed next to them on the dashboard
FR-20: Tasks should be marked as done once they have been completed	**Yes**	When a meeting is scheduled or a file is uploaded, the system automatically adds the task id to the list of array of completed tasks for that student in the database

FR-21: Supervisors should optionally be able to submit feedback comments alongside a mark for a student submission	**Yes**	This functionality is available via the same page as described in FR-8/9 – supervisor can submit textual comments which will be displayed with the mark on the feedback page for students
FR-23: Supervisors should be able to view relevant details about students they are supervising	**Yes**	Information about students can be viewed on the student details page by clicking on the students name in the 'my students' card on the dashboard.
FR-24: Placement Staff should be able to edit the details of current tasks	**Yes**	Users with admin privileges can edit details of a task by clicking the edit icon for the task on the manage tasks page
FR-25: Placement Team staff should be able to change student details	**No**	Due to time constraints, this functionality is not currently available within the application – the only way to edit student data is through an API request or in the database
FR-26: Placement Team staff should also be able to see students they are supervising	**Yes**	Placement team staff accounts are supervisor accounts with extended admin privileges, so they have access to the same features as supervisors
FR-27: Placement Team staff should be able to sort the list of students by field	**Yes**	Users with admin privileges will can sort the table by header on the manage students page
FR-28: Placement Team staff should be able to search for students by key identifiers	**Yes**	Users with admin privileges can search using the table filter box on the manage students page
FR-29: Placement Team Staff should be able to give tasks deadlines	**Yes**	A deadline is added for a task when created on the create task page
Could Have		
FR-30: System could get and display data about users from the Microsoft Office365 API	**Yes**	The getCurrentUser method of the API retrieves data about the student from the Microsoft Graph API and combines it with data about the student in from MongoDB database
FR-31: Student could receive an email	**No**	Due to time constraints, automated email functionality was not implemented

when a deadline is approaching		
FR-32: Student could be able to edit their own profile information	**No**	Same as FR-25
FR-33: Students could see a progress bar which shows them the percentage of tasks they have completed	**Yes**	Progress bar is shown in the 'My tasks' card on the dashboard for student users
FR-34: Placement Team Staff could create scheduled emails to send to all students at certain points during their placement	**No**	Same as FR-31
FR-35: Placement Team Staff could be able to save a task as a draft rather than publish it to students	**Yes**	Users with admin privileges can drag and drop tasks between the 'published' and 'drafts' groups on the manage tasks page
NFR-1: System should work on all main browsers	**Yes**	System has been tested on Chrome, Firefox, Edge
NFR-2: System should load user data within a reasonable amount of time	**Yes**	Assuming the user has a reasonable internet, connection speed data is loaded within a reasonable amount of time
NFR-3: System should be intuitive and easy to use	**Partial**	System was designed in line with Nielsen's usability heuristics, but has not been user tested to conform user experience
NFR-4: System should not allow users to change other user's data without correct permissions	**Partial**	Users without admin privileges do not have access to pages that allow them to edit user data – but the API could feasibly be called by an unauthorised user
NFR-5: System should update data in database on action	**Yes**	Data is always posted to the database, not stored locally in browser

As shown in the figure above, the project successfully implemented 85% of the requirements set out at the beginning, fulfilling 100% of the 'must have' requirements and 93% of the 'should have' requirements. In this way, the project can very much be considered successful.

Only half of the 'could have' features were fully realised in the final system, but this is acceptable as these requirements were not key to the applications functionality and it could be argued that aiming to achieve all requirements to a high quality was over-ambitious.

7 Future Work

Automated Emails

It was the original intention of the project to incorporate an automated email function into the application – however, due to time constraints this was not accomplished. Automated emails would be sent as deadline reminders, submission verification, and notifications about student activity for supervisors prompting action.

