Submitted for the Degree of M.Eng. In Computer Science 2020/2021

Marking Scheme - Software Development

“A Student Performance Monitoring App”
CODD01

Registration Number: 201709873
By Heather Thorburn

“Except where explicitly stated all the work in this report, including appendices, is my own and was carried out during my 4th year. It has not been submitted for assessment in any other context.”

Signature: 

Abstract

Excellent student performance is mutually beneficial to both students and educational institutions. Students will benefit as they can use what they have learned throughout their education to further their long-term career goals. It is also in the educational institutions best interest for their students to perform well, to improve their reputation and make them more attractive to prospective students.

There are many factors that can influence student performance, including socio-economic factors like family history and where they grew up. Measurable factors that can affect student performance include attendance, coursework grades and extenuating circumstances faced by a student.

The aim of this project is to create an application called StudentCheck for university staff and students to record these measurable factors and collate these into a concise format to measure student performance throughout an academic year. The application offers personalised feedback and recommendations and notifies appropriate personnel when a student’s performance is below satisfactory levels.

The project was implemented with flexibility in mind, making it usable across a variety of devices and database systems. The application also accounts for the shift to online deliverables and online classes. The project was tested thoroughly using black-box testing and distributed to university staff for user evaluation. University staff reported that it was an application they would consider using in their role.

Acknowledgments

I would like to thank my supervisor Alex Coddington for their continuous support and feedback throughout the process of developing this application.

I would also like to thank everyone who took the time to participate in the user background study and evaluation. The feedback gained greatly influenced the application.

1 Introduction

There are many factors that may influence a student's performance. Some of these factors may not be noticed by staff for a variety of reasons, such as a high student to teacher ratio or lack of data being recorded (Millea, 2018). Students may also be unaware of services available to them to help them improve their performance. This project aims to create an application that will simplify the process of gathering and viewing student performance data, and to offer appropriate advice to students and staff. This chapter will outline the aims and objectives of this project, and also explain the report structure.

1.1 Project Aim

This project aims to create an application that will allow university staff and students to view a student's performance throughout an academic year. The attributes collected will include class attendance, coursework grades and extenuating circumstances experienced by the student. The data will be uploaded by the teaching staff and students, and the performance data will be collated and displayed in a concise student profile with simple colour coding systems. Appropriate users will receive notifications when a student's performance data drops below a satisfactory level.

1.2 Project Objectives

The objective of the project is to create a web application with multiple types of user that allows them to easily upload, edit and view performance data, and to offer personalised advice in order to maximise student performance.

Teaching staff are able to use this application to easily upload coursework marks and attendance, and to view these statistics for any of the students participating in one of their classes. Heads of Department and personal development advisors are able to use this application to view cross-module information for students within their department and/or their advisees and to be notified of any concerning statistics that may lead to decrease in a student's performance. Students can to use this service to view their own cross-module data and therefore have the ability to reflect on their performance, and to be offered personalised advice to increase their

academic performance. This application should be simple to use and offer meaningful advice, in order to ensure that staff and students will use this service to compliment other services such as Moodle (Moodle, 2020). The application should also be as flexible as possible, such as being able to be used on a variety of devices, with the nature of in-person teaching at University being held in multiple locations throughout campus. The application should also be able to interface with a variety of database management systems, so that universities are not limited to a particular system. The application will be tested with users to ensure that it satisfies the above requirements.

1.3 Report Structure

This report is split into sections that cover the development process of this application. The following chapter, Chapter 2 will explain the background and related research conducted in order to determine the features that should be included in the application and to ensure that the problem is suitably solved by the application. Chapter 3 will address the project specification, and explains how the requirements were decided upon based on the background research. Chapter 4 will explain the design process of the application, with Chapter 5 explaining how features were implemented in more detail. Chapter 6 will include the testing of the application, and user testing to ensure that the application is fit for purpose, and finally Chapter 7 and 8 will cover the overall evaluation of the project and process.

2 Background and Related Research

2.1 Related Literature

The success of students is important for students and also educational institutions. Students will benefit from the completion of studies as it will aid in achieving their long-term career goals and provide them knowledge in their chosen area of expertise. Educational institutions also value successful students as this will aid in student retention and reputation.

By evaluating student performance, this application aims to aid the completion of the students' studies, and will also provide feedback to educators, ensuring they are successfully delivering course material, and tracking whether their students are

performing at or above a satisfactory standard.

To determine what factors influence student performance and why monitoring student performance is important, academic research was used to determine what data should be included in the application so that the information stored and viewed is useful for the users. Potential issues in current data collection methods were also addressed so that features can be included to mitigate these issues. Research was also conducted on what makes a successful application in terms of user interface so that users will want to use the application.

2.1.1 Motivation to record student attendance

Over time there has been research with varying results on whether attendance affects student performance. Some research shows that it is a fundamental indicator of overall performance, whereas some show that students can perform well during examinations despite non-attendance. Some studies show (Lukkarinen, 2016), that some students' academic performance is not linked to their attendance record, there are key groups of students where attendance and participation in classes is a significant indicator of how engaged the student is with the course (Rissanen, Anna, 2018). The research also shows that there is a correlation between student attendance and student retainment in higher education.

2.1.2 Issues with current method of taking attendance

The usual method of taking student attendance is via a paper record that is passed around a classroom while the lecture is taking place, or calling out the names of students log attendance. This system is inefficient and research (Mohamed and Raghu, 2012) shows that the former leads to problems regarding the false attendance of absent students, and the latter can consume approximately 5 to 10 minutes of teaching time in a class of 100 students. This system also creates a large burden on administrative staff, with studies (Bowen, 2004) showing that student retention staff spend up to 40% of their time working on data collection and entry. This time could be used more efficiently to ensure the retention of students in university. Therefore, there is reason to assume that an electronic solution would be beneficial to university staff.

2.1.3 Motivation to offer students personalised advice

Studies show (Illovsky, 1997), that utilising student counselling services sees increases in both student retention and assessment results. However, the Student Academic Experience Survey of 2016 (Neves, Hilman 2016) showed that 25% of students were not aware of how to contact counselling services and 7% did not know what services were available. Therefore, there is reason to believe an application that offers personalised student resource recommendations, as well as detailing who and how to contact, would be beneficial on student grades and retention.

2.2 User Survey

To further gather what features university staff would find useful in an application to monitor student progress, a survey was conducted and distributed to 16 university staff members, with 12 responses received. The survey was distributed to a range of different departments and universities such as Computer & Information Sciences, Social Sciences and Medicine so that the results were not restricted to a single department. It was also distributed to a range of staff roles such as teaching assistants and heads of department.

The format of the survey primarily consisted of possible features that could be included in the application based on the findings of the Literature Review and research into related technologies. Questions were also asked to determine what issues staff are currently faced with when recording performance data such as attendance, so that features could be included to help mitigate these problems.

The findings of this survey showed that the majority of respondents would be interested in using an application that allows you to take attendance during class, with 10 out of 12 respondents deeming that it would be a useful feature to include. The following question showed that 9 out of 12 respondents do experience issues with circulating a traditional register around a class, given a large class size setting such as a lecture.

The survey also showed that the majority of respondents would find the ability to view students performance data in a collated profile cross-module would be

beneficial, as well as the ability to export this profile to a file such as a PDF for later use.

The survey was opened for comments from the participants to recommend any other features they would find useful, or any comments about the features mentioned in the survey. Out of the 12 participants, 7 added additional feedback. Out of these responses, 3 stated that whilst there was value in the ability to view cross-module data for students, this should be a feature that is implemented carefully, and only accessible to certain members of staff, such as advisors and heads of department. The reasoning for this was that the ability for any member of staff to view cross module data could cause bias or create a misleading picture on how a student is performing.

Other features and feedback that were highlighted included, features should not take any additional time and should improve upon the corresponding traditional methods. For example, a participant highlighted the issue of requiring multiple registers for classes that are delivered concurrently. One suggested feature was an algorithm that could estimate the students likelihood to withdraw from university based upon past statistics, and another participant suggested a feature that encourages students to engage with relevant student support services.

The questions asked during the survey and detailed results of the user survey can be found in Appendix A.

3 Software Specification

The following chapter will define the project specification as a result of the background research, and functional and non-functional requirements of the system.

3.1 Functional Requirements

From the background research it was determined that attendance, coursework and extenuating circumstances experienced by the student are all factors that can affect a students academic performance and engagement. These can all be measured feasibly within the scope of this project. Using this MoSCow (Must have,

Should have, Could have) method, requirements that the user *must* be offered were defined. It was also deduced that there should be some limitations for what data should be accessed by members of staff and students to reduce bias. Therefore the functional requirements of the system have been defined below.

- ✔ Coursework Management
  - ✔ Class coordinators *must* be able to upload coursework marks for a coursework in their class.
- ✔ Lecture Management
  - ✔ Class coordinators *must* be able to record attendance for a class.
- ✔ Extenuating Circumstances
  - ✔ Staff and students *must* be able to upload any personal difficulties a student may be facing that is affecting their studies.
- ✔ Student Performance
  - ✔ Class coordinators *must* be able to view statistics relating to a students performance for their classes.
  - ✔ Student advisors *must* be able to view cross-module statistics for students they are advisors to.
  - ✔ Department heads *must* be able to view cross-module statistics for students within their department.
  - ✔ Students *must* be able to view their own cross-module statistics.
  - ✔ Students *must* be alerted if their performance is less than satisfactory for any of their classes
  - ✔ Class coordinators *must* be alerted if a student's performance is less than satisfactory for their class.
  - ✔ Advisors and Department heads *must* be alerted if a student's performance is less than satisfactory for more than one class.
- ✔ Personalised feedback and advice
  - ✔ Staff and students *must* receive personalised messages regarding the student's statistics.
  - ✔ Staff and students *must* be recommended student services depending on difficulties the student is facing that is affecting their studies.
- ✔ Limiting Access
  - ✔ Users *must* only be able to access pages that they are authorised to view, for example class-coordinators must only be able to view the students in their classes.

3.2 Non-Functional Requirements

The following have also been defined to ensure that this is an application that

university staff and students will want to engage with in conjunction with other services, and as a result of the background research.

- The application *should* be as flexible as possible
    - As there are many different database management systems that universities can use to manage their students and classes, the ability for the application to work over a variety of these would be beneficial.
    - The nature of universities means that students and staff are not in a static atmosphere. They may have classes in different rooms or buildings, or in the current climate may be working from home. This means that they could have varying access to technologies at any time, so the ability for this service to be used across a variety of devices, such as mobile and desktop, and over different browsers would be beneficial.
    - This has been labeled a *should have* requirement because it will still be possible to deliver an application that can outline the functional requirements, even if it can only interface with a specific database management system or is not viewable on other screen-sizes. However, it is still a requirement that will contribute to the overall success of the application.
- The application *must* be easy to use and navigate
    - As there are services that offer information about students already, the system must be as easy to use as possible to ensure that the users will want to use it as well as other services.

3.3 Project Plan

At the beginning of this project, a plan was developed to ensure that the project could be completed by defining what a basic and advanced implementation of this project would be. Additional features outside the scope of the project, which could be beneficial to the end user were also defined for potential implementation if there was adequate time and resources.

The background research and software specification were determined by December. Following this, research into the most suitable types of application, implementation languages and database structures was conducted. Then, an early prototype of the front-end was produced in order to familiarise with the chosen implementation languages, libraries and frameworks. The database was then constructed and integrated into the prototype. After the prototype was produced, a

technology that would allow for greater portability with regards to the database was discovered, so amendments to the plan were made.

After the initial prototype was created, the basic implementation of the program was scheduled for completion by 9th February. The basic implementation addressed all functional requirements. Particular points of interest would be as follows:

- Ability to record attendance via a simple checklist.
- Coursework marks were added individually.
- Student profiles and tables that displayed students were simple with minimal styling, and no ability to search, sort or filter.
- The navigation was conducted by a static navigation bar.
- The overall styling was minimal but ensured that the system is viewable across a variety of screen sizes.

The advanced implementation expanded on the functional requirements to offer the user more choices when uploading data, and styling would be improved upon. This stage will be completed by the 23rd of February. Additional features will include:

- Coursework marks and attendance can be uploaded in CSV file format.
- Searching, sorting and filtering students will be implemented in all relevant tables.
- Additional styling will be implemented to make the application more visually appealing.

After this stage any additional features that can be feasibly added will be included. This included the ability for:

- Students can record their attendance themselves via a QR code.
- The ability to export student profiles to a PDF.
- Automated emails that send statistics to relevant users.
- Ability to record attendance at classes where the entire enrolment list may not be listed to attend, such as Labs and Tutorials.
- An algorithm to calculate overall student engagement.

This stage was completed by 4th March.

After this, testing, evaluations and documentation was conducted and the report was completed by 22nd March, to allow sufficient time in case of changes of schedule or any errors or difficulties that may be encountered.

The project followed an Agile approach as stages in the application development were designed, implemented and tested individually. Agile methods such as user stories were used to help define requirements which can be found in Appendix B.1 and more detailed overviews of the project plan can be seen in Appendix B.2 and Appendix B.3.

4 System Design

The following chapter will describe how StudentCheck was designed prior to the final implementation as a result of the background research.

4.1 Single-Page Applications and Multi-Page Applications

Research was carried out regarding what style of application offered a better user experience, Single Page Applications or Multi-Page Applications. Multi-page applications (MPAs) consist of multiple pages, when there is a transition between two of these pages, a request is sent to the server for the contents of the new page, the server then sends a new file which is loaded in the browser. This results in the browser completely reloading all contents of the page, even contents which have remained the same such as headers and footers. MPAs do carry some benefits. It allows for pages to be more visible in search engines, as keywords can be added to individual pages. As it is the “traditional,” way to develop web applications, there are many technologies that can be used for development. (Solovei, Olshevska and Bortsova, 2018)

However, MPAs do have some issues. When a new page is selected, MPAs must send a request to the server and wait for a response. This typically means they are slower than Single-Page Applications (SPAs), which can lead to a worse user experience and performance issues. MPAs server-side and client are also highly coupled which decreases the portability of an application.

SPAs differ from MPAs as they only get one file from the server. This page is loaded and is never refreshed when navigating around the application. Instead, only parts of the page are re-rendered and communication with the server-side is done using asynchronous requests.

As the page never reloads, it offers a much smoother user experience. Elements such as navigation and headers can remain constant, and because entire pages do not need to be requested and loaded, SPAs tend to be faster than MPAs, again benefiting the user experience. It also has the benefit of ensuring the client-side and server-side are decoupled or very loosely coupled, adding to the portability and allowing easier maintenance (Solovei, Olshevska and Bortsova, 2018).

4.2 Application Architecture

For the scope of the project, to maximise the user experience and make the system as fast to use as possible, it was decided that a Single-Page Application would be most suitable. The application will consist of a client side and server side. The client side will handle the display of the user interface and handling of the user input. The server side will manage data and requests to the database.

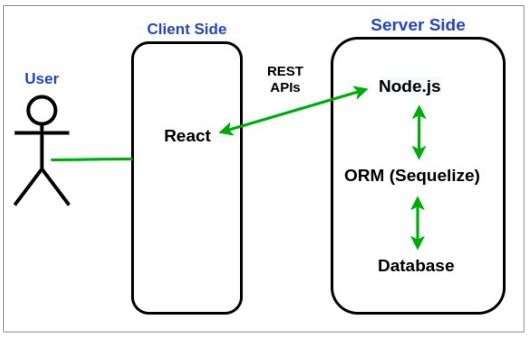


Figure 1: Diagram of Final Application Architecture

They communicate using REST APIs, which are requests and responses which use the HTTP protocol and HTTP methods such as GET, POST, PUT and DELETE.

4.3 Client-Side Architecture

To further expand on the client architecture, methods of organising code were researched. A familiar pattern is the Model, View, Controller pattern (Sanderson, 2010). This is where the client side is split into three components. The Model handles the data for the application, the View is responsible for displaying the UI and the Controller acts as an intermediary between the Model and View. However, as more functionality is added to the application, there is a need for more models and views to accommodate these.

As frameworks were being researched for the client side, other approaches to organise the client-side were considered. This led to the use of the Redux Architecture (Redux, 2020).

The Redux Architecture contains four entities. The View, Actions, Reducers and the Store. The View is composed of all components that make up the UI of the application, and the components pass data to Actions. The Actions are where the data is then dispatched to the store, for example, dispatching a REST request to the server-side. The Actions corresponding Reducer then changes the state of the application depending on the Action dispatched and data received. This result is kept in a store, which can be accessed from anywhere within the View (Paul, Nalwaya, 2019).

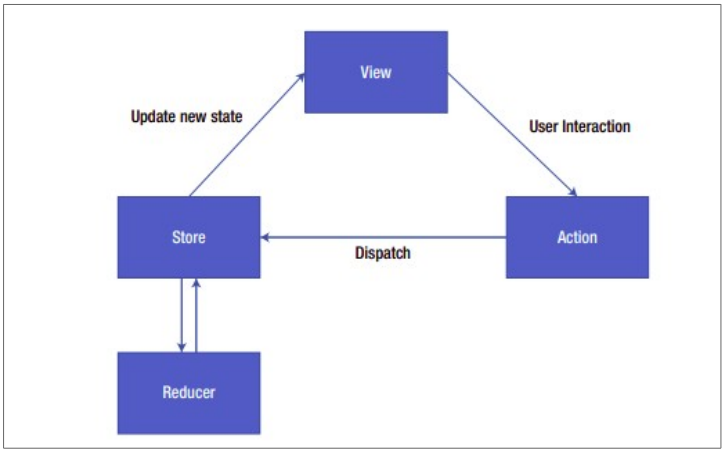


Figure 2: Demonstrating Redux Structure Source (Paul, Nalwaya, 2019)

It was decided that the React framework would be used for the client side, as it is a well-regarded framework for SPAs, and that Redux would therefore be a more suitable option. It allows for the client side to be organised in a maintainable way and Redux is a very popular way to organise code with React. This meant there was plentiful documentation and support for using the official React Redux library (React Redux, 2020). It also has the benefit that all the states are kept in a singular store, it would therefore be easier to add to and modify, rather than dealing with many views and controllers.

4.4 Front End High Level Architecture

As React was being used, it was necessary to ensure that the View is organised in a way that is easy to understand to make the application more maintainable.