One method of implementing this within the system would be through the Nodemailer [27] package for NodeJS. This package facilitates the automated sending of emails in NodeJS web applications, allowing the system to send emails when certain endpoints are called. Further to this, Nodemailer could be integrated with the Node Cron [28] in order to facilitate the automated sending of scheduled emails based on date/time.

Extrapolating from this, the system could allow Placement Team staff to write custom email templates in the application and set schedules to send them according to variables such as the users start date or task deadline dates. This feature would undoubtedly save the Placement Team staff copious amounts of time, as well as improving the user experience for students and supervisors.

Moderators and 3-tier marking

In its current state the application allows Supervisors and Admins to submit marks and feedback for students work. However, during the course of the project the desire was expressed for it to facilitate the full marking process for final reports: marks are submitted by the supervisor, moderator and then verified and confirmed by a final user.

The current system goes some way to facilitate the moderating of students – the supervisor dashboard already contains a section for users to view the students they are moderating, and the moderator name and id are already written into the User schema for the database. To fully implement the moderation process, much of the experience can be the same as for supervising, and marks could also be stored in the database in the same way. The main work to be done would be to enable the 3-tier marking system by enabling an admin to confirm a final mark, which would ideally require a new component. This component could perhaps be displayed to admin users on the dashboard in a new 'Confirm Marks' which would show a list of students that had received marks from both their supervisor and moderator and were awaiting confirmation.

Migrating the system to university servers

If the application is to be used by the School of Computer Science, then it should ideally be hosted on the school's web servers. In theory this should be a fairly simple task; the application is currently deployed on Amazon EC2 on a simple Linux virtual machine (VM), so the school should be able to easily deploy a similar VM on their own servers to host the application. The only thing that would need to be changed in the front and back-end code would be the URLs for HTTPS requests.

The application would also need to be linked to the Schools MongoDB instance rather than my personal MongoDB Atlas cluster. This migration should be simple because I have not used any real data, so no data needs to be migrated, the back-end API should simply be pointed to the schools MongoDB instance which would then be populated with the real data.

One further consideration for migrating the system could be the applications dependencies on third-party packages. There are potential security risks of using third-party packages in a system that handles sensitive student data, and these would need to be assessed and the code possibly adapted.

Better error handling

Some error handling is already implemented within the system, but it could be much more robust. Future work in this regard would be updating the API to send more meaningful HTTP statuses on error, and for the front-end client to handle and interpret these to give the user meaningful error messages.

Security

Some security measures are already in place within the application – a user cannot access any of the front-end client pages without a valid authorisation token from signing into their Microsoft account. However, there is currently little barrier to stop unauthorised requests to the API, other than only accepting responses from whitelisted URLs. In theory this could be circumvented by malicious users.

In order to increase the security of the API, authorization for all requests could be implemented using JSON Web Tokens (JWT) [29], an open source, industry-standard method of authorised communication between web applications. Implementing this authorisation technique would mean that only users who had been issued a token by the API were able to make requests and change data.

Added to this, currently the API endpoints that should only be available to admins (such as POSTing tasks) are not limited based on user credentials. JWT tokens allow user-based validation, meaning tokens with different levels of permissions could be given out based on the user.

Comprehensive Unit Testing

In its current state, unit test coverage of the application is much poorer than intended at the project's outset. More comprehensive unit tests would substantially increase the maintainability of the application by future developers, and undoubtedly uncover and fix unforeseen bugs that currently exist within the system.

Ideally, every function in every file should be tested in order to ensure it behaves as expected, and to ensure that any breaking changes are acknowledged by the developer.

Welsh Language Translation

As a Welsh institution, Cardiff University requires all of its resources to be available in welsh as well as English. In order to facilitate a fairly painless translation process, I opted to use localisation to store the majority of text displayed to the user as key-value pairs in the locale-en.json file within the application's front-end client.

This means that all that needs to be done to facilitate Welsh translation is for a matching file to be created with Welsh language value alternatives for each key. An option would then need to be added for the user to select which translation they would like to see.