There is a directory that will contain the ‘Screens’. These are the result of all components necessary combined into one component which will be displayed to the user when they navigate to a different page. There is a ‘Components’ directory which will contain all smaller components that are re-used throughout the application and will be split up into further directories such as Tables and Profile Components. During implementation many screens and components were added

which were not determined during the design phase. Another subdirectory was also added, called “Functions”. This was added to accommodate for the PDF generation.

The Screens directory was split into subdirectories that will essentially represent the different functionalities of the program. Any screens that do not share a similar screen and therefore would not require an entire subdirectory is added to the Screens directory and clearly named to represent its functionality. It consists of:

- Attendance - all screens required for taking attendance.
- Coursework - screens required for adding, editing and marking coursework.
- Profiles - class profiles and profiles for departmental staff and advisors.
- ViewStudents - responsible for screens that display lists of students in a table format.
- Notifications - screens to display notifications for students and staff.

4.5 Server-Side Design

Originally the server-side was composed of different files to hold relevant routes for the REST APIs, and a file that established a connection with the database. However this structure was revisited when it was decided an Object Relational Mapper would be implemented.

To accommodate this, a directory called, “Models,” holds all the abstractions created from the database. These abstractions are objects which reflect all the tables and views contained in the database, along with their associations to other tables and together make a “virtual database”, which can be used to query like a regular database.

Another file *authorisation.js* was added to contain all functions and modules necessary to authorise user requests. A file, *server.js* handles the connection to the database and establishes the routes for the REST requests.

4.6 Database Design

As this system deals with multiple types of user and has different functions that require storage, such as coursework and attendance, it was evident that the database would be large and require sufficient planning. The database was designed to represent the data needed for all features of the system requirements

whilst avoiding data redundancy. It was decided that a relational database would be used. This allows the data to be easily split into categories such as students and staff, and allows joins between different tables.

Views were also created to store the results of queries that were neccessary throughout the application, such as attendance and weighted grades.

To ensure a successful implementation and demonstration of this system, assumptions about the data available were made based on existing student management systems such as Moodle. As the database structure is large, the full database structure and assumptions made can be seen in Appendix C.

4.7 User Interface Design

StudentCheck was designed to be simple and easy to use. Therefore it was important that the application could be easily navigated and free from unnecessary content.

Other factors taken into account when deciding the user interface include issues regarding accessibility. Wherever a colour was selected for the main colour scheme, the colour was tested for its contrast to ensure that it meets the WCAG Level AAA requirement, where a contrast ratio of 4.5:1 for normal text and 3:1 for large text is required (W3C, 2020).

A colour coding system for performance was also planned to be implemented. The obvious choice would be to associate green with, ‘good’, statistics, orange with, ‘concerning’, and red with ‘bad’, statistics, however this could prove an issue with red/green colourblindness. For this reason, it was decided that while these colours would be used, the colour will not be the only performance indicator. For example, when indicating a ‘good’ performance it will be colour coded in green, but another indicator will be very clearly shown as well that is not dependent on colour. This principle was implemented throughout the system, where colour would not be the only indicator of features or messages.

A defined colour palette was used for the application. The selected colours had a wide range of uses throughout the application and would therefore allow the

application to look more consistent (Garrett, 2011).

4.7.1 Navigation

As the pages available for staff and students will be significantly different, instead of creating a single navigation system with many conditional statements, two navigation systems were created for students and staff respectively.

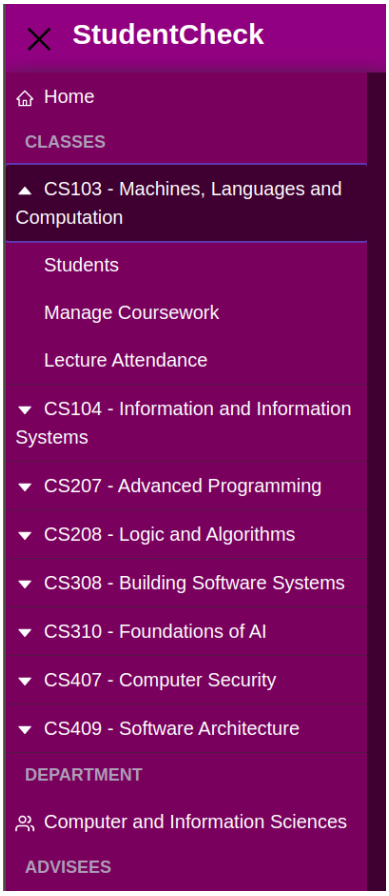


Figure 3 : Staff Navigation Bar

For the staff navigation system, to improve user experience it was desirable to allow staff to navigate between classes, departments and advisees where applicable from any point in the application.

Therefore a *global navigation* element would be added, which allows the user to access key points within the program from one consistent place (Garrett, 2011). This led to a choice between a navigation header or a sidebar that are constant throughout the application. As staff are likely to teach many classes in an academic year, it was decided that a sidebar would be more suitable. After creating some designs for a navigation header and a navigation sidebar, it appeared that the navigation sidebar could hold more information that was easily accessible, and any overflow could easily be accounted for by making the sidebar scrollable. The sidebar also appeared to be friendlier towards mobile devices as the sidebar can be hidden and accessed via some kind of toggle, which would be more difficult with a header navigation bar.

Next, there should be a way to separate the different functionality for staff where applicable. This includes the ability to view students in classes, take attendance, manage coursework for a class, view students in a department or students that a staff member advises. To keep the sidebar as concise as possible, the different

functionalities were added into dropdown menus so that the user can access these at will.

Despite students having significantly fewer features accessible to them, for the purposes of development, the same sidebar structure was used. However, this could be addressed during further developments, as students only have three pages which they could navigate to, this resulted in the sidebar looking quite sparse.

4.7.2 Displaying Students

To find student profiles, it was decided that a list of students related to a class or department would be listed in a table, with a relevant link. To help identify students who may have concerning statistics, the table will contain custom cells that display the statistics using the defined colour scheme.

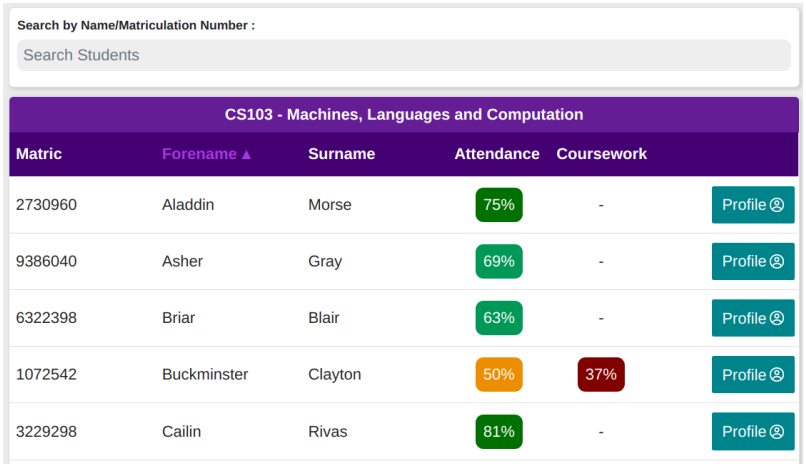


Figure 4 : Class Table Screen

The initial prototype allowed the table and selected students to be displayed on the same screen. The user would click on the student they wish to view, and their profile would be dynamically loaded in a component to the right of the table. The reasoning for this was to allow staff to easily view and compare student statistics. This early design can be viewed in Appendix E.

However, this was replaced with a separate screen for the profile and the table of students. This was because the screen essentially had to be split in half for the early design, and therefore not much information could be displayed in the table while keeping the font at an accessible size, this led to a lot of overflow in the table. The early design also posed the issue of displaying this design on smaller screen sizes.

As the number of students in a class can be very large, the table will be sortable. A search bar will be included so that users can easily find a particular student.

The table will include a button which takes the user to the students more in-depth profile.

The table will be amended for department heads and advisors. Instead of the attendance and coursework columns, there will be a warning sign in the table row if the student's coursework and/or attendance is less than satisfactory for more than one class. This will be as defined in the functional requirements, this is when these users will be notified. It was considered to include an overall coursework grade and attendance statistic across all classes in this table, however the warning sign seemed to more closely align with the functional requirements.

4.7.3 Student Profiles

In order to create a concise student profile, it was decided that related information would be grouped together to aid readability and allow the user to more easily identify the information they are looking for. For example, general information about the student would be grouped, and attendance information would be grouped.

An early prototype displayed all of these groups in one large component so that the user could view all of the information about a student at once. This early prototype can be seen in Appendix D. A graph was also added, however it took up a lot of space, and after using multiple libraries to generate graphs, it was found that they were not responsive and did not display well on smaller screen sizes. As more information was added to the profile, it began to look cluttered and difficult to read. Therefore the profile was revisited, and the general information and information regarding their performance statistics were separated into two components. These could be switched between using a toggle. This meant that more information could be displayed without compromising the readability of the components. This also supports the grouping of relevant information, as the academic information is kept separate from the general information of the student.

Absences will be displayed if any exist. This allows users to spot if there was a time that attendance levels were particularly low. A breakdown of grades will also be offered. The grade statistic is weighted, therefore if a student were to score 100% in a coursework worth 75% and 0% in a coursework work 25% their coursework average will be high, however students and staff may still want to reflect on the low scoring coursework.

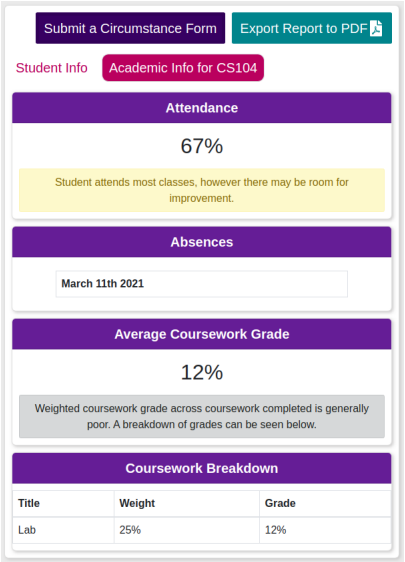


Figure 5 : Student Class Profile

The statistics will be accompanied by a text description following a colour coded scheme. On this page, they will be offered an option to log an extenuating circumstances form to add a note about any issues a student may be facing which may affect their studies. As the further implementation of exporting the profile to PDF was included in the implementation, an option for this will also be included.

Departmental heads, advisors and students will have access to a profile that combines all to these profiles for each class. These will be added to the navigational toggles.

4.7.4 Extenuating Circumstances

The form will allow the user to enter input and recommend services to the student. After researching best practices for online forms, to improve the flow and usability of forms, one topic should be addressed at a time. If there are multiple topics, they should be split across different pages (Jarrett, Gaffney 2009). Therefore the form will be split into multiple sections, one section for user input, one for the recommendations and the last to review and submit the application.

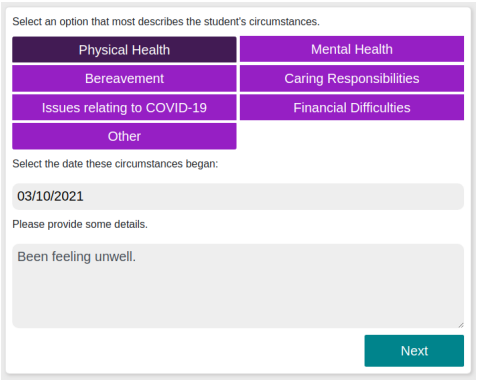


Figure 6 : Circumstance Form Screen

The user will be offered a selection of options to choose from that most closely match their circumstances. They can then input any further details regarding the issues they're facing, such as the date the issues began. Regarding selecting the option, some research was conducted into how to do this and it was decided that buttons that act like a radio selection would be a better solution than say a dropdown (Garrett, 2011). This was because dropdowns can hide options from the users view, and can also cause issues if there are too few or too many options. Too few options means that the space saved by a dropdown makes using the dropdown nearly redundant, and too many can lead to a confusing list and can potentially cause overflow.

Clicking on the next button will take them to a page where services that are matched to their previous selection are displayed with links to the services webpage. Following this they can navigate to another page where they can review the entered information and submit. Submitting this will return a message displaying that the coursework has been successfully submitted or a suitable error message.

5 Detailed Design and Implementation

This chapter will describe the detailed design and implementation of the fundamental features in the application

5.1 Technologies Used

A variety of different technologies were used in the implementation of this project. The following section will describe the technologies used, why they were used and any alternatives that were considered.

As many technologies were used, only the libraries, frameworks and implementation languages used frequently through the application were fundamental to meeting the requirements of the system. Any other technologies used will be included in the further sections and in Appendix F.

5.1.1 Server Side

5.1.1.1 Node.js

Node.js is a JavaScript run-time environment (Node.js, 2021) that was used for the server-side of the application. This was used as it allowed the client-side and server-side to both be written in JavaScript, leading to faster development and easier maintenance.

5.1.1.2 Sequelize

Sequelize is an Object Relational Mapper (ORM) for Node.js (Sequelize, 2021). Object Relational Mapping is the process of generating virtual objects that map to tables in a relational database. These objects can then be effectively queried in the programming language, in this case JavaScript, rather than in raw queries. In Sequelize, these generated objects are called, ‘models’.

In the original implementation of StudentCheck, queries to the database were written as raw queries. However as the database grew, it became apparent that a lot of time was being spent ensuring that the syntax for the query was correct and many parts of queries were repetitive. Sequelize allows you to conduct queries with much simpler and more intuitive syntax. This meant that queries could be more effectively written with fewer syntactic errors, For example, a *“SELECT * FROM students”,* query is simplified to *“Students.findAll()”.*

Another benefit of using an ORM for StudentCheck is the fact it abstracts the database system. During the implementation of StudentCheck a MySQL database system was used, however Sequelize is compatible with a variety of relational database systems such as Postgres, MariaDB, SQLite and Microsoft SQL Server. Sequelize includes a query-builder, which generates vendor specific queries, so switching between database systems can be done with minimal changes to the backend of the system. This therefore adds portability to the application. The query builder also adds security as it uses parameterized queries, which help mitigate SQL injections. It has multiple ‘sync’ functions which can generate tables and relations in the database system from the models. This proved particularly useful when mitigating the MySQL database to DEVWEB from working locally. It also means that

if a database does not contain a table it can easily be added from the application.

Some issues did arise when using Sequelize. During implementation, it was noticed that when conducting a query which included an attribute that you may wish to be sorted, for example, a list of absences by a particular student, they could not be ordered. The documentation was consulted and while there was support to order your query results by an attribute, it was not supported for ‘nested,’ queries. This means any queries that include a join. At the point of writing, this is an open issue for the developers of Sequelize that they are aiming to fix. As many of the tables implemented in the application have a sorting function, this problem was mostly mitigated.

5.1.2 Client Side

5.1.2.1 React

React is a JavaScript library (React, 2021) that was used for the frontend development of StudentCheck.

In early development, plain JavaScript was used. However, as more elements were added to the HTML DOM, it became difficult to keep track of the class names and IDs of different elements to ensure that data was displayed correctly. The resulting code became disordered and challenging to maintain. Many event listeners also had to be added manually in order to keep the application as responsive as possible. JavaScript libraries were therefore considered to organise the code efficiently and to aid the development process.

React was selected as it allows developers to create ‘components’ which are pieces of the user interface. These are then combined to create the overall user interface of the system. The components are easy to reuse throughout the system, therefore adding consistency to the user interface, speeding up the development process, and aiding maintainability.

React also stores a virtual representation of objects in the DOM, which is referred to as the virtual DOM. Whenever there is a change of state or properties (referred to in React as *props*) of a component, this virtual DOM is compared to the browser DOM.

Only the objects which differ between the virtual DOM and the browser DOM are updated. This provides a smoother UI, as objects are not unnecessarily re-rendered when there is a change of state somewhere in the system.

5.1.2.2 React Redux

As Redux was used to organise the client-side code, React Redux (React Redux, 2021) was used as it is the official Redux binding for React.

The Redux store was also used in the application to maintain consistency throughout the UI for a logged-in user and to prevent unnecessarily rendering components and dispatching calls to the server side. For example, if the store contains a list of students for a given class, and the user navigates to a page which displays this list of students, it would be unnecessary to make another call to the server side.

It was also useful to easily determine the status of calls to the server-side. For example when the client makes a call to the server side and it has not yet received a response, the state is set to, ‘loading,’ which can be used to give feedback to the user, such as a spinner component.

5.1.2.3 React Bootstrap

React Bootstrap (React Bootstrap, 2020) is a frontend framework with many pre-made and reusable components. It was used for objects that occur consistently throughout the system such as buttons, alerts and cards.

React Bootstrap was used in order to speed up development, and to provide consistency in the user interface.

5.1.2.4 CSS

Cascading Style Sheet (CSS, 2020) was used in order to further style the system to make it more visually appealing. It also allowed adjust the user interface to different screen sizes through the use of media-queries. This allowed for the page layouts to be changed and unnecessary components hidden from view on smaller screens.

5.1.2.5 jsPDF

jsPDF (jsPDF, 2020) library was used in order to generate PDF reports for students,

along with the plugin jsPDF-Autotable (jsPDF-AutoTable, 2020).

An alternative option that was considered was to generate the PDFs using Puppeteer. Puppeteer (Puppeteer, 2020) is a library for Node.js which allows you to create a headless Chrome which allows you to navigate to a given URL. This can be used to take screenshots of browser pages and to generate PDFs from these screenshots. This carried the benefit of being able to style the PDF with consistent styling to the rest of the system. The process is also carried out on the server-side, therefore not using the user's browser resources.

The Puppeteer approach was very difficult to implement, and when successfully implemented it was found to be excessively slow, taking at least 10 seconds even after researching and revisiting the code specifically to improve speed. Alternative approaches were researched, and jsPDF created a PDF with the information desired at a much more reasonable speed. However, it was far more limited in terms of styling, and is generated on the client-side.

It was then decided that the jsPDF approach was a better solution, as a non-functional requirement of StudentCheck was fast and easy usability. It also displayed the necessary information adequately and as both the tables within the StudentCheck application and the tables generated by jsPDF were relatively simple anyway, the differences in styling between the application and the exported PDFs did not seem very noticeable.

5.1.3 Other Technologies

5.1.3.1 JSON Web Tokens

After conducting the first user survey, it was evident that confidentiality of student data was crucial for the end user. This meant that it was necessary to implement a way to authorise users to view certain pages.

JSON Web Tokens (JWTs) (JWT, 2020) provide a way to securely transit information as a JSON object. JWTs are separated into three parts, the header, payload and signature. The header will usually consist of the type of token and the signing algorithm. The payload will contain the claims about the user and any other

additional data that could be used for authorisation. For example, in the case of a teacher, the JWT will contain the classes that the user possessing the token teaches. The signature is then generated using the encoded data within the header and payload.

5.1.3.2 Chrome DevTools

Chrome DevTools (Chrome DevTools, 2020) also have many useful extensions which were used during development. For example, an extension to monitor the Redux store was used, as well as an extension called LightHouse (LightHouse, 2020). LightHouse tests individual pages against accessibility standards and best practices, and creates a concise report on areas where your page could be improved. LightHouse was particularly useful to ensure that the contrasts between different components were adequate and that aria-labels and roles were used correctly.

5.2 Coursework Functionality Development

The following sections will describe the development process to allow staff to add and upload coursework marks for students in their class.

The functionality associated with adding and editing student marks required forms to gather user data to be sent to the server. For forms across the application, it was ensured that the *label* tag was used to allow for form controls to be easily identified, and that buttons to carry out actions were clearly identifiable.

5.2.1 Creating Coursework

Initially, it was intended that the coursework would be added to the database management system directly. However, it was decided early on in implementation to allow class coordinators the ability to add, edit and delete from the application to allow them more freedom to manage their coursework for students.

To add coursework, a class coordinator can navigate to a page where they will be presented with a form to input a coursework title, coursework description and the weight of the coursework. When the submit coursework button is pressed and the user input validation functions are passed, the data from the form is dispatched to the server-side to be added to the database.

Extra validation was added to ensure that the class coordinator cannot add a coursework that makes the total weight of the coursework added exceed 100%. A count of the coursework already added is held in the Redux store which is used at the validation stage.

At the server-side when a new coursework is added, a list of students who are enrolled for the relevant class is generated using the Sequelize function *findAll()*. This is generated using the association between Students and Classes through Enrolment. When the coursework is added to the database, it also executes a batch insert query using the Sequelize function *bulkCreate().* This adds all students enrolled in the class to the Coursework Grades table with the coursework added ID, and null values for feedback and grade. This is making the assumption that all students enrolled in the class are expected to complete coursework for the class. This was implemented so that students who had not received a grade could be easily identified, rather than comparing an enrolment list to a list of students who had received grades. This would also be useful when implementing bulk addition of grades via CSV file in later implementation.

While user input validation was implemented, to further account for user error, an edit coursework functionality was added so that class coordinators can edit the coursework they have already input to the database.

5.2.2 Editing Coursework

The appearance of the Edit Coursework form is very similar to that of the addition of coursework with a few key differences. The title, description and weight from the database are set as default values for the inputs so that the user can easily see what information is already present in the database.

The functionality to delete the coursework is also added. Due to the nature of the association in the database between coursework and coursework grades, and to avoid redundant or invalid data, removing coursework will result in a cascading deletion of all coursework grades associated with the coursework. This is a drastic action, so verification of the user action and the consequences of this action was added.

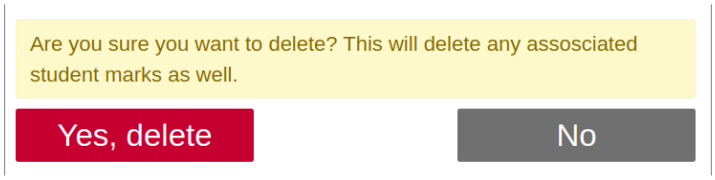


Figure 7: Coursework Deletion Verification

Upon requesting to delete a coursework, the user is presented with an alert that makes them aware that deletion will result in the deletion of coursework marks associated. The buttons to confirm or return are placed far away from each other to prevent accidental clicking of the wrong button and the button to delete is labelled again as delete, which removes all ambiguity.

Clicking No will revert the form back to its original state, and clicking Yes with dispatch an action to delete the coursework from the database and all corresponding entries in the CourseworkGrades table.

5.2.3 Adding Coursework Marks

Two ways to add coursework marks were implemented. The first way, as highlighted in the basic implementation of the project, was adding and editing coursework marks individually against a list of students enrolled in the class. The ability to bulk add coursework marks was added to make adding coursework marks easier and faster for staff coordinators.

5.2.3.1 Adding Coursework Marks Individually

As when a coursework is added by the class coordinator, this triggers a *bulkCreate()* operation of students enrolled in the class in the coursework grades table, this functionality was relatively simple to implement. An action is dispatched to the server-side which returns a list of all the coursework grades associated with a coursework and displayed in a table. One of the table columns has a custom cell, which will be conditionally rendered. If the grade entry is equal to null then it

displays that this student has not yet received a grade. Otherwise, if a grade exists, this grade is displayed in the table using the colour grading.

A button in the table corresponds to a link to a form where the class coordinator can edit and save an individual students grade.

5.2.3.2 Adding Coursework via CSV

As part of the advanced implementation, this functionality was added. On this page, the user is prompted to upload a CSV file. To successfully implement this, a CSV of the correct format is required from the user. For this reason, an information alert is displayed with instructions for the user on how they should format their CSV file.

The CSV must contain three headers, “Matric”, “Grade”, and “Feedback”. Matriculation number was chosen as an identifier for the student as it is likely that a student will include their student ID somewhere on a coursework submission and this can be used to uniquely identify a student in the database. The list of students who are expected to complete this coursework is collected from the state. When the CSV file is uploaded, a CSV parser called ***papaParse*** (PapaParse, 2020) is used to extract the information and, provided that the user has formatted their file correctly, each student ID entry in the.csv file is compared to the list of students. If an entry does not exist in the list of students, the entry is pushed to an error array so that entries that cannot be added to the database are relayed to the user.

If an entry is recognised, it is converted to a JSON object and added to an array. The completed array is dispatched to the server-side, and a bulk update is performed. This was done using another *bulkCreate(),* however adding the Sequelize parameter *onDuplicate* and defining the attributes to be updated. As error checking has been conducted on the client-side, this should be successfully added, however if there are any errors that occur on the server-side, these are relayed to the user via a status 400 sent from the server with an accompanying message. The user will also be displayed with any entries in the error array that were not added to the database. It was decided to allow uploading coursework marks, even if there was an error in the file and some entries were recognised to account for situations where maybe one or a few student matriculation numbers are wrong. With the

functionality to allow the user to upload coursework marks individually, it is simple for the user to navigate back, and check which students have not been assigned grades.

Upon testing this functionality, some problems were encountered. The parser did not trim the input that it was parsing, therefore any accidental spaces entered meant that the coursework marks would not be successfully added to the database. This was simple to mitigate by ensuring that the output from the parser, specifically the student matriculation number attribute, was trimmed of any whitespace before being used to write to the database. To further account for some user error, the headers identified by the parser are converted to lowercase to account for any differentiation in case. The parser also appeared to add an extra row to the parse that was completely empty. This did not necessarily cause any issues with uploading marks to the database, but it meant that there were some extra redundant loops when doing the comparisons between the class list and the file uploaded. This was mitigated as this appeared to be a common issue of the parser and the empty row was to indicate the end of the file, and a command had been added to ignore empty rows.

5.3 Attendance Functionality Development

The following section will describe the process of allowing staff to record attendance for classes. The user is offered three ways to record attendance. One is a simple checklist, which was inspired by the traditional ways of recording attendance, such as a register being passed around a room, or names called out during the class. The second option is via uploading a .csv file and the final option is generating a QR code that students can use to record their own attendance.

During early implementation, class sessions were added to the database directly. These would then be queried and the user could make a selection from the lecture sessions available in the database. This added the benefit of avoiding user error when adding lectures themselves, however it was found to be limiting. It offered little flexibility in the case that a lecture time or date changed, or if the lecture was cancelled altogether. It also meant that ad hoc sessions could not be added without interacting directly with the database.

The options to display the selection was also difficult to implement so that it was easily usable on desktop and mobile. For example, dropdown menus were used for lecture selections, however if there are too many lectures available it can lead to scrolling issues.

In a first step to mitigate this, the query to get lecture sessions was limited to a week before the current date and the current date. If the results were less than 3, then they could be selected via radio buttons, and more than that could be displayed in a dropdown. This helped the user interface issue, however did not address the flexibility issue.

Instead the user is presented with a React component called *DatePicker* (DatePicker, 2020). This presents a text input where users can either type in a date, or a calendar will be presented where they can select a date. A similar component is used for the time of the lecture. This allows the user flexibility in creating lectures and also has a friendlier user interface. When the lecturer submits the lecture date and time, this information is dispatched to be added to the lecture sessions table and the ID generated is sent from the server to the client-side so that it can be used for attendance taking. When the state is updated with a lesson ID, the inputs are disabled so the user cannot try to edit the date after the lecture has been created. If they have made an error, they can press the Undo button which removes the ID from the state.

This approach however is not perfect, as it can lead to lecture sessions with no attendance. For example if a user were to submit a lecture then navigate backwards without taking any form of attendance. This does not lead to any errors in student attendance statistics as the actual attendance of students is stored in another table, however it can lead to redundant entries in the lecture sessions table. Ways to mitigate this will be discussed in further developments.

The basic implementation meant that an assumption was made that all students enrolled in a class will be expected to attend every class held. This obviously does not account for classes where only a selection of students are expected to attend, such as tutorials and seminars.

After the implementation it was decided to at least add some functionality to allow taking the attendance at these smaller classes. This was added for the attendance taking option using the checklist. It was not added to other options due to time constraints, and would be added to the other options in further developments.

5.3.1 Attendance via Checklist

This function was the simplest method to implement. A React component called ***React Data Table Component*** (React Data Table Component, 2020)**.** This was used throughout the application to maintain consistency in displaying data. It also has the functionality to easily implement selectable rows via a checkbox, and the selected values are dynamically added or removed to an array. There is also functionality that allows you to define your own function, which is called whenever there is a change in the selected rows.

To take the attendance, the students enrolled in the class are requested from the server. This is displayed in the data table component. When a student is selected or deselected, a function is called that takes the array of selected rows generated by the data table component which is an array of JSON objects with unnecessary information about the table, and extracts the student matriculation numbers.

When the user decides to submit the register, the full register is looped through, and if the matriculation number in the current loop is not present in the selected rows, it is added into an array of matriculation numbers. We now have two arrays, one of matriculation numbers representing students that are present and another for students who are not present.

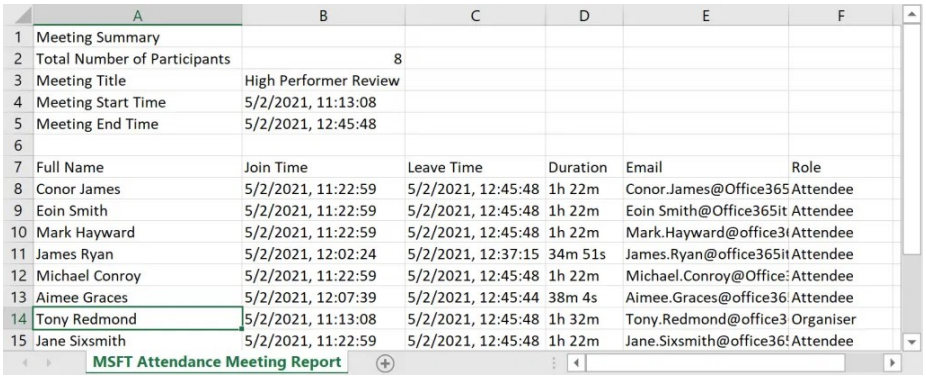
This information is sent along with the lecture ID generated to the server, where two bulk create statements add the present and absent students.

5.3.2 Attendance via CSV

Zoom (Zoom, 2020) and Microsoft Teams (Microsoft Teams, 2020) both offer the functionality to export the reports about sessions to a CSV file. They are composed of details about the meeting such as time and date, but also list all attendees in the meeting. The application offers the user to upload one of these reports.

Examples were found of the reports generated by Zoom and Microsoft Teams. The reports generated were slightly different in terms of headers, for example Microsoft Teams uses the header “Email”, whereas Zoom uses the header, “User Email”.

Instead of asking the user to edit the header to some constant header, the parser ***papaParse*** will look for headers that are either, “User Email,” or, “Email” to account for Zoom or Microsoft Teams reports. This was done to avoid giving the user more work by having to ensure that the header is a certain value. However a message is displayed to the user stating that if they are using another service or a custom .csv file containing attendance, they can still use the function so long as the CSV file contains the headers, “User Email,” or, “Email”.



	A	B	C	D	E	F
1	Meeting Summary					
2	Total Number of Participants	8				
3	Meeting Title	High Performer Review				
4	Meeting Start Time	5/2/2021, 11:13:08				
5	Meeting End Time	5/2/2021, 12:45:48				
6						
7	Full Name	Join Time	Leave Time	Duration	Email	Role
8	Conor James	5/2/2021, 11:22:59	5/2/2021, 12:45:48	1h 22m	Conor.James@Office365	Attendee
9	Eoin Smith	5/2/2021, 11:22:59	5/2/2021, 12:45:48	1h 22m	Eoin Smith@Office365it	Attendee
10	Mark Hayward	5/2/2021, 11:22:59	5/2/2021, 12:45:48	1h 22m	Mark.Hayward@office3(	Attendee
11	James Ryan	5/2/2021, 12:02:24	5/2/2021, 12:37:15	34m 51s	James.Ryan@office365it	Attendee
12	Michael Conroy	5/2/2021, 11:22:59	5/2/2021, 12:45:48	1h 22m	Michael.Conroy@Office:	Attendee
13	Aimee Graces	5/2/2021, 12:07:39	5/2/2021, 12:45:44	38m 4s	Aimee.Graces@office36	Attendee
14	Tony Redmond	5/2/2021, 11:13:08	5/2/2021, 12:45:48	1h 32m	Tony.Redmond@office3	Organiser
15	Jane Sixsmith	5/2/2021, 11:22:59	5/2/2021, 12:45:48	1h 22m	Jane.Sixsmith@office36	Attendee

MSFT Attendance Meeting Report

Figure 8: Example output from Microsoft Teams meeting, Source https://www.jumpto365.com/blog/how-to-use-the-new-attendance-report-in-microsoft-teams-meetings

As we can assume that any students that are not contained in the CSV file were absent from the class, and the email column in the Students table in the database only holds unique values, the results from the parser can be used to take attendance. This is making the assumption that students will use these services with the same email address entered into the database.

When the file is uploaded, the list of students enrolled in the class is looped through and compared to the result of the parser. If the email of the current student in the class list loop is not present in the result of the parser, then their matriculation number is added to an array representing absent students, and if they do exist in parser results, the matriculation number is added to an array representing absent students.

This conversion from email to matriculation number using the data from the class list state means we can use the same function to record attendance as the checklist functionality. Originally there were two different functions on the server-side to add attendance to the database via matriculation number or by e-mail address. However the conversion to matriculation number and using the same function proved a much simpler and efficient method. This is because it was required to loop through the list of students anyway, as we needed to extract the necessary information to convert to JSON format. As the attendance table in the database requires the student matriculation number, it avoided two very similar queries on the server-side.

5.3.3 Attendance via QR Code

This function was added as part of the advanced functionality. This was indicated as a worthwhile feature from the user survey, and would also be useful in both in-person classes and classes conducted via video conferencing by sharing their screen. It also removes the need for the class coordinator to actually upload attendance.

A URL is created using the ID generated when the lecture is created. A QR generating component called ***React-QR-Code*** (React-QR-Code, 2020) then adds this URL as a props (property) and renders a custom QR code. When this is scanned it directs the user to a simple page where they can enter their student ID, which is dispatched to the server-side and entered into the attendance database if their student ID is present in the enrolment list for the class the lecture belongs to.

This then led to the question of how to handle students who have not attended. Obviously absent students will not scan the QR code, which will lead to them not being entered into the attendance database. This means their absence will not be accounted for when calculating their overall statistics. Therefore, a button was added for the class coordinator to confirm that the QR code is being used for attendance. This triggers a call to the server side that enters all students enrolled in the class as absent via a bulk create or update statement.

Bulk create or update was used to account for some user error where they may

distribute the QR code before pressing the confirm button. The Sequelize bulk create or update function allows you to address duplicate entries that may occur during the bulk create. This means that if a student has logged their attendance before the confirm button has been pushed, their entry will simply be skipped and their entry will not be overridden to absent.

5.4 Displaying Lists of Students

When displaying the lists of students, it was decided to create two tables in the UI. A class table for class coordinators and another table called a Super Table for departmental staff and advisors. These were implemented because class coordinators will be displayed grades and attendance statistics for their particular class, whereas these fields would not be required when viewing students in a department or a staff member's advisees. The departmental and advisee's table would also require more information such as year group and course.

The component used to display tabular data, as mentioned above, is called React Data Table Component. It comes with many features to improve user experience such as sorting and pagination. The sorting functionality was used, so a user can sort any column by ascending or descending order. The pagination feature was also initially used, however it proved to cause issues with the UI, as the font and icons were extremely small, smaller than the standard of at least 12px, and despite consulting the documentation, could not be changed size through the component or with CSS.

5.4.1 Class Student Table

When a class coordinator navigates to the page to display students for a given class, the class code is extracted from the route. This is then sent to the server side, where a *findAll* query is conducted

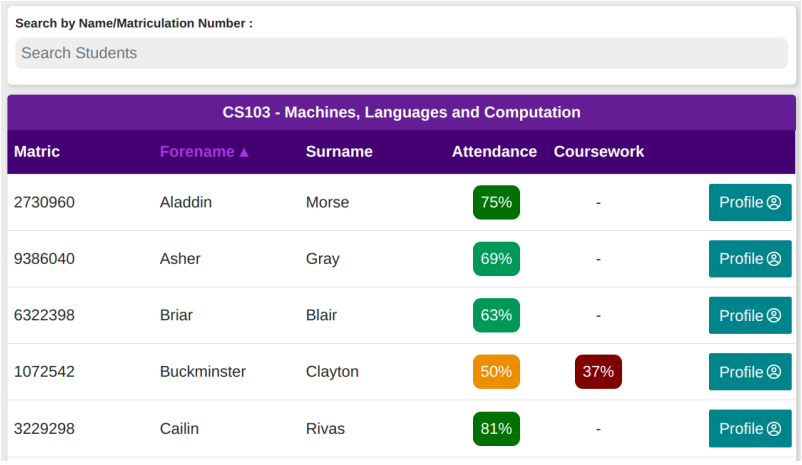


Figure 9: Class Table Page

to find all students who are enrolled in the class using the enrolment junction table. The attendance statistics and weighted grades from the respective views are included in this query. The result is sent to the client side as a JSON object and used in the datatable. Custom cells were added to the table to display the attendance and grades. They are wrapped in a *div* which, depending on the grade, is given the class “excellent” (over 80%), “good” (60%-80%), “warning” (40%-60%) and “danger” (under 40%). This is used to create a colour coding system in the table so that the user can easily identify students with concerning statistics.