Employer Data

A useful future piece of functionality that may benefit users would be storing data about employers within the system. Many employers take students from the university every year, and if documents and data such as risk assessments could be stored for each employer then it may mean less repetition of administrative processes for the Placement Team staff.

Database optimisation

The database currently has no problems returning data whilst populated with only a small amount of test data. However, as the application scales it is feasible to imagine that database performance will decrease. In order to combat this, indexes could be added to the database for regularly queried fields, and the database architecture may need to be reviewed.

Batch mark release

Another feature that could be useful for the Placement Team would be the ability to release all marks for a certain assessment at the same time. In theory this should not be difficult to do, perhaps including a 'released' Boolean with each piece of feedback which is initially false until the user chooses to release all marks.

Document Download/Task Attachments

In order to further add to the usefulness of the application for students, the capability could be added to the system to add attached documents to tasks. This would mean that the Placement Team could attach a document to be filled in, the student would then be able to view and download this document from the task page to be filled in and uploaded.

8 Conclusions

The end result of the project is a modern, interactive full-stack user application, which allows users of different types to fulfil the required tasks. During the design and implementation phases of the project, effort was made to produce a high standard of code quality that will allow future developers to easily maintain and extend the project.

However, some of the objectives set out at the beginning of the project were not achieved. Unit test coverage and error handling in the application is not to the standard initially desired, meaning the application will be more prone to unforeseen bugs if deployed and used by the school.

Added to this, due to time constraints exacerbated by the COVID-19 pandemic peak during implementation, it has not been possible to user test the application. It would be desirable to have a period of user testing and improvement before the system was used.

Overall, I would consider this a successful project, only requiring a small amount of future work in order to be deployed as a functioning asset for the School of Computer Science. This project will be combined with the work of two other projects in order to create a system that can be used by students and supervisors encompassing the full placement journey - from the first day of searching, to the last day of work.

9 Reflection

I have learned an incredible amount from this project. The challenge of building a fully functioning application from scratch has tested my abilities as a software engineer. From initially having never created a full-stack web application, or worked with any of the technologies, I now feel confident in my ability to create further, better web applications.

From this project I have learned that it may be preferable to implement a narrower range of features and focus on highest code standard possible. In future projects I would focus more time at the beginning of the project on learning and put a heavier emphasis on testing (both user and automated). I believe this would have resulted in an application with fewer features, but I would have more confidence in its robustness and the reliability of the system going forwards.

Further, it would have been preferable to collaborate more with students developing the other aspects of the placement management system, perhaps more closely aligning our choice of technologies in order to save time and effort for developers further down the line. I have learned not to lose track of the context of the project, and will try and think about this more in future projects.

Additionally, I feel the design process should have been more structured and in line with more defined design principles. Although I am happy with the end look and feel of the application, some of the design was done on what 'felt right' to me. Whilst instinct can be a great indicator in UX/UI design, I should have used more formal processes during a software development process of this scale.

Finally, in future projects I would spend a larger proportion of the time doing research, putting together requirements and gathering and evaluating results. The majority of the time I dedicated to this project was during the design and implementation stages, because this is the part of the process I enjoy the most. Because of this, I managed to implement a lot of features, but perhaps neglected other parts of the software development process.

Overall this project has given me many practical skills I can take forwards, as well as teaching me many valuable lessons about the challenges of implementing a software development project.

10 Appendices

10.1 Appendix A - Test Cases

Test case number:		Related Use Case:		User Type:	
1		UC1 – View Supervisor		Student	
Success Criteria:					
Student is able to view supervisor details					
Preconditions					
• User account exists					
• Student has been assigned a supervisor					
Step	**Action**		**Response**		**Pass/fail**
1	User logs in		Dashboard page presented, supervisor name present in 'profile' card		Pass
2	User clicks supervisor name		Pop-up modal appears displaying supervisor name and available details		Pass
Comments:					
Behaves as expected.					