Originally, SVG icons were used along with the colour coding system instead of displaying the statistic value. For example, a smiling face for “excellent,” and, “good,” and a warning symbol for, “warning,” and, “danger”. However, with this method, accounting for issues regarding colour blindness, meant that potentially a user could not tell the difference between, “warning,” and “danger”. Using more icons to represent each category resulted in the table looking cluttered, therefore it was decided to display the statistic with a relative background instead.

A button is added that links to a students class profile, which will be discussed later in this section.

To make finding a particular student as efficient as possible, a search bar was added. When a user enters a value into the search bar, the class list used to populate the data table is filtered to only include objects where the *matric, forename* or *surname* fields contain this value. The data table then re-renders to reflect this data.

Upon reflection, the algorithm to filter could have been implemented differently. The filter works by finding objects which contain the search term using the JavaScript method *includes()*. However this means that if a user were to, for example search, “s”, it will return all students with the letter “s” in their name. This information is probably not particularly useful and not what the user is looking for. Therefore, the filter function only becomes practical as you add more characters. An amendment that could be made is if the search term is a singular character, the filter function will return all students with forenames or surnames that begin with

that character. Then, for multiple characters use the *includes()* method.

The table shown above displays a lot of data, and during implementation it was too much data to display on smaller screens. Resizing the page, and using Chrome DevTools to emulate mobile/tablet devices shows that the table had many overflow issues. Therefore, using CSS Media Queries and functionality within the data table component, columns were omitted from the table and buttons were simplified as the screen size decreased. As a staff member is unlikely to know a students matriculation number off the top of their head, this column was the first to be omitted with the screen width reduced to below 956px, which is approximately the screen width of a landscape tablet.

The attendance and coursework columns were hidden for small screen sizes. This was because the forename, surname of the students and the link to their profile were important attributes that should be included in the table at all times, and it did not seem possible to include all five columns without compromising on important factors for usability such as the font-size and padding.

Through trial and error, the button was edited to remove the “Profile,” text and only includes a SVG indicating a profile at a certain screen width when overflow began.

5.4.2 Department and Advisees Table

Implementing the table for Departmental Staff and Advisors was relatively similar to the implementation of the class table, with a few differences. An action dispatches a call to the server side to get the list of students in a department or a list of advisees respectively. Unlike the class table above, class statistics are not included in the *findAll* query. Instead, it includes any entries in the View that contains the results of the stored query which finds any students who have an attendance of below 20% or coursework grades below 40% in more than one class.

The result of the query is sent back and stored in the state and used in another React Data Table Component. If a student has an associated entry in the View, then a warning sign is placed in a custom cell to indicate that their profile has some concerning statistics.

More information is included in this table such as what course and year a student is enrolled in, giving the user more filter options. Filtering by Name or Matriculation Number follows the same algorithm as above in the class table.

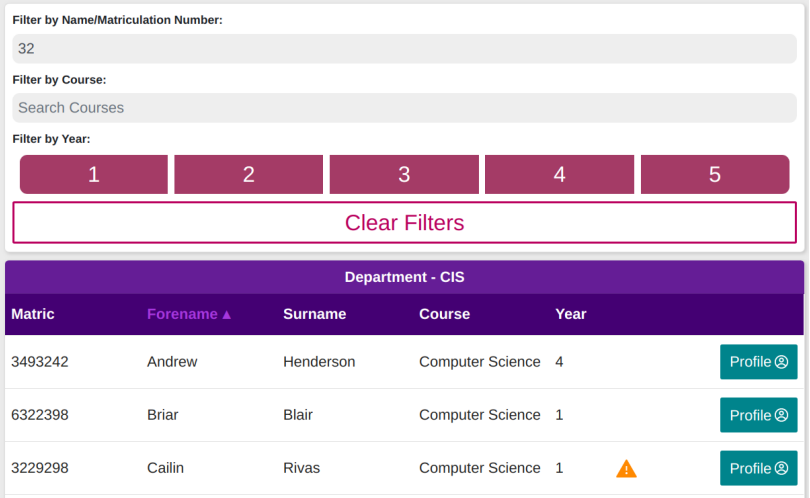


Figure 10: Department Students Page

Filtering by year was done by a React Bootstrap ButtonGroup which stores any selected buttons in the state which can be used in the filter algorithm. This can be configured to emulate either a radio button or checkbox which allows for multiple selections. The ButtonGroup was used because a search bar seemed redundant for what would only be single integer values, and avoid dropdowns. The checkbox implementation was originally used but it proved to be difficult to deselect, even when a clear filters button was added that completely erases the filter values in the state.

The radio button was used instead as it also seemed plausible that a user will be looking through a year group at a time. However, in future development multiple year selection would be an added feature.

In a similar way to the class table, this table was made usable across a variety of screen widths by gradually removing columns as the screen width reduced.

5.5 Student Profiles

There are two profiles available. The profiles available to class coordinators, and the profiles available to departmental staff, advisors and students. The latter contains all cross-module data about the student whereas the former is limited to a singular

class.

5.5.1 Class Profile

When the user navigates to a class profile, an action is dispatched that sends a request to find a particular students class information. A *findOne()* query is executed to find a student by their matriculation number, which is gathered from the route path of the student profile. Using Sequelize's association rules, any associated grades, attendances, weighted grades, average attendance and extenuating circumstances are included in the query. The result of the query is sent back as a JSON object which can then be used to compose the profile.

The profile is separated into two sections. There is one section for general information such as name, e-mail address and course, and another for statistics regarding coursework and attendance. This was done as the amount of information being displayed was growing large and separating the sections avoided any very long vertical scrolls. These two sections are navigated through two React Bootstrap Tabs.

The academic information tab includes personalised information in colour coded boxes. This is done in a similar way to the colour grading system in the student tables, where conditional boxes are rendered based on the statistic value. If any unattended lectures exist, these are also displayed.

5.5.2 Exporting Profile to PDF

This feature was included in the advanced implementation of the project. After some experimentation with technologies, this feature ended up being relatively simple to implement.

When the user clicks the button to export to PDF, the entire JSON object that was sent from the server-side to compose the profile is passed to a function *classPDFGenerator()*. Using the

Report for CS104	Issued on: March 22nd 2021
Name:	Cailin Rivas
E-mail:	Maecenas.libero.est@fringilla.net
Course:	Computer Science
Academic Year:	1
Department:	Computer and Information Sciences
Advisor:	Melyssa Crawford

Enrolments	
CS103	Machines, Languages and Computation
CS104	Information and Information Systems

Statistics	
Attendance	67%
Weighted Average Grade	12%

Coursework	Weight	Grade	Feedback
Lab	25%	12%	bad

Figure 11: PDF Produced for Student

jsPDF library, a new PDF document is created, and using its extended library jsPDF-Autotable, tables that reflect the information in the profiles are created by extracting the information from the JSON object again.

The PDF document is then saved and automatically opened in the users browser.

5.5.3 Super Profiles

Super profiles have been defined in the application as profiles that contain cross module data. These profiles are available to department staff, advisors and students.

The implementation of this type of profile is very similar to that of the class profiles. The main differences are that when the action dispatches to collect the data for the profile, it by association includes all classes that the student is enrolled in and collects the statistics for each class.

When the profile is displayed, the react method *map()* is used, which is very similar to the JavaScript *forEach()* method, to create Tabs for all the classes that the student is enrolled in. The JSON object contains an array of Objects which contains the relative information for each class. Therefore, for each class, the corresponding Object is passed as a prop to a React component *ClassOverview* which displays the information in a similar format to that of the class table above.

Due to the way the JSON object was structured from the query, it was not feasible to re-use the *AcademicInfo* component as above. This is because of the extra association for enrollment used to get the information for all the classes for a student. This led to some similar code between the *AcademicInfo* and *ClassOverview* components which is something that should be revisited in future developments to aid maintainability.

5.6 Notifications

To implement notifications, the information about the logged in user is collected from the state. There are two different actions, one for staff members and one for students. Depending on the user logged in, the appropriate action is dispatched to the server-side to get notifications.

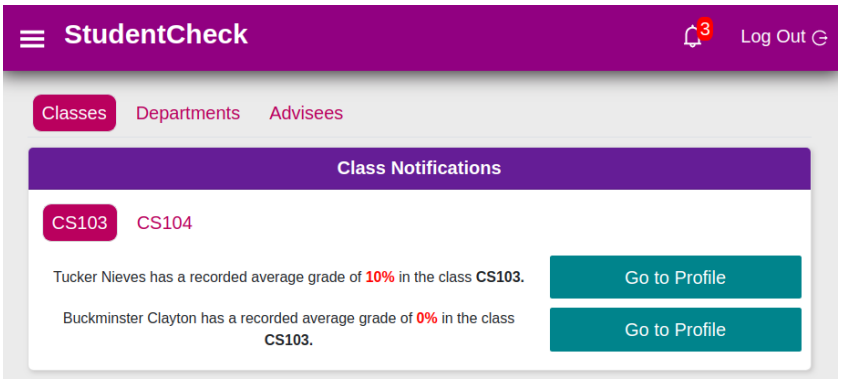


Figure 12: Staff Notifications Page

For staff members, due to the associations between Staff and Classes, Departments and Advisees (Students), a *findOne()* Sequelize query is carried out to find a staff member by the logged in user's ID. The associations allow for Departments, Classes and Advisees to be included in this query, along with any entries in the Low Grades, Low Attendance and Department Notifications Views.

A similar case is carried out for students, however only their corresponding entries in the Low Grades and Low Attendance Views are included.

This action is dispatched when the user logs in. There is then a set interval to dispatch this action every 5 minutes while the user is logged in to keep the notifications up-to-date.

The notifications in the state are looped through to count how many notifications there are and displays this in a bell icon in the header, along with a corresponding number of notifications. Clicking on this bell icon will take them to a page where they can then view the notifications in detail, and links to the appropriate profile of the student the notification is in regards to.

Push Notifications and Sockets were considered for this functionality, so that users

did not need to be logged in to be alerted if a student's performance dropped below a satisfactory level. This was not possible to implement in the timeframe available, but is a technology that could be explored in future developments.

5.7 Extenuating Circumstances

To implement the submission of issues that may be affecting a students performance, the user can navigate to a page either through a students profile if they are staff, or if the user is a student, they will be presented the option in their navigation and profile. When the page is navigated to, if it has not been stored in the state already, an action is dispatched that gets the types of circumstances in the *CircumstancesTypes* table, and through association the services entered in *StudentResources* linked to each circumstance.

The form is split into three sections. The first section allows the user to input the details of their circumstances. This is done with text inputs, and the React DatePicker component used throughout the programme. There is also a React Bootstrap Button group with a radio selection that the user must select to continue to the next page. This is populated with the circumstance types in the state. When the user selects an option, the selected options key is stored in a state.

When the user navigates forward, the selected key is used to display the services associated with the circumstance type selected, as the circumstance types are stored in a JSON object along with their associated resources.

Navigating forward offers the change for the user to review before submission. Upon confirmation another action is dispatched which puts the student ID and circumstance type along with the additional information added in the first step into the *ExtenuatingCircumstances* model.

5.8 Authorisation

When a user attempts to log in to the system, an action is dispatched that checks their username and password. If the login attempt is successful, a JSON web token is generated using a secret on the server side and the data about the user that indicates what pages they should have access to, such as classes they teach and

departments they are head of. This is sent back to the client side. This JSON web token is kept in Local Storage.

```
authenticateToken: function (req, res, next) {
    const token = req.headers.authorization;
    if (!token || token == null) {
        return res.status(401).send({ message: "No token"})
    }
    jwt.verify(token, process.env.JWT_SECRET = process.env.JWT_SECRET = "coffee", (err, decoded) => {
        if (err) {
            res.status(401).send({ message: "Token is not valid." })
        } else {
            req.user = decoded;
            next();
        }
    })
},

authenticateTeacher: function (req, res, next) {
    var class_code = req.query.class_code;
    if (!class_code || class_code == null) {
        class_code = req.body.class_code;
    }
    if ((!class_code || class_code == null) && req.body.params)  {
        class_code = req.body.params.class_code;
    }
    if ((!class_code || class_code == null) && req.query.params) {
        class_code = req.query.params.class_code;
    }
    if (req.user.classes.some(item => item.class_code.toLowerCase() == class_code.toLowerCase())) {
        next();
    } else {
        res.status(401).send({ message: "Not authorised access to this class!" })
    }
},
```

Figure 13 : Example of token authorisation for a teacher

Whenever the user navigates to a page that has restricted access, such as a class list of students, this JSON web token is sent along with the REST request. On the server-side, this token is verified to be a valid token and decoded. Providing it is a valid token, the content of the token is checked to ensure the user holds the correct credentials to access the data they are requesting. If they do not have the required credentials, a 401 (Not Authorized) status code is sent from the server with an appropriate message, otherwise the REST request will continue as normal.

If an error is sent, this error will be stored in the state by the reducer and this will be displayed to the user.

5.9 Routing

Another way to ensure that users cannot access resources they are not authorised to view was by setting up Private Routes specifically for Staff and Students. For example if a logged out user were to attempt the staff homepage, they will be

redirected to the login page.

Routing was used in the application to define path names which would render components. This was implemented using the ***React-Router*** library and documentation (React Router, 2020). To define private routes, the routes are wrapped in a component called for example, *StaffRoute.* This component takes the component that is to be rendered on the screen as a prop and all other properties such as the pathname are also collected using a property called rest.

There is a state in the Redux store that indicates if a staff member is logged in. This state is checked and if a staff member is logged in, the component is rendered, otherwise the user is redirected.

6 Verification and Validation

6.1 Testing Strategy

To compare the completed application with the requirements defined in the Software Specification, a series of tests were conducted. The majority of the testing was conducted with black box testing, however this was combined with testing the cross-browser compatibility, accessibility and responsiveness of the application. User testing was also conducted. Unit testing was considered, however due to the nature of the application, requiring user input and information from the database, it was very difficult to isolate code into units that work independently to test.

6.2 Black Box Testing

Black box testing is a testing method where mostly the functional elements of the application are tested, without knowing the internal structure of the code (Nidhra, 2012).

The Black Box testing was conducted by defining a list of tasks that relate to the requirements of the application. The expected output is noted, along with the description of why this test is being conducted. The result can be a pass if the actual output matches the expected output, or a fail along with the actual output from the application.

As the application ended up with a lot of functionality to test, the black box tests

results were vast. An example of the tests conducted can be seen below and the full list of tests are included in Appendix G.

Login Tests			
Activity Tested	**Expected Output**	**Description**	**Result**
Attempt to login at staff login tab with empty staff ID and empty password fields.	Error message is displayed asking for the user to input a valid staff ID and password.	To determine that if the user leaves both the staff ID and password field empty that the application will not attempt to log them in.	Pass
Attempt to login at staff login tab with empty password field.	Error message is displayed asking for the user to input a valid password.	To determine that the system recognises that a staff ID has been entered but that the password field is empty.	Pass
Attempt to login at staff login tab with empty Staff ID field	Error message is displayed asking for the user to input a valid staff ID	To determine that the system recognises that a password has been entered but the staff ID field is empty	Pass

6.3 Browser Compatibility

As a non-functional requirement of the system is to be as flexible as possible, all of he blackbox tests were carried out over a variety of browsers. Development was primarily on Chrome, but research was conducted on the most frequently used browsers. Included in Appendix H is a graph showing the current browsers with the highest market shares worldwide. It is clear that Chrome dominates this table, however there are other browsers that should be tested for compatibility, namely Firefox, Edge, Safari and Internet Explorer. Due to limited access to technologies

where Safari could be installed, an online service called LambdaTest was used which allows you to emulate different browsers. These results are also included in Appendix H.

The results from the test shows that over older versions of Chrome, Firefox, Edge and Safari, there were no issuesThere was however a major issue with Internet Explorer. Upon loading the page, the screen remains blank and the console log returns no errors.

After some investigation, the reason that the application was not running on Internet Explorer is because ES5 methods are not supported by IE. To mitigate this, libraries called ***React-App-Polyfill*** (React App Polyfill, 2020) and ***core-js*** (Core.js, 2020) were used. Polyfills allow for functions that are not supported by older browsers or browsers that do not support certain features to have the features filled out with methods to make them supportable. After using these libraries, the application was usable on IE11.

6.4 Responsiveness

Using the Chrome DevTools, the application was tested across a variety of screen sizes to ensure that there were no issues with the UI and that features that should not be available to mobile users, such as uploading CSV files are hidden.

For example, below are screenshots of the application on an 764px by 1024px iPad, and a 375px by 667px iPhone 6.

Overall, the application performed well across a variety of screen sizes. There was a slight issue with overflow with very small screen widths such as the iPhone 5 at 320px wide in the pages that contained tables of students. This meant that a small fraction of the profile button appeared to be cut off, however the button is still visible and usable.