Test case number:		Related Use Case:		User Type:	
2		UC1 – View Supervisor		Student	
Success Criteria:					
Student with no supervisor assigned should not see supervisor details					
Preconditions					
• User account exists					
• Student has not been assigned supervisor					
Step	**Action**		**Response**		**Pass/fail**
1	User logs in		Dashboard page presented, supervisor name field not present in profile card		Pass
Comments:					
Behaves as expected					

Test case number:		Related Use Case:		User Type:	
3		UC2 – Upload Document		Student	
Success Criteria:					
Student should be able to upload PDF/DOCX document for task					
Preconditions					
• User account exists					
• File upload task available to student					
• Student on file upload page					
Step	**Action**		**Response**		**Pass/fail**
1	User selects pdf document		File accepted and preview shown, Upload button enabled		Pass

2	User selects docx document	File accepted, message shown in preview window 'preview only available for pdf files', upload button enabled	Pass
3	Upload file button clicked	Redirect to submissions screen, document appears in list	Pass
Comments:			

Test case number: 4	**Related Use Case:** UC2 – Upload file	**User Type:** Student

Success Criteria:
Student should be not be able to upload documents of type other than pdf/docx

Preconditions
- User account exists
- File upload task available to student
- Student on file upload page

Step	Action	Response	Pass/fail
1	User selects jpeg image	Image rejected, upload button remains disabled	Pass
Comments:			

Test case number: 5	**Related Use Case:** UC3 – View Marks	**User Type:** Student

Success Criteria:
Student should be able to view feedback about a task that has been marked

Preconditions
- User account exists
- User has been given feedback by supervisor
- User is on feedback page

Step	Action	Response	Pass/fail
1	User navigated to feedback page	List of tasks with feedback displayed	Pass
2	User clicks on a task	Feedback for task is shown in right hand panel	Pass
Comments:			

Test case number: 6	**Related Use Case:** UC3 – View Marks	**User Type:** Student

Success Criteria:
If no feedback has been given, should not cause an error

Preconditions

• User account exists • User has not been given any feedback • User is on feedback page			
Step	**Action**	**Response**	**Pass/fail**
1	User navigated to feedback page	Message displayed telling user they have not received any feedback yet	Pass
Comments:			

Test case number: 7	**Related Use Case:** UC4 – Arrange a meeting	**User Type:** Student
Success Criteria: Student should be able to schedule a meeting for a time and location		
Preconditions • User account exists • User is on schedule meeting page		

Step	**Action**	**Response**	**Pass/fail**
1	User enters name, picks date, adds time	Upload button enabled	Pass
2	User clicks upload button	Page redirected to student details, meetings tab. Meeting displayed in list of proposed meetings	Pass
Comments:			

Test case number: 8	**Related Use Case:** UC4 – Arrange a meeting	**User Type:** Student
Success Criteria: Student should not be able to submit a meeting without name, time and		
Preconditions • User account exists • User is on schedule meeting page		

Step	**Action**	**Response**	**Pass/fail**
1	User inputs name and date but not time	Submit button disabled	Pass
2	User inputs date and time but no name	Submit button disabled	Pass
3	User inputs name and time but no date	Submit button disabled	Pass
Comments:			

Test case number: 9	Related Use Case: UC5 – View assigned student and details about student	User Type: Supervisor

Success Criteria:
Supervisor should be able to see a list of students they are supervising

Preconditions
- User account exists
- User should be assigned at least one student to supervise

Step	Action	Response	Pass/fail
1	User navigated to dashboard page	List of students user is supervising appears in 'my students' card	Pass

Comments:

Test case number: 10	Related Use Case: UC5 – View assigned student and details about student	User Type: Supervisor

Success Criteria:
If user is not supervising any students should not cause error

Preconditions
- User account exists
- User should not have any students assigned to them