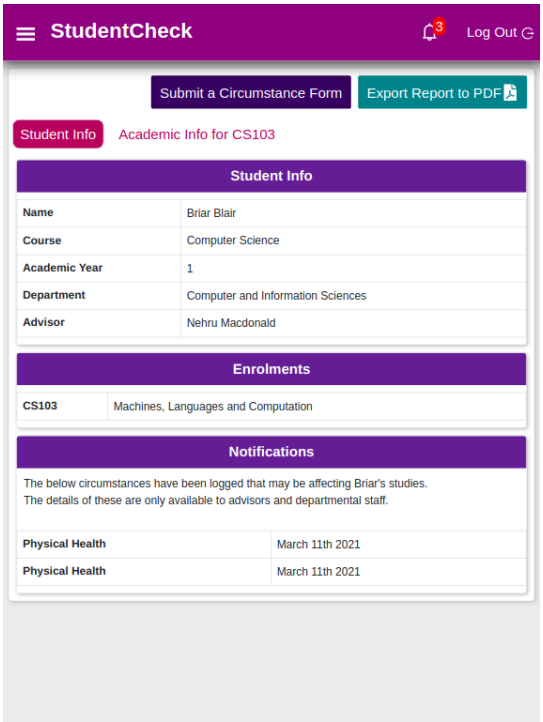


Figure 14 : Profile on iPad

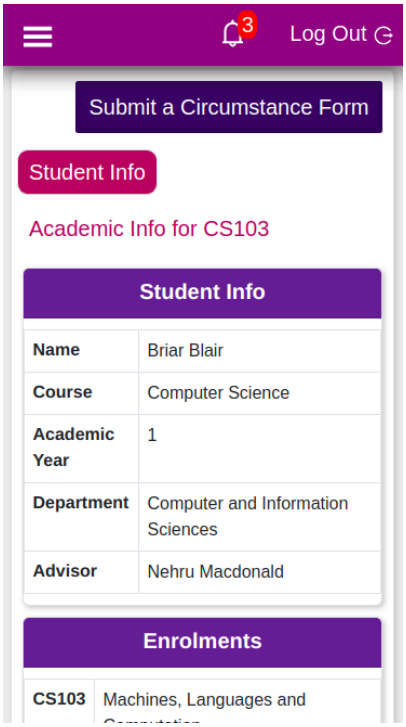


Figure 15 : Profile on iPhone

6.5 Accessibility

LightHouse was used to test accessibility, which scans webpages and creates a comprehensive report on the accessibility of the page and areas where it could be improved upon. It can highlight signs of poor accessibility such as poor colour contrast and incorrect aria-roles and labels. An example of this report is included which was conducted on a student's profile screen.

This was conducted on a variety of pages throughout the application, and the set of results can be seen in Appendix I.

Manual checks throughout the application were also conducted to ensure that the pages could be navigated through using the Tab key.

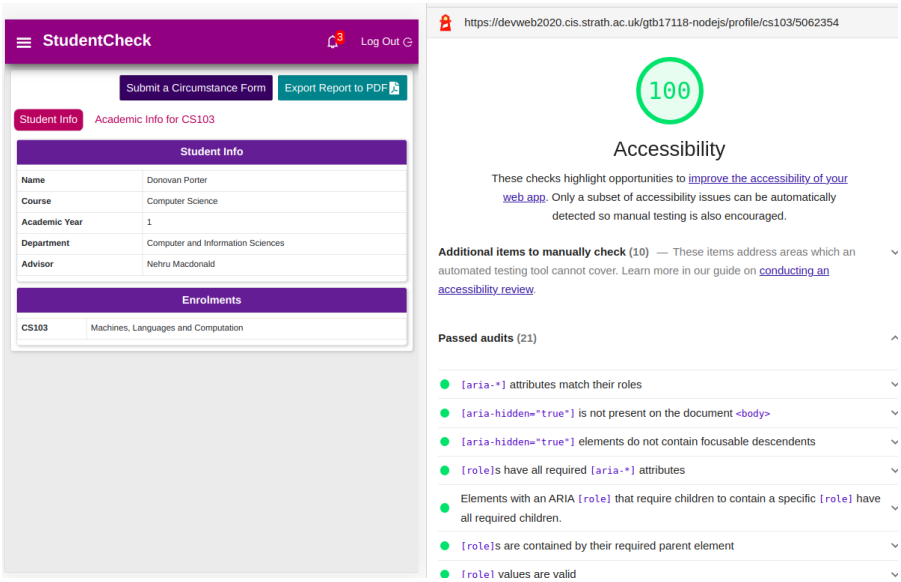


Figure 16 : Example of LightHouse Testing

6.6 User Acceptance Testing

As this application was developed for certain groups of users, it was decided that another user survey would be conducted to evaluate how the users find the application. During the survey, the users were asked to navigate to the StudentCheck application. They were then asked to complete a series of tasks and offer feedback on how effective the application was, and whether they ran into any issues during the process, to help determine bugs.

The survey was distributed to multiple different universities, departments and also to varying staff roles. Unfortunately, due to the timing of the survey and many staff members being very busy with running remote classes and preparing for exams, only 5 responses were recorded. However the responses received offered valuable feedback, so have been included in the report in Appendix J.

The survey revealed one issue, which turned out to be an indexing issue when counting the amount of notifications which lead to the system running very slowly

as it was stuck in a loop. To prevent this from happening to the rest of the participants so that feedback about the rest of the system could be evaluated, this was very easily fixed before re-deploying the survey.

Otherwise, much of the feedback for the application was positive, and many constructive features and improvements were suggested that could be addressed in further developments which will be discussed in the evaluation.

7 Results and Evaluation

This chapter describes the overall outcome of the project and the process of development. It will discuss how closely the project matches the functional and non-functional requirements of the system. It will also describe the feedback given during the user evaluation.

7.1 Final Product

The result of this project is an application for university staff and students. Referring back to the function requirements of the application:

- ✔ Coursework Management
  - ✔ Class coordinators *must* be able to upload coursework marks for a coursework in their class.
    - ○ **Class coordinators can create, edit and upload coursework marks. This can be done using a CSV file, or adding coursework marks individually.**
- ✔ Lecture Management
  - ✔ Class coordinators *must* be able to record attendance for a class.
    - ○ **Class coordinators can create and record attendance using a checklist, QR Code or CSV file.**
- ✔ Extenuating Circumstances
  - ✔ Staff and students *must* be able to upload any personal difficulties a student may be facing that is affecting their studies.
    - ○ **Students and Staff can submit an 'Extenuating Circumstances' form.**
- ✔ Student Performance
  - ✔ Class coordinators *must* be able to view statistics relating to a students performance for their classes.
    - ○ **Class coordinators can view profiles for students tailored to**

their class that includes the students grades and attendance data.

- ✔ Student advisors *must* be able to view cross-module statistics for students they are advisors to.
- ✔ Department heads *must* be able to view cross-module statistics for students within their department.
- ✔ Students *must* be able to view their own cross-module statistics.
  - ○ **Super profiles were created which contains cross-module data, but is limited to relevant departmental heads, advisors and students.**
- ✔ Students *must* be alerted if their performance is less than satisfactory for any of their classes
- ✔ Class coordinators *must* be alerted if a student's performance is less than satisfactory for their class.
- ✔ Advisors and Department heads *must* be alerted if a student's performance is less than satisfactory for more than one class
  - ○ **Logged in users notifications are periodically refreshed, and displays to the user whether performance is unsatisfactory.**

- ✔ Limiting Access
  - ✔ Users *must* only be able to access pages that they are authorised to view, for example class-coordinators must only be able to view the students in their classes.
    - ○ **This was achieved through the use of JSON Web Tokens to authorise access to certain data.**

The following requirements were met to a certain degree, however are relatively static and should be revisited in further developments.

- ✔ Personalised feedback and advice
  - ✔ Staff and students *must* receive personalised messages regarding the student's statistics.
  - ✔ Staff and students *must* be recommended student services depending on difficulties the student is facing that is affecting their studies.

Conditional messages are displayed to the user depending on the students statistics. Users are also recommended services based on their selections in the circumstance forms. This be improved upon which will be discussed in further developments.

A more detailed walkthrough of the program can be viewed in the user manual in Appendix K .

7.1.1 Staff

Upon successful login, staff are presented with a simple homepage with a header displaying their notifications, a logout button and a toggle containing a sidebar. The sidebar is dependent on the user signed in, but contains sections detailing classes they are class coordinators for, departments they are head of, and students they advise. This sidebar can be used to navigate around the application.

The user can choose to view coursework for a class. They can add a new coursework, edit an existing one or manage grades for students enrolled in the class. The managing grades section displays all students enrolled in a class and their grade, if it exists, in a colour coded table. The user can add coursework marks for students either individually or by uploading a CSV file from their device.

The user can also navigate to a section to take attendance for a given class. Here the user can add a new lecture session, and then they are presented with three options to take attendance. These options include a manual checklist, a QR code generated for their class and the option to upload a CSV file.

The user can navigate to Students in either of their classes, where they will be presented with colour coded indicators of the students average grade and attendance, and an option to search for particular students by their name or matriculation number. Departmental staff and advisors can also view the students in their department with appropriate filters, with a warning sign indicating whether the students performance is unsatisfactory in more than one class.

From these tables, staff can navigate to student profiles, which display their statistics and any circumstances they may be facing which may affect their studies. These profiles offer text descriptions based on the statistics, and also offers the user the option to add a note of circumstances that the student may be facing. During this, university services which may be able to aid their circumstances will be displayed. The user can also export the student profile to a PDF document. These profiles are either tailored to a specific class, or show cross-module data for

departmental staff or advisors.

The notification bell will display how many students have been counted in their classes, departments and list of advisees who have unsatisfactory average grades and/or attendance.

7.1.2 Students

When a student logs in, they are presented with their own cross-module profile, and a header with a notification bell which alerts them if their performance is unsatisfactory in any of their classes. They are also offered the option to submit a note of circumstances they are facing that may be affecting their studies, and an advice page which recommends them services based on the notes they have submitted.

7.2 Overall Evaluation

To evaluate the project, the application developed was compared to the requirements defined in the software specification.

Overall, the application offers functionality that meets the functional requirements defined and meeting the functional requirements was considered a success. Class coordinators can upload coursework marks and log attendance. Through the student tables and profile pages, class coordinators can view statistics for students in their class, and departmental staff and advisors can view cross-module data. Staff can also take note of circumstances that are affecting the student, and are suggested tailored services offered by the university that could aid these circumstances.

Students can also view their own cross-module statistics and submit any issues they may be facing. They are offered advice on student services that can aid their circumstances.

Both users are also offered notifications on whether a student’s performance is less than satisfactory and these notifications are tailored to the type of user logged in, such as students, class coordinators and departmental heads.

When the basic functional requirements were implemented, the application was expanded on to include further implementation such as uploading data via CSV files and adding search and sort functionality to the tables. These were implemented successfully, although as mentioned during the detailed implementation, the searching algorithm could have been improved upon. Overall, taking this into account, the implementation of the further implementation was considered a moderate success, and could be revisited in further developments.

Some features were implemented that were included in the advanced implementation, which were considered optional functionalities, that from the background research would be useful features to include. The features implemented fully included the ability to take attendance via QR code and to export the profile to PDF format. The implementation of these features were considered a success. A partial implementation of allowing the user to take attendance of classes where the entire class enrolment list may not be required to attend, such as labs and tutorials. This was considered a partial implementation as it is only available for the attendance taking option via checklist, and is not available with the method to upload via CSV file or QR code. Unfortunately due to time constraints, the algorithm to calculate overall student engagement and the automated email system were not implemented.

For non-functional requirements, the application was considered a success. From the testing stage it was demonstrated that the application was flexible, in that it is usable in different browsers and is responsive in design and usable across a variety of screen widths. By implementing the ORM, the application can be adapted to work with a variety of database management systems. By making the application a SPA and through the use of Redux and the Redux architecture, the frontend is also largely decoupled from the backend.

From a usability standpoint, the responses received from the user survey was largely positive regarding the user interface and ease of use. Some areas were noted that could be improved upon, such as clarity of the sorting functions. All five participants stated that the look and feel of the application was, “Very good”, and four out of five stated that this is an application that they would definitely consider

using in their role as university staff, with the other user stating that it's an application that they would probably use. The response for the clarity of the student statistics was also very positive. The full result and evaluation of the user survey can be seen in Appendix K.

The requirement that other authorised users should only be able to view certain pages was also met through the use of private routes and JSON web tokens. From a security standpoint, by using an ORM that uses parameterized statements which mitigate SQL injections.

Therefore overall, the project was considered a success, with some areas that could be improved upon from the user evaluation and in hindsight. Despite not all of the advanced features being implemented, these were not required to consider the project a success and provide areas to explore in further developments.

8 Summary and Conclusions

This chapter will reflect on the overall project development including the development methodology and any challenges that arose during the development. It will also discuss the future developments that could be explored.

8.1 Overall Reflection of the Project

While the project was considered a success, there were some definite areas of improvement discovered during the development of the application

The initial design stage and research into technologies could have been improved upon. Despite going into implementation with a strong idea of what the application would look like, the technologies being used and the structure of the application, it was discovered that a lot of time was being consumed on various tasks. That led to a complete overhaul of technologies being used on the server-side to implement the ORM. While the ORM was definitely a worthwhile technology to implement in the long run that sped up development, it required a lot of refactoring of the server-side that had already been implemented. It also meant that the design stage had to be revisited which meant that the original Waterfall methodology turned into an Agile approach.

The design stage was also revisited in terms of the UI. The designs for the UI were changed multiple times as it was discovered that more information would need to be included, making the UI cluttered and difficult to read. Overall, the final implementation of the UI was considered successful and was responsive and visually appealing.

Due to the pitfalls in the design stage, a lot of time was spent on the implementation as many changes were made throughout the process, resulting in a cascading effect on the testing, evaluation and documentation stage. Also in hindsight, too much focus was on implementing as much useful functionality as possible which also led to a lot of changes in the design stage.

The user surveys could have explored more about the user experience, although fortunately the participants were very forthcoming with general feedback that proved very useful.

These were all very useful learning experiences, and the project allowed for experience in a variety of new technologies such as React, ORMs and NodeJS and the opportunity to learn about new ways to structure code such as Redux was also very beneficial. As these technologies are very often used in industry, these learning opportunities will mostly likely be very useful in future projects.

8.2 Further Developments

As mentioned, some features could be revisited such as the search algorithm could be improved. The extra features such as the engagement calculation and the automated email system could be implemented.

8.2.1 Database Management

The application works with a dataset stored in the database. As the ORM has sync functions to create a database on the database management system of choice, the application would currently work with the system being synced to the database management system, and then administration can manage the data through an administrative tool such as PHPMyAdmin.

The reasoning for this was because there are many administrative tools available

and this allowed university administrators to determine which to use, or if they were accustomed to a certain management system then they would not need to acclimatise to using StudentCheck to manage the database. This however, was not the best decision and another feature that should be added is the ability to manage the database fully from the application after the database structure has been synced from the ORM. For example, an administrative account could be added where they can bulk upload students and staff and assign enrolments to students via CSV, and assign teachers to classes and departments. The application could also be extended to other types of user such as heads of year.

Functions could be added to the server-side that could check and clean up redundant entries. For example, lectures that do not have any corresponding attendance.

8.2.1 Recommendations

The resources recommendation could also be more personalised. The recommendations are relatively static, but another table could be added for extreme or multiple cases. For example, if the student experiences multiple illnesses in a short amount of time they could be referred to their department head for possible extension or repeated study.

A feature that could be added in the long-term if the application was used over a large enough set of students over a period of time, is the statistical data of students who end up withdrawing from their studies. This could be used to see if there was a correlation between attendance, grades and extenuating circumstances and withdrawal from studies. This could then be used to predict if a student is showing signs of potentially withdrawing from studies.

8.3 Conclusion

Overall, the completion of this project has been a very satisfying process, and a lot of techniques and new technologies have been discovered which can be used in the future. The positive response from the user survey showed that the application can be useful for the target group which is a huge achievement.

Through the research, testing and evaluation conducted, a flexible application that

can be used to help staff and students to reflect on a student's performance has been implemented and services available to the student can be recommended. This can hopefully help student engagement and retention, so students can gain more from their studies.

Bibliography

References

Bowen, Eleri, Price, Trevor, Lloyd, Steve, and Thomas, Steve. "Improving the Quantity and Quality of Attendance Data to Enhance Student Retention." *Journal of Further and Higher Education* 29.4 (2005): 375-85. Web.

Chrome DevTools (2020) Retrieved from https://developer.chrome.com/docs/devtools/

Core.js (2020) Retrieved from https://www.npmjs.com/package/core-js

CSS (2020) Retrieved from https://www.w3.org/TR/CSS/#css

DatePicker (2020) Retrieved from https://www.npmjs.com/package/react-datepicker

Garrett, Jesse James. The Elements of User Experience [internet Resource] : User-centered Design for the Web and beyond. 2nd ed. 2011. Voices That Matter. Web.

Illovsky, Michael E. "Effects of Counseling on Grades and Retention." *Journal of College Student Psychotherapy* 12.1 (1997): 29-44. Web.

Jarrett, Caroline., and Gerry. Gaffney. *Forms That Work : Designing Web Forms for Usability*. San Francisco, Calif. : Oxford: Morgan Kaufmann ; Elsevier Science [distributor], 2008. Print. Interactive Technologies.

JsPDF (2020) Retrieved from https://parall.ax/products/jspdf

jsPDF-Autotable (2020) Retrieved from https://simonbengtsson.github.io/jsPDF-AutoTable/

JWT (202) Retrieved from https://jwt.io/

Lighthouse (2020) Retrieved from https://developers.google.com/web/tools/lighthouse

Millea, Meghan, et al. "What Matters in College Students Success? Determinants of College Retention and Graduation Rates" *Education*, vol. 138, no. 4, 2018, p. 309+. *Gale Academic OneFile*, link.gale.com/apps/doc/A543610935/AONE?

u=ustrath&sid=AONE&xid=64f669b9.

Mohamed Basheer, K. P, and Raghu, C. V. "Fingerprint Attendance System for Classroom Needs." *2012 Annual IEEE India Conference (INDICON)* (2012): 433-38. Web.

Moodle (2020) Retrieved from https://moodle.org/

Neves, Jonathan, Hilman, Nick, “The 2016 Student Academic Experience Survey”, *Higher Education Policy Institute.* Web, available at https://www.hepi.ac.uk/wp-content/uploads/2016/06/Student-Academic-Experience-Survey-2016.pdf [Accessed 27th Nov 2020]

Nidhra, S., 2012. Black Box and White Box Testing Techniques - A Literature Review. *International Journal of Embedded Systems and Applications*, 2(2), pp.29-50.

Node.js. (2020) Retrieved from https://nodejs.org/en/

PapaParse (2020) Retrieved from https://www.papaparse.com/

Paul, Akshat., and Abhishek. Nalwaya. React Native for Mobile Development [internet Resource] : Harness the Power of React Native to Create Stunning IOS and Android Applications. 2nd Ed. 2019.. ed. 2019. Web.

Puppeteer (2020) Retrieved from https://pptr.dev/

Microsoft Teams (2020) Retrieved from https://developer.microsoft.com/en-us/microsoft-teams/docs

Zoom (2020) Retrieved from https://marketplace.zoom.us/docs/api-reference/introduction

React Bootstrap (2020) Retrieved from https://react-bootstrap.github.io/

React Data Table Component (2020) Retrieved from https://www.npmjs.com/package/react-data-table-component

React App Polyfill (2020) Retrieved from https://www.npmjs.com/package/react-app-polyfill

React Redux (2020) Retrieved from https://react-redux.js.org/

React Router (2020) Retrieved from https://reactrouter.com/

React-QR-Code (2020) Retrieved from https://www.npmjs.com/package/react-qr-code

Rissanen, Anna. "Student Engagement in Large Classroom: The Effect on Grades, Attendance and Student Experiences in an Undergraduate Biology Course." *Canadian Journal of Science, Mathematics and Technology Education* 18.2 (2018): 136-53. Web.

Redux. (2020) Retrieved from https://redux.js.org/

Sanderson, Steven. Pro ASP.NET MVC 2 Framework. 1st ed. Berkeley, CA: Apress, 2010. Web.

Sequelize. (2020) Retrieved from https://sequelize.org/

Solovei, V, Olshevska, O, and Bortsova, Y. "The Difference Between Developing Single Page Application and Traditional Web Application Based on Mechatronics Robot Laboratory Onaft Application." *Automatizácia Tehnologičeskih I Biznes-processov* 10.1 (2018): Automatizácia Tehnologičeskih I Biznes-processov, 2018-04-09, Vol.10 (1). Web.

WCAG Guidelines : Retrieved from https://www.w3.org/WAI/standards-guidelines/wcag/

Learning Resources

To familiarise with the technologies used in the project the following resources where used:

Learning React, Node and React Redux :

https://www.udemy.com/course/build-ecommerce-website-like-amazon-react-node-mongodb/

Code from this article aided in deployment to DEVWEB :

https://www.digitalocean.com/community/tutorials/nodejs-serving-static-files-in-express

Appendices

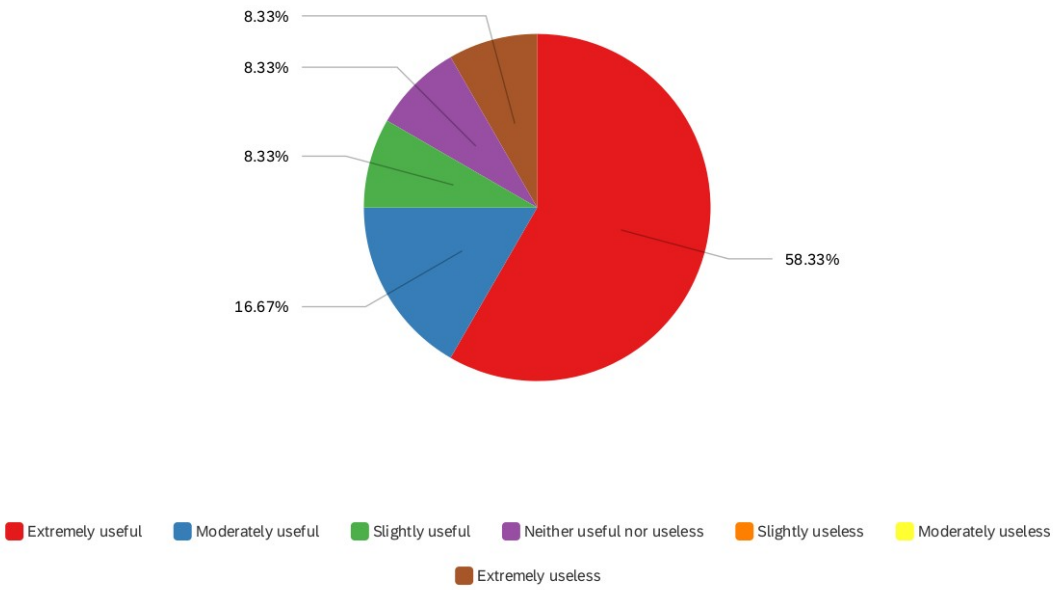
Appendix A - Background Research User Survey Results

This appendix will show all the questions asked in the user survey, as well as the collated results from the participants.

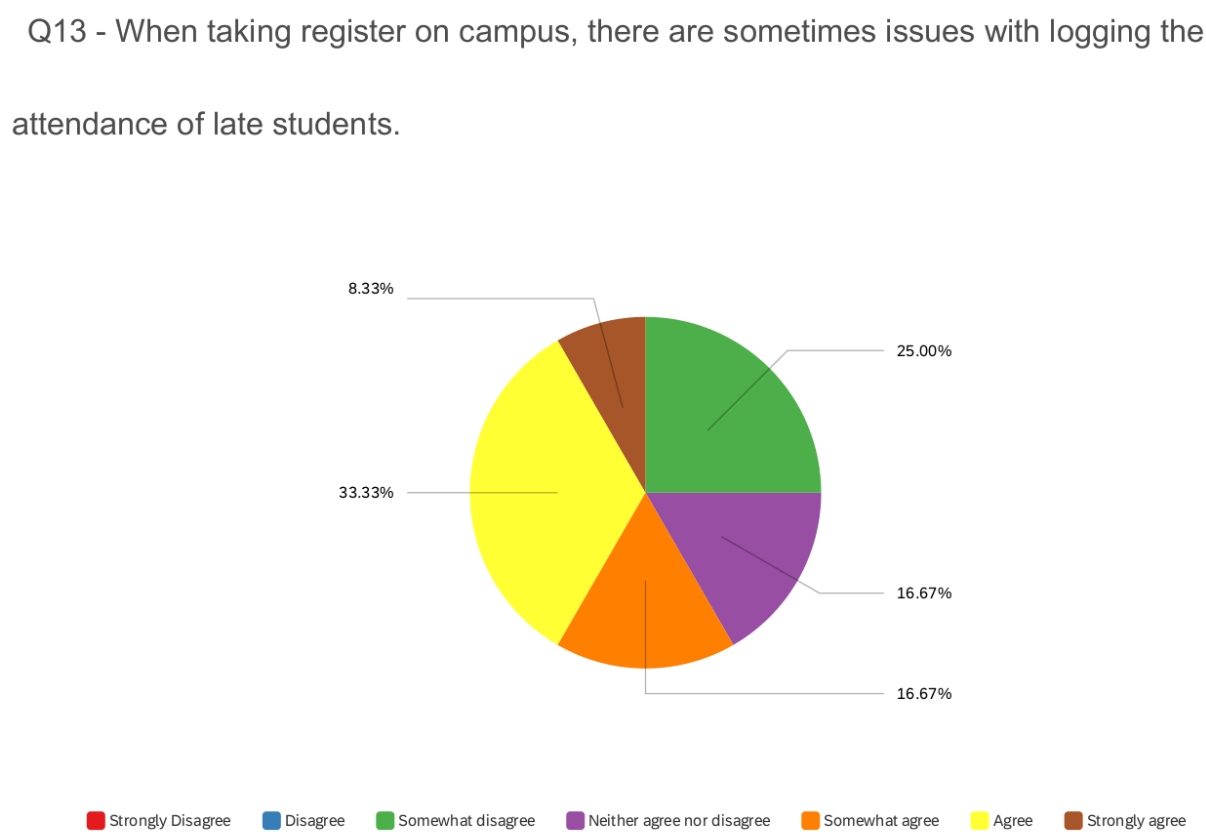
All questions have been added in order that they were asked. The question numbers appear disjointed, however this is just due to an indexing error in the survey as some questions were written, then rearranged or deleted when the survey was being designed.

The results and how they affected the project specification, and areas that could have been improved upon will be discussed in this section.

Q1 - Would you be interested in using an application that allows you to take attendance during class.

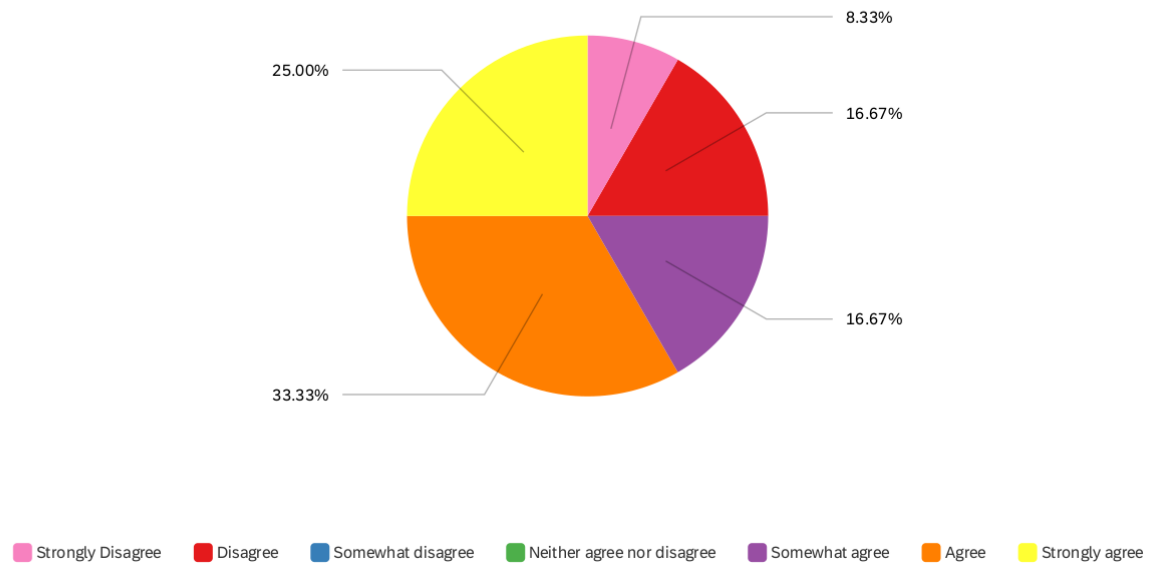


The aim of this question was to solidify the findings of the background research and to affirm that the project should contain features that allow the user to take attendance during class, which this question did as the majority of the participants responded that such a feature would be useful.



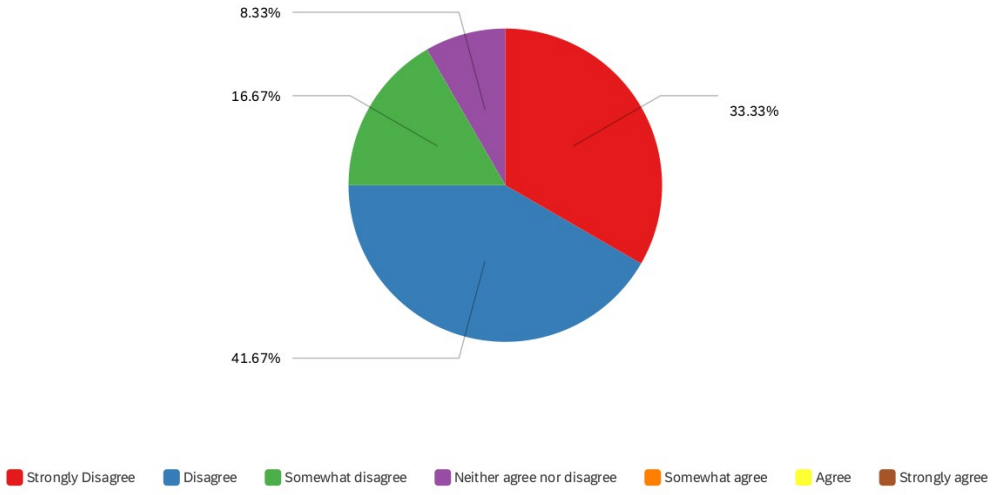
In hindsight this question was not too insightful, but it was asked to see if a method that allowed for late students to be accounted for was worth implementing in this development cycle. As the results were mixed, this was not treated as an issue that needed to be addressed urgently but could be addressed in further developments.

Q14 - When taking register on campus, there are sometimes issues with the register circulating the room during the duration of the lecture.

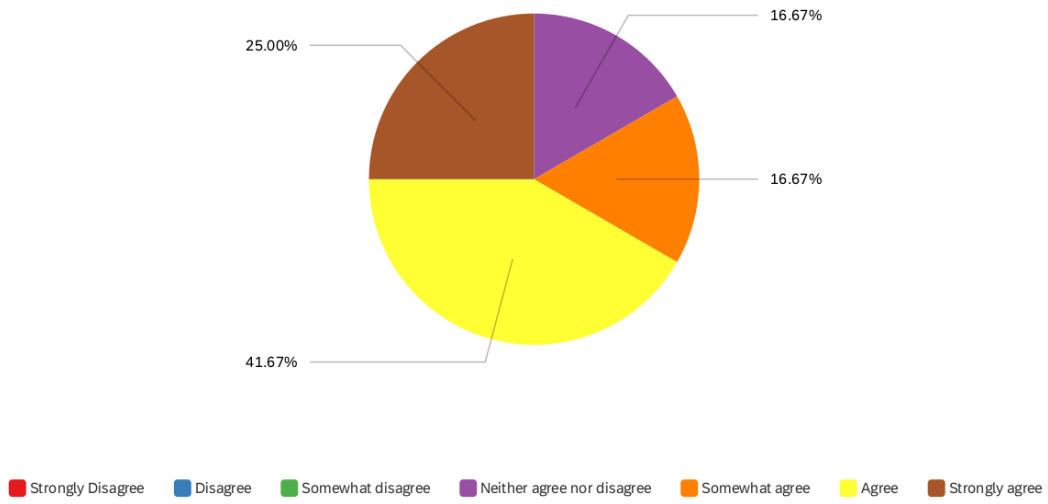


This was another question that was asked to confirm the findings in the background research. As the majority of users said this was an issue that they encountered there will be focus on a method to take attendance that is quick to use for a large class.

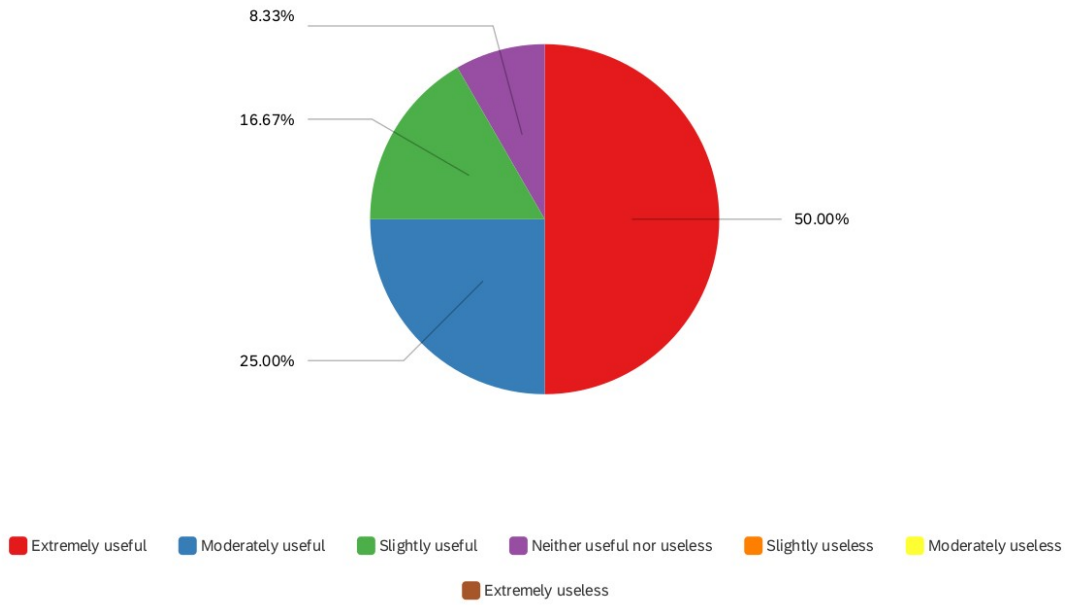
Q15 - I can currently easily view students' coursework marks and attendance across all classes.



Q16 - It is/would be beneficial to be able to view a student's coursework marks and attendance across all classes to monitor student's overall performance.

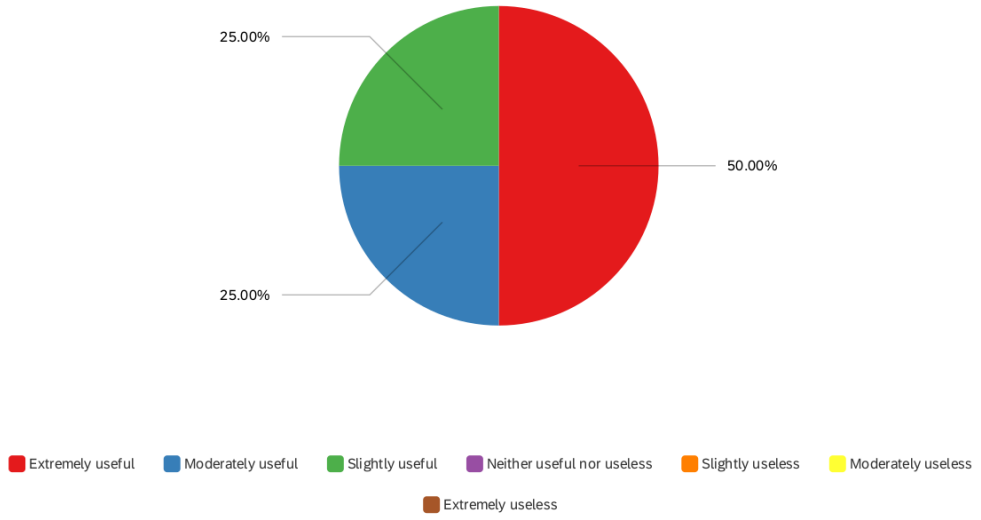


Q25 - Individual student profiles that display their coursework grades, feedback and attendance across all modules.

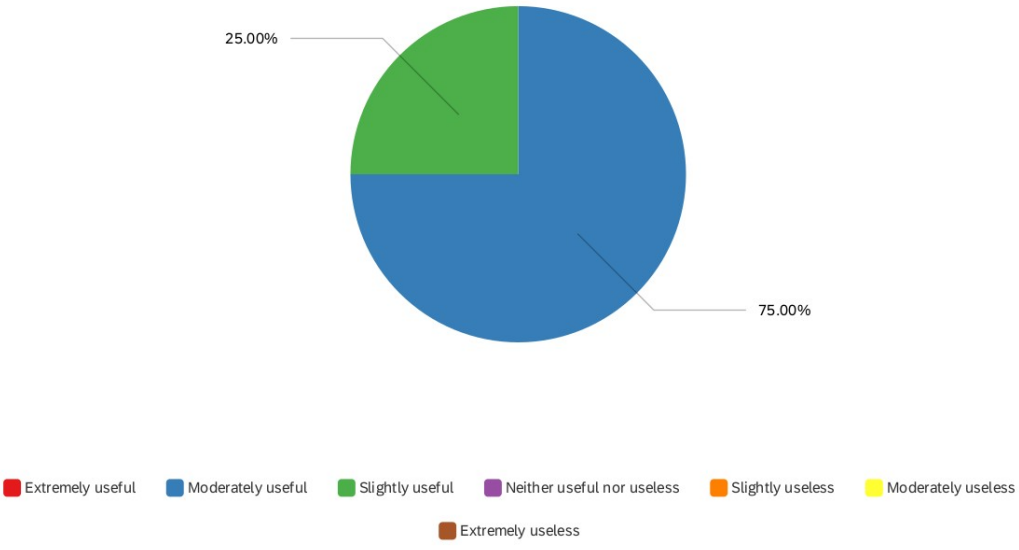


These questions were asked to gauge whether staff can easily access cross-module data, and whether it would be/is beneficial. From the respondents the majority disagreed that they can easily view the kind of data, but the majority of the respondents believed it would be useful. However, later there was some feedback to show that while this may be useful, allowing staff members to view cross-module data should be moderated.