Step	Action	Response	Pass/fail
1	User navigated to dashboard page	Message displayed in 'my students' card informing user they have not been assigned any students	Pass

Comments:

Test case number: 11	Related Use Case: UC5 – View assigned student and details about student	User Type: Supervisor

Success Criteria:
User should be able to view details about student they are supervising

Preconditions
- User account exists
- User is assigned at least one student to supervise

Step	Action	Response	Pass/fail
1	User clicks on name of student from list in 'my students'	Application navigates to student details page for the student	Pass

Comments:

Test case number:	Related Use Case:	User Type:

12	UC6 - View/Download students work	Supervisor

Success Criteria:
Supervisor should be able to view and download students work

Preconditions
- User account exists
- User should be assigned at least one student to supervise
- Student should have submitted at least one file
- User is on student details page

Step	Action	Response	Pass/fail
1	Click on the 'submissions' tab	Upload tasks for which the student has submitted work are listed on right	Pass
2	Click on a task name to view submissions	Application redirects to submissions page for this task for this student, download button is shown if a preview of the file is available	Fail

Comments:
Supervisor can download documents via the browser pdf view window – however, as docx preview is not supported, any docx documents cannot be downloaded – manual download button should be added for all documents

Test case number: 13	**Related Use Case:** UC7 – Mark assignment	**User Type:** Supervisor

Success Criteria:
Supervisor should be able to submit a comment and/or mark about students work for a task

Preconditions
- User account exists
- User is assigned a student that has completed at least one assessment upload task
- User is on student details page, submissions tab

Step	Action	Response	Pass/fail
1	User clicks the mark icon	Mark assignment modal opens for the task	Pass
2	User fills in mark and comment, clicks submit button	Feedback is submitted, green notification appears on screen informing user feedback has been submitted	Pass

Comments:

Test case number: 14	**Related Use Case:** UC7 – Mark assignment	**User Type:** Supervisor

Success Criteria:
Supervisor should not be able to submit feedback if mark field is blank

Preconditions
• User account exists
• User is assigned a student that has completed at least one assessment upload task
• User is on student details page, submissions tab

Step	Action	Response	Pass/fail
1	User clicks the mark icon	Mark assignment modal opens for the task	Pass
2	User inputs a comment but leaves mark field blank	Submit task button is disabled	Pass

Comments:

Test case number: 13	Related Use Case: UC8 - View Tasks	User Type: Admin

Success Criteria:
User should be able to see a list of tasks

Preconditions
• User account exists
• At least one task exists

Step	Action	Response	Pass/fail
1	User navigated to manage tasks page	List of tasks stored within the system is displayed	Pass

Comments:

Test case number: 14	Related Use Case: UC8 – View Tasks	User Type: Admin

Success Criteria:
Should not error if no tasks have been created yet.

Preconditions
• User account exists
• No tasks have been created

Step	Action	Response	Pass/fail
1	User navigated to manage tasks page	Task is displayed with empty 'published' and 'drafts' section but no explanation	Fail

Comments:
It is hard to tell what has happened here from the user perspective, should instead display a message informing the user no tasks have been created yet.

Test case number: 15	Related Use Case: UC9 – Create new task	User Type: Admin

Success Criteria:
User should be able to create a task with a name, type, description, deadline

Preconditions
• User account exists

• User is on manage tasks page			
Step	**Action**	**Response**	**Pass/fail**
1	Click plus in top right corner	Application redirects to create task page	Pass
2	Fill in form and click submit	Application redirects back to manage tasks page, new task is visible at bottom of list of published tasks	Pass
Comments:			

Test case number: 16	**Related Use Case:** UC9 – Create new task	**User Type:** Admin
Success Criteria: User should not be able to submit a task if any fields invalid		
Preconditions • User account exists • User is on create task page		