Q20 - Ability for students to log their own attendance in class with their smartphone device (both on campus and virtually) via a QR code or PIN.

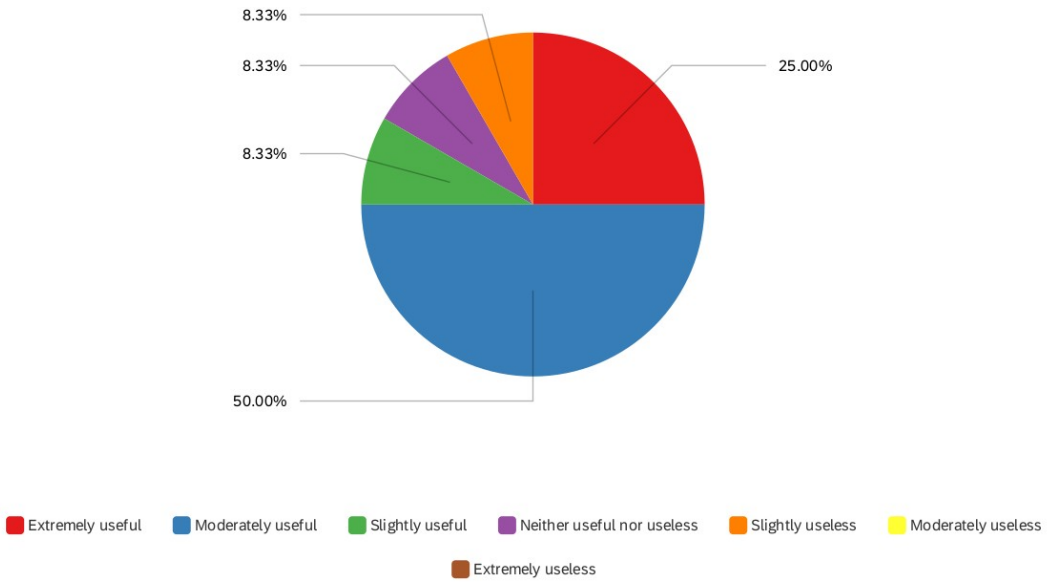


Q19 - Ability to export a student's performance report to a .doc, .pdf or other file.



As these features received overall positive responses, this was added with high importance to the advanced implementations of this development cycle

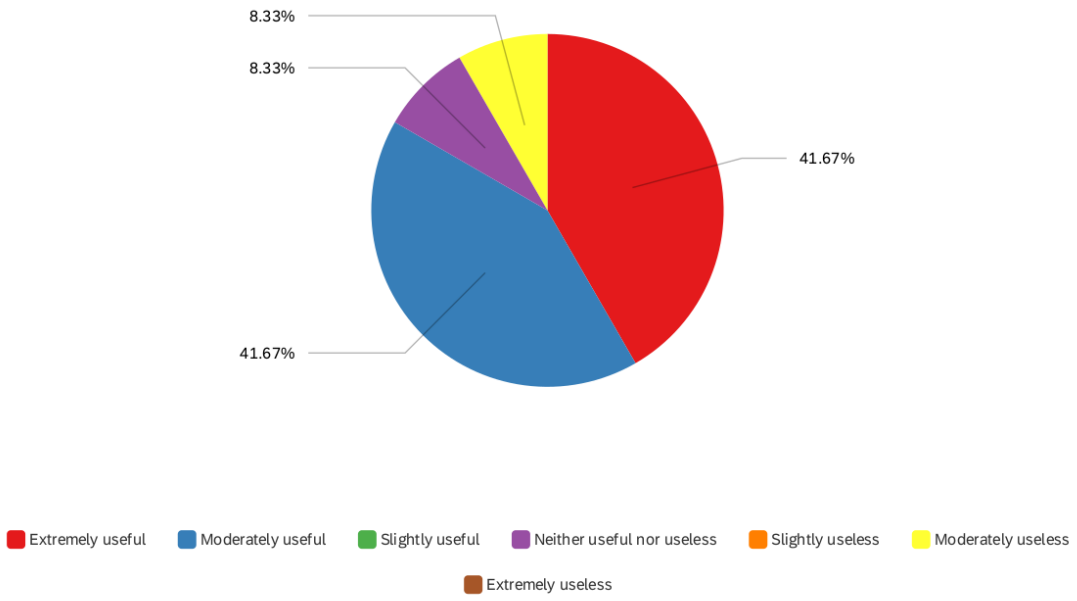
Q21 - Ability for lecturer to have an account to view their classes and students within the classes.



.

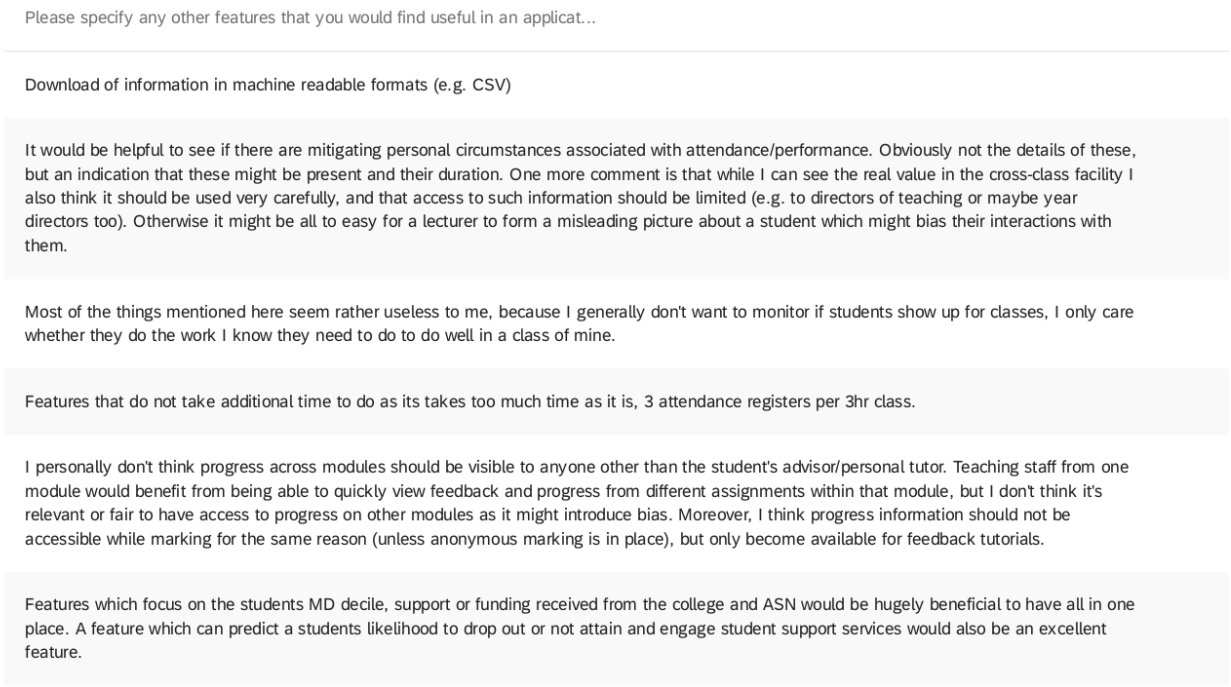
This question could have been worded differently. This was to gauge whether it would be useful for the staff member to have an account so that they can easily view the student statistics for students in their class. This is however addressed in the feedback and was added to the basic implementation.

Q22 - Ability to upload student attendance for virtual lessons via the .csv file of participants generated by video conferencing clients such as Zoom.



This was added to account for the switch to online deliverables and video conferencing lectures. The majority of respondents said that this feature would be useful and there would be added to the application after basic implementation.

Q26 - Please specify any other features that you would find useful in an application to record student progess.



Please specify any other features that you would find useful in an applicat...
Download of information in machine readable formats (e.g. CSV)
It would be helpful to see if there are mitigating personal circumstances associated with attendance/performance. Obviously not the details of these, but an indication that these might be present and their duration. One more comment is that while I can see the real value in the cross-class facility I also think it should be used very carefully, and that access to such information should be limited (e.g. to directors of teaching or maybe year directors too). Otherwise it might be all to easy for a lecturer to form a misleading picture about a student which might bias their interactions with them.
Most of the things mentioned here seem rather useless to me, because I generally don't want to monitor if students show up for classes, I only care whether they do the work I know they need to do to do well in a class of mine.
Features that do not take additional time to do as its takes too much time as it is, 3 attendance registers per 3hr class.
I personally don't think progress across modules should be visible to anyone other than the student's advisor/personal tutor. Teaching staff from one module would benefit from being able to quickly view feedback and progress from different assignments within that module, but I don't think it's relevant or fair to have access to progress on other modules as it might introduce bias. Moreover, I think progress information should not be accessible while marking for the same reason (unless anonymous marking is in place), but only become available for feedback tutorials.
Features which focus on the students MD decile, support or funding received from the college and ASN would be hugely beneficial to have all in one place. A feature which can predict a students likelihood to drop out or not attain and engage student support services would also be an excellent feature.

The feedback from the survey was very interesting and would impact the implementation drastically. It had not been considered the issue of bias when viewing cross-module data and this was pointed out by two respondents. Therefore, this was taken into account for the software specification that access to student statistics should be restricted to certain personnel.

Feedback also included that features should be quick to use so this would emphasised in the non-functional requirements.

Extra features that were suggested in this survey were algorithms to determine how likely students are to withdraw from studies and exporting to machine readable formats such as CSV, which would be added to extra implementations.

Appendix B – Agile Methods

Appedix B.1 – User Stories

As a result of the background research and to help determine the functional and non-functional requirements of the system, user stories were produced with their corresponding acceptance criteria.

A user story is a requirement from the perspective of the end user of a system, which describes a desired functionality. From this, a list of requirements can be produced to ensure that all end users needs are met under different scenarios.

Some examples of user stories with their respective acceptance criteria for the basic implementation for this project are defined below.

- As a class coordinator at a university, I must be able to record the attendance of my classes, so that I can view attendance data for students at a later date.
    - Given I am class coordinator.
    - I can navigate to a page, “Attendance”, for my class.
    - I can input a time and date for a class that I have held or am currently holding.
    - The system will present me a list of students that are listed to attend this class.
    - I can select students who were/are present at the class.
    - I can click a button, “Submit Attendance”.
    - I am presented with a confirmation that the attendance has been successfully recorded.
- As a class coordinator at a university, I must be able to add coursework for my class, so I can add student’s grades at a later time.
    - Given I am a class coordinator.
    - I can navigate to a page, “Manage Coursework”, for my class.

- I can click a button, “Add Coursework”.
- I can input a title, description and weight contributed to coursework total
- I can click a button “Add Coursework”
- I am presented with a confirmation that the coursework has been successfully added.

- As a class coordinator at a university, I must be able to add a student’s coursework grade, so I can view these grades at a later date.
  - Given I am a class coordinator.
  - I can navigate to a page, “Manage Coursework”, for my class.
  - The system will present me with a list of coursework titles for my class.
  - I can click a button, “Add Grades” for the coursework that I wish to add the grade for.
  - The system will present me with a list of students who are enrolled to complete this coursework.
  - I can click a button, “Manage Grade”, for the student I wish to add a grade for.
  - The system will present me with a form where I can add a grade and feedback.
  - I can click a button, “Submit Grade”.
  - I am presented with a confirmation that the grade has been successfully recorded.

Appendix B.2 – Original Project Plan

1st October - 1st December

- Background research about attendance data, coursework grades and extenuating circumstances
- Research technologies that are suitable for the application
- Conduct initial user survey
- Write background research and project specification chapters

1st December - 4th January

- Create prototype with hardcoded data which addresses basic functionality
- Construct database
- Continue writing report, focusing on System Design

4th January – 22nd January

- Integrate database with prototype
- Continue writing report, focusing on Detailed Implementation
- Start research to address advanced features and extra implementation

22nd January - 14th February

- Implement advanced and extra features feasible within timeframe
- Research most suitable testing strategies
- Plan user evaluation
- Finish detailed design chapter of report

14th Febrary - 8th March

- Conduct testing on finished implementation
- Write and deploy user evaluation
- Write testing portion of report

8th March - 22nd March

- Evaluate project and interpret user evaluation
- Finish draft of report

22nd March – 29th March

- Fine tune report
- Project submission

Appendix B.3 – Amended Project Plan

Due to issues such as adjusting to changing technologies, deployment to DEVWEB, adjusting to online learning and illness, the plan ended up changing and was conducted as follows:

1st October - 1st December

- Background research about attendance data, coursework grades and extenuating circumstances
- Research technologies that are suitable for the application
- Conduct initial user survey
- Write background research and project specification chapters

1st December - 16th December

- Minimal work on prototype during exams.

16th December - 4th January

- Create prototype with hardcoded data which addresses basic functionality

4th January – 22nd January

- Construct database
- Continue writing report, focusing on System Design
- Start research to address advanced features and extra implementation

22nd January - 14th February

- Integrate database with prototype using ORM
- Continue writing report, focusing on Detailed Implementation
- Plan user evaluation
- Finish detailed design chapter of report

14th Febrary - 8th March

- Research most suitable testing strategies
- Implement advanced and extra features feasible within timeframe
- Write testing portion of report

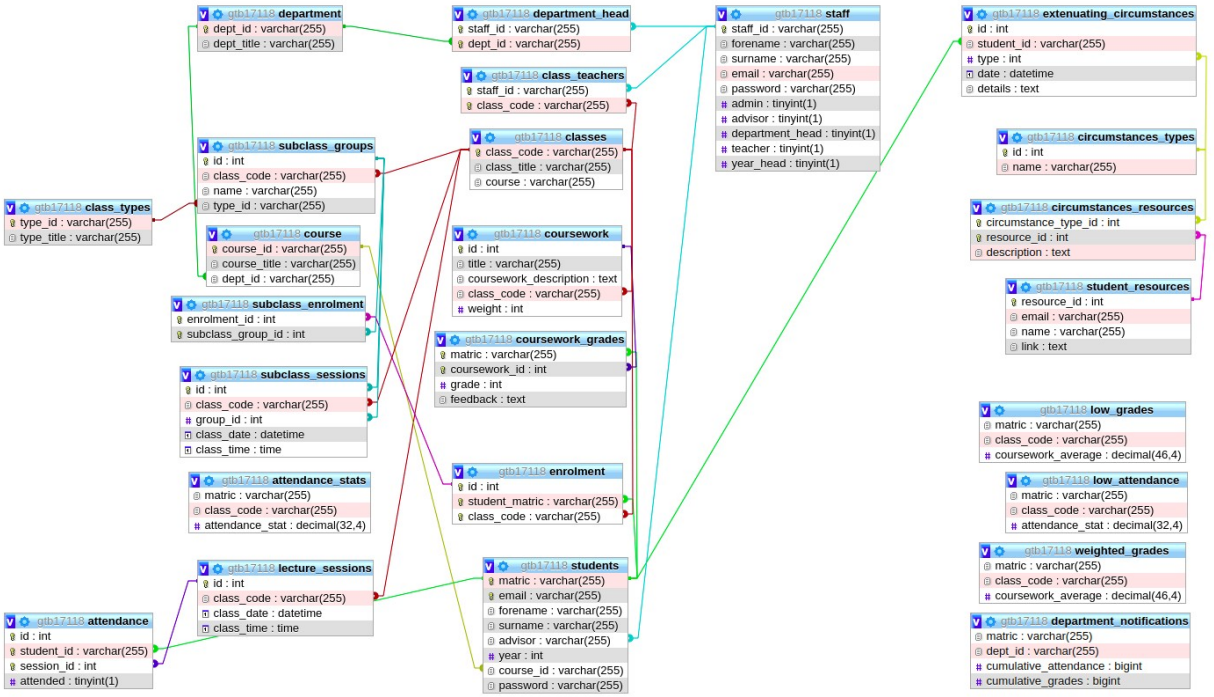
8th March - 22nd March

- Conduct testing on finished implementation
- Write and deploy user evaluation
- Evaluate project and interpret user evaluation
- Finish draft of report

22nd March – 29th March

- Fine tune report
- Project submission

Appendix C - Database Design



Appendix C - Overview of Database Structure

As the database became very large, it was difficult to clearly show all the relationships between the tables. The colour coded lines show the relationships between each table.

This table was generated using the *sync()* function from the Sequelize ORM and the models in the server-side. Before implementing the ORM, this database was constructed locally, and the models were defined by the already constructed database. When the project was deployed to DEVWEB, the database also needed to be transferred. This was done simply by deploying and running the server-side code.

Appendix C.1 Tables

Appendix C.1.1 Staff

As the main user of this system is university staff, this was the first part of the database to be designed. It was assumed that we have access to a list of university staff who have a unique staff ID, name and a contact e-mail address. It was also assumed that we know which classes a staff member teaches, which students they act as advisors to and whether they are the head of a particular department.

A table was constructed to reflect this data with the Staff ID acting as the Primary Key to the table. Columns with the type boolean were added to the table indicating whether they are a teacher, advisor or department head. These were to be used to implement a personalised UI and for permissions to pages, however these columns were later not used as an approach was implemented which would eliminate the need to add these columns.

Appendix C.1.2 Classes

A table for classes taught within the University was added with the assumption that each class has some unique code, for example “CS408”. This class code will act as a Primary Key for the table.

As university staff can teach multiple classes and a class can potentially have multiple teachers, this indicated a many-to-many (M:N) relationship. This is not usually supported in relational databases, however after research into database normalisation, it was decided that to represent the class-teacher relationship a further table would be added to act as a junction table with fields for the Staff ID and class code, with foreign keys to their respective tables.