Step	**Action**	**Response**	**Pass/fail**
1	Fill in all fields except name	Submit button disabled	Pass
2	Fill in all fields excluding type	Submit button disabled	Pass
3	Fill in all fields excluding date	Submit button disabled	Pass
4	Fill in all fields excluding time	Submit button disabled	Pass
5	Fill in all fields excluding description	Submit button disabled	Pass
Comments:			

Test case number: 17	**Related Use Case:** UC10 – Edit tasks	**User Type:** Admin
Success Criteria: User should be able to edit the details of an existing task		
Preconditions • User account exists • User on manage tasks page		

Step	**Action**	**Response**	**Pass/fail**
1	User clicks edit icon of a task	Application redirects to edit task page, displaying details about the current task	Pass
2	User edits some of the details and clicks submit button	Application redirects back to manage tasks page	Pass
Comments:			

Test case number: 18	Related Use Case: UC10 – Edit tasks	User Type: Admin
Success Criteria: User should not be able to submit an edit invalidates task		
Preconditions • User account exists • User on edit task page		

Step	Action	Response	Pass/fail
1	Delete name field	Submit button is disabled	Pass
2	Delete date field	Submit button is disabled	Pass
3	Delete time field	Submit button is disabled	Pass

Comments:

Test case number: 19	Related Use Case: UC11 – View all students	User Type: Admin
Success Criteria: User should be able to view list of all students		
Preconditions • User account exists • User on manage students page • There are student accounts in the system		

Step	Action	Response	Pass/fail
1	User navigated to manage students page	Table of students is displayed	pass

Comments:

Test case number: 20	Related Use Case: UC11 – View all students	User Type: Admin
Success Criteria: User should be able to sort the table by heading		
Preconditions • User account exists • User on manage students page • There are student accounts in the system		

Step	Action	Response	Pass/fail
1	User clicks id header	Table displayed sorted by Id	Pass
2	User clicks name header	Table displayed sorted by name	Pass
3	User clicks supervisor header	Table displayed sorted by supervisor	Pass

	User clicks placement provider header	Table displayed sorted by placement provider	Pass

Comments:

Test case number: 21	Related Use Case: UC12 – Assign supervisor to student	User Type: Admin

Success Criteria:
User should be able to assign a supervisor to a student

Preconditions
• User account exists
• User is on manage students page, and table is populated

Step	Action	Response	Pass/fail
1	User clicks 'assign supervisor' button for a student	Assign supervisor modal appears with no supervisor selected, assign button is disabled	Pass
2	User clicks a supervisor name to select it	Supervisor name changes colour and assign button is enabled	Pass
3	User clicks assign button	Supervisor is assigned to student, modal closes and supervisor appears in supervisor column for that student	Pass

Comments:

Test case number: 22	Related Use Case: UC12 – Assign supervisor for student	User Type: Supervisor

Success Criteria:
User should be able to re-assign a student that already has a supervisor

Preconditions
• User account exists
• User is on manage students page, and table is populated

Step	Action	Response	Pass/fail
1	User clicks 're-assign supervisor button for a student'	Assign supervisor modal appears with student's current supervisor selected	Pass
2	User clicks a different supervisor name to select it	Highlighted supervisor name changes from previous supervisor to new one	Pass
3	User clicks re-assign button	Modal closes, new supervisor name appears in supervisor column next to student	Pass

Comments:

This test case passes, but user can click assign button even if supervisor hasn't changed, this could be changed for better usability

11 References

[1]   QuantumIT, "InPlace - Smart Placement Solutions," 2020. [Online]. Available: https://www.inplacesoftware.com/. [Accessed 1 May 2020].

[2]   Creatrix, "Creatrix Campus - Placement Management," 2020. [Online]. Available: https://www.creatrixcampus.com/placement-management-software. [Accessed 1 May 2020].

[3]   QSR International, "Sonia Student Management," 2020. [Online]. Available: https://www.qsrinternational.com/sonia-student-management-software/about/sonia. [Accessed 1 May 2020].

[4]   Atlassian, "What is Agile?," 2020. [Online]. Available: https://www.atlassian.com/agile. [Accessed 2 May 2020].

[5]   Redmonk, "The RedMonk Programming Language Rankings: January 2019," 2019. [Online]. Available: https://redmonk.com/sogrady/2019/03/20/language-rankings-1-19/. [Accessed 2 May 2020].

[6]   Facebook Inc., "ReactJS," 2020. [Online]. Available: https://reactjs.org/. [Accessed 2 May 2020].

[7]   Google, "Angular," 2020. [Online]. Available: https://angular.io/. [Accessed 2 May 2020].

[8]   VMware inc., "Spring Boot," 2020. [Online]. Available: https://spring.io/projects/spring-boot. [Accessed 2 May 2020].

[9]   OpenJS Foundation, "Node.js," 2020. [Online]. Available: https://nodejs.org/en/about/. [Accessed 2 May 2020].

[10]   OpenJS Foundation, "Express," 2020. [Online]. Available: https://expressjs.com/. [Accessed 2 May 2020].

[11]   MongoDB, inc., "What is MongoDB?," 2020. [Online]. Available: https://www.mongodb.com/what-is-mongodb. [Accessed 2 May 2020].

[12]   Microsoft, "Microsoft Power Apps," 2020. [Online]. Available: https://powerapps.microsoft.com/en-us/. [Accessed 2 May 2020].

[13]   Amazon Web Services, "Amazon EC2," Amazon, 2020. [Online]. Available: https://aws.amazon.com/ec2/. [Accessed 3 May 2020].

[14]   Amazon Web Services, "Amazon Route 53," Amazon, 2020. [Online]. Available: https://aws.amazon.com/route53/. [Accessed 3 May 2020].

[15]   LearnBoost, "Mongoose," 2020. [Online]. Available: https://mongoosejs.com/. [Accessed 2 May 2020].

[16]   Microsoft, "Overview of Microsoft Graph," 2020. [Online]. Available: https://docs.microsoft.com/en-us/graph/overview. [Accessed 3 May 2020].

[17]   Microsoft, "Microsoft Identity Platform Documentation," 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/active-directory/develop/. [Accessed 2 May 2020].

[18]   Microsoft, "Overview of Microsoft Authentication Library (MSAL)," 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/active-directory/develop/msal-overview. [Accessed 3 May 2020].

[19] Postman, "Postman," 2020. [Online]. Available: https://www.postman.com/. [Accessed 3 May 2020].

[20] Google, "Material Design - The Colour System," [Online]. Available: https://material.io/design/color/the-color-system.html. [Accessed 4 May 2020].

[21] Google, "Angular coding style guide," 2020. [Online]. Available: https://angular.io/guide/styleguide. [Accessed 8 May 2020].

[22] R. C. Martin, Agile software development: principles, patterns, and practices., Prentice Hall, 2002.

[23] Google, "Angular Material," 2020. [Online]. Available: https://material.angular.io/. [Accessed 10 May 2020].

[24] RxJS, "RxJS Overview," 2020. [Online]. Available: https://rxjs-dev.firebaseapp.com/guide/overview. [Accessed 8 May 2020].

[25] Bootstrap, "ng Bootstrap," 2020. [Online]. Available: https://ng-bootstrap.github.io/#/home. [Accessed 8 May 2020].

[26] Electronic Fronteir Foundation, "Certbot," 2020. [Online]. Available: https://certbot.eff.org/. [Accessed 15 May 2020].

[27] A. Reinman, "Nodemailer," 2020. [Online]. Available: https://nodemailer.com/about/. [Accessed 10 May 2020].

[28] L. Merencia, "Node Cron - npm," 2020. [Online]. Available: https://www.npmjs.com/package/node-cron. [Accessed 10 May 2020].

[29] Auth0, "JSON Web Tokens," 2020. [Online]. Available: https://jwt.io/. [Accessed 10 May 2020].