Appendix C.1.2 Departments

Similarly to the classes, it was assumed that there are departments with unique department identifiers, for example “CIS” for the Computer & Information Sciences department. This was added to a table, and again similarly to the Classes table, a junction table was added to represent the M:N relationship between Head Departmental Staff and Departments.

Appendix C.1.3 Students

It was assumed that students have a unique identifier such as a matriculation number, which will act as the Primary Key for the table. We also have basic information about the student such as their email address, year and the course they are enrolled in.

Students are assigned an advisor. The creation of another junction table for Students and Staff to indicate this relationship was considered, however a student will most likely only have one advisor at any one time, and staff may have many advisees, this indicates a 1:M relationship. Therefore, a field in the students table that is a foreign key to the Staff primary key was sufficient whilst also avoiding any data redundancies.

However, a student will most likely be enrolled in many classes at any time, and classes will of course have more than one student, indicating a M:N relationship. A junction table for enrolments was therefore created, with fields for student ID and class code was created with respective foreign keys.

Appendix C.1.4 Coursework and Attendance

The coursework table will have an auto-incremented Primary Key. This was decided as coursework will be added by the user throughout the academic year, and it eliminates the need for the user to generate a unique identifier for the table themselves. This will also prevent any collisions with Primary Key values. Another table will be added to represent the M:N relationship between coursework and students with added fields for the grade and feedback.

A field for the amount of weight this coursework holds will be included in the coursework table. This will be used to calculate a weighted grade for students, which is a more accurate indicator of performance rather than a simple average.

Similarly, a class sessions table will be added with an auto-incremented Primary Key, with another table to represent the M:N relationship between students and class sessions, and a boolean field showing whether they attended the class.

Appendix C.1.5 Extenuating Circumstances

A table containing types of circumstances that a student may encounter throughout their studies will be created along with another table which stores the resources available to students. A junction table was made between different types of circumstances, and student resources to indicate which services aid which circumstances. For example, the Student Finance Department can aid students who are experiencing financial difficulties. This means that students can then add circumstances, and be offered the relevant services depending on the type of circumstance.

Appendix C.2 Views

A view is a virtual table which contains the results of a stored query. As the application is rating students performance, it was evident that there would be calculations based on the data stored about each student. Rather than writing functions within the application or having to re-write complicated queries, views were constructed for values that would be needed throughout the application, which are based upon calculations of student data held within the database.

Appendix C.2.1 Attendance and Coursework Grade Views

A view was constructed to calculate a students attendance at a given class. As the attendance is represented as a boolean in the attendance table, indicated by either a ‘1’ for true, or ‘0’ for false, the calculation can be made easily by the sum of attendances of a student for lectures belonging to a certain class, divided by the count of these entries multiplied by 100 to give a percentage of classes attended.

Similarly to the attendance view, this view selects all coursework grades of a particular student for a particular class and uses the weight defined in the coursework table to calculate a weighted average for each student for each class.

Appendix C.2.1 Low Attendance and Low Coursework Grades Views

Two views were created which are updated when a students attendance or weighted grade drops below a satisfactory level. These will be used to notify students and staff when this occurs. A further view was added that joins these two tables and counts whether a student appears more than once. This indicates that a

students attendance and/or grades are below a satisfactory level for more than one class, and will be used to alert head departmental staff and a students respective advisor.

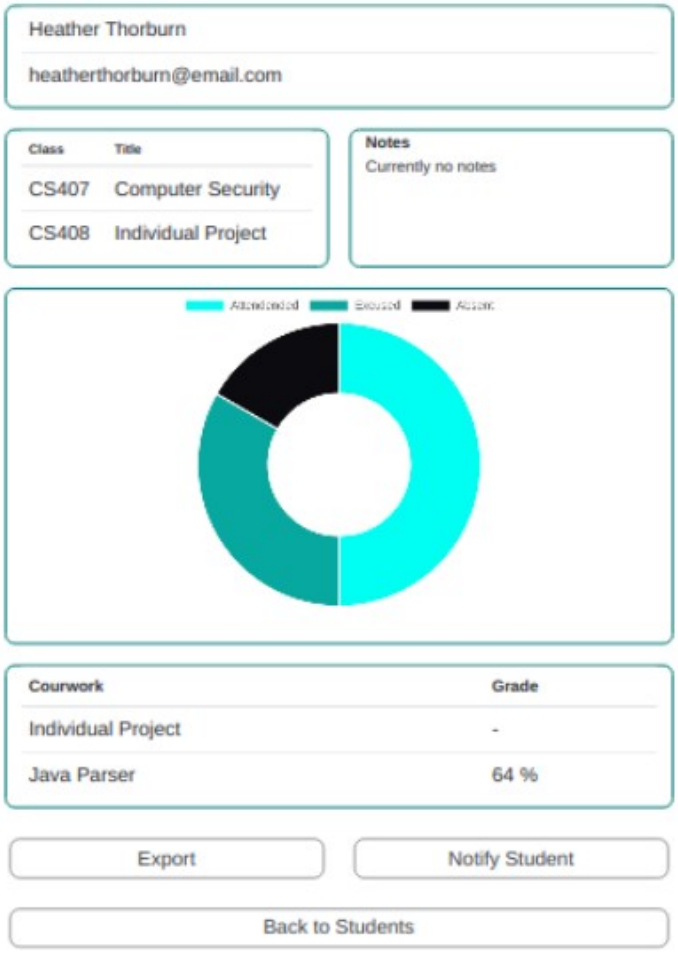
For the purposes of this application, low attendance is defined at under 20% attendance. As from the background research there are varying results on whether attendance is actually an indicator of academic success, this has been set low to accommodate this. Low grades are defined as 40% as this is generally the grade in the UK University Grading System needed to pass a module.

Appendix D - Early Prototype of Student Profile

This is an early prototype of the student profile as a singular page. The amount of space on even on desktop was very limited, and this was before custom messages regarding a students performance was added.

The graph was taking up a lot of space, and many libraries were tested to find a responsive graph. All tested did not respond well when the page was resized. On reflection, the graph could have been included and hidden with a media query in CSS, however even when resizing the page marginally the graphs would cause overflow issues.

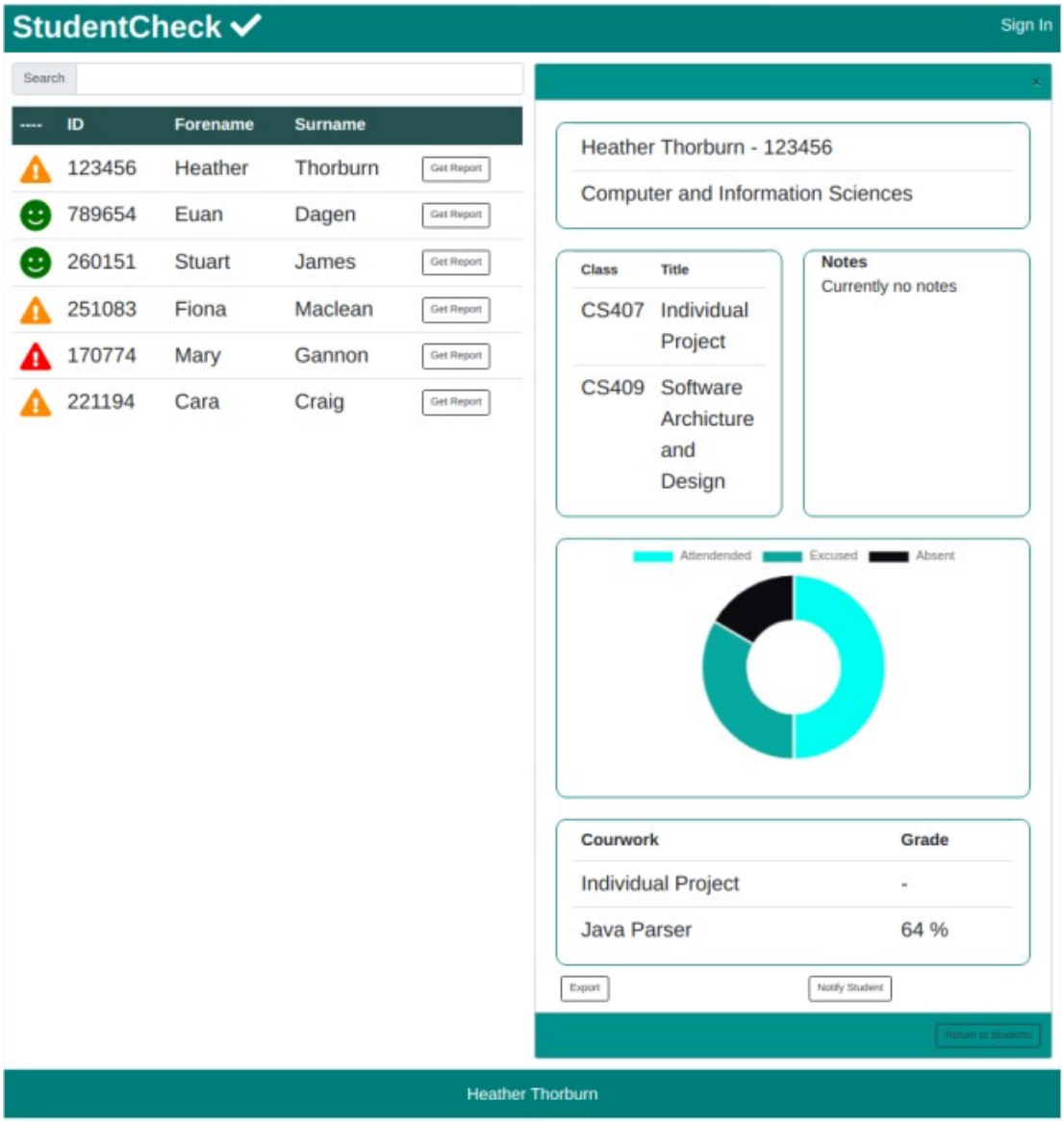
To maximise the amount of information that could be displayed in the profile, it was decided to separate the static information such as name and email address and add them to different components.



Appendix D - Early Prototype of Profile

Appendix E - Early Design of Displaying Students

This early design was created for displaying students. The table of students and the selected profile would be displayed side by side. The reasoning for this was to prevent any unnecessary navigating between displaying a student table and viewing the profile, so that the user could easily and quickly compare student statistics.



Appendix E - Early Design of Student Table

The issue with this prototype was that it did not respond well across different screen sizes, and the amount of information that could be displayed, even on desktop screen sizes was quite limited. It was therefore decided to keep the student table and profiles as separate pages so that the personalised messages, details of grades and coursework could be added without compromising the readability of the profile or having to include multiple tabs.

Appendix F - Extra Technologies Used

This section will detail libraries and components that were used that did not add as much functionality to the program as the ones mentioned in the Detailed Implementation.

Appendix F.1 React Icons

This is a library that combines many icon libraries such as Font Awesome and Bootstrap Icons. Font Awesome was originally used however the selection of available icons was quite limited. React Icons was added so that more suitable icons could be used throughout the application.

Retrieved from : https://react-icons.github.io/react-icons/

Appendix F.2 React Transition Group

This is a library that allows you to wrap components in an extra component. This new component automatically applies class names which can be used in CSS to change the way the inner component appears in the browser when it is rendered.

As the application is a SPA, there are no refreshes throughout the system. When a new component is rendered, it is done so very quickly, and can appear quite jarring. React Transition Group was used so that the components could appear with a very slight fade-in effect using the CSS operations *opacity* and *transition*.

Retrieved from : https://www.npmjs.com/package/react-transition-group

Appendix F.3 Moment.js

This is a library for handling how times and dates are displayed. As times and dates are stored in the database as Timestamps, the formatting is not very visually appealing for the user. Moment.js allows for these to be converted to a more

readable format.

Retrieved from : https://momentjs.com/

Appendix F.4 Express

Express is a framework for Node.js for handling different HTTP requests at different routes. It was used to separate the routes for dispatching actions, and also to allow the responses to be returned in JSON.

It was also neccessary to join the static files to the server-side for deployment. This code was found in the Express documents.

Retrieved from : https://www.npmjs.com/package/express

Appendix F.5 Express Async Handler

This is a middleware that allows you do define intermediate routes before proceeding to final route. This was used to define functions to check the authenticity of JSON Web Tokens before retrieving data.

Retrieved from : https://www.npmjs.com/package/express-async-handler

Appendix F.6 Bycrypt

Bcrypt is a library that allows you to hash and salt passwords, and also decrypt passwords. This was used for the Registration functionality

Retrieved from : https://www.npmjs.com/package/bcrypt

Appendix F.7 Axios

Axios was used to send the HTTP requests from the client-side to the server-side.

Retrieved from : https://github.com/axios/axios

Appendix G - Black Box Testing

These tests were conducted to ensure that the application returns the expected output given user interaction. This was the main method of testing the application, and because the application has a lot of opportunities for user input and error, this stage ended up being very extensive.

Appendix G.1 Staff Login

Staff Login Tests			
Activity Tested	***Expected Output***	***Description***	***Result***
Attempt to login at staff login tab with empty staff ID and empty password fields.	Error message is displayed asking for the user to input a valid staff ID and password.	To determine that if the user leaves both the staff ID and password field empty that the application will not attempt to log them in.	Pass
Attempt to login at staff login tab with empty password field.	Error message is displayed asking for the user to input a valid password.	To determine that the system recognises that a staff ID has been entered but that the password field is empty.	Pass
Attempt to login at staff login tab with empty Staff ID field	Error message is displayed asking for the user to input a valid staff ID	To determine that the system recognises that a password has been entered but the staff ID field is empty	Pass
Attempt to login at the staff login tab with credentials that are not present in the database.	An error message should be displayed stating that the email or password is not recognised.	To determine that the staff user cannot login to the system with invalid credentials	Pass
Attempt to login at the staff login tab with credentials that match a record in the database.	Users should be redirected to the staff home page.	To determine that users can successfully login to the system with valid credentials.	Pass

Appendix G.2 Student Login

Student Login Tests			
Activity Tested	***Expected Output***	***Description***	***Result***
Attempt to login at student login tab with empty student ID and empty password fields.	Error message is displayed asking for the user to input a valid staff ID and password.	To determine that if the user leaves both the student ID and password field empty that the application will not attempt to log them in.	Pass
Attempt to login at student login tab with empty password field.	Error message is displayed asking for the user to input a valid password.	To determine that the system recognises that a student ID has been entered but that the password field is empty.	Pass
Attempt to login at student login tab with empty Staff ID field	Error message is displayed asking for the user to input a valid staff ID	To determine that the system recognises that a password has been entered but the student ID field is empty	Pass
Attempt to login at the student login tab with credentials that are not present in the database.	An error message should be displayed stating that the email or password is not recognised.	To determine that the student user cannot login to the system with invalid credentials	Pass
Attempt to login at the student login tab with credentials that match a record in the database.	Users should be redirected to the staff home page.	To determine that users can successfully login to the system with valid credentials.	Pass

Appendix G.3 Displaying Students (Classes)

Display Students Tests - Classes			
Activity Tested	***Expected Output***	***Description***	***Result***
Clicking on ‘Students’, link for a given class where students are enrolled.	Main component is rerendered to display a table of students enrolled in the given class.	Verifies that clicking on the ‘Students’ link dispatches an event and receives a list of students enrolled in a class.	Pass
Clicking on, ‘Students’ link for a given class where no students are enrolled.	Main component should be rerendered to display a message stating that there are no students enrolled in the class.	Verifies that the table is not displayed if there is no data to be displayed.	Pass
Clicking on the Matriculation header in the table.	Table should be sorted into ascending order by Matriculation number.	Verifies that the table can be sorted by Matriculation number.	Pass
Clicking on the Forename header in the table.	Table should be sorted into ascending order by Forename.	Verifies that the table can be sorted by Forename.	Pass
Clicking on the Surname header in the table.	Table should be sorted into ascending order by Surname.	Verifies that the table can be sorted by Surname.	Pass
Typing “Claudia,” into the search bar in a class where a “Claudia” exists.	Table should return entries containing Claudia in the Forename and Surname	Verifies that the searching algorithm works correctly.	Pass

Typing “Claudia” into the search bar where “Claudia” does not exist.	Table should return a message saying that no entries have been found.	Verifies that the table recognises no entries have been found.	Pass

Appendix G.4 Displaying Students (Advisees and Department)

Display Students Tests - Super Table (Department and Advisees)			
Activity Tested	***Expected Output***	***Description***	***Result***
Clicking on ‘Departments’, link for a given department where students are enrolled.	Main component is rerendered to display a table of students enrolled in the given department.	Verifies that clicking on the ‘Departments’ link dispatches an event and receives a list of students enrolled in a department.	Pass
Clicking on ‘Advisees link’, as a logged in staff member.	Main component is rerendered to display a table of students the staff member is an advisor of.	Verifies that clicking on the ‘Advisees’ link dispatches an event and receives a list of students the staff member is an advisor of.	Pass
Clicking on, ‘Students’ link for a given class where no students are enrolled.	Main component should be rerendered to display a message stating that there are no students enrolled in the class.	Verifies that the table is not displayed if there is no data to be displayed.	Pass

Clicking on the Matriculation header in the table.	Table should be sorted into ascending order by Matriculation number.	Verifies that the table can be sorted by Matriculation number.	Pass
Clicking on the Forename header in the table.	Table should be sorted into ascending order by Forename.	Verifies that the table can be sorted by Forename.	Pass
Clicking on the Surname header in the table.	Table should be sorted into ascending order by Surname.	Verifies that the table can be sorted by Surname.	Pass
Clicking on the Coursework header in the table.	Table should be sorted into ascending order by Coursework.	Verifies that the table can be sorted by Coursework.	Pass
Typing “Claudia,” into the first search bar in a class where a “Claudia” exists.	Table should return entries containing Claudia in the Forename and Surname	Verifies that the searching algorithm works correctly by filtering by a specific name.	Pass
Typing “Claudia” into the first search bar where “Claudia” does not exist.	Table should return a message saying that no entries have been found.	Verifies that the table recognises no entries have been found.	Pass
Typing “Computer Science” into the second search bar where “Computer Science” does exist.	Table should return a all students enrolled in the course, “Computer Science”	Verifies that the search algorithm for courses successfully filters.	Pass
Typing “Computer Science” into the first search bar	Table should return a message saying that no	Verifies that the table recognises no entries have	Pass

where “Computer Science” does not exist.	entries have been found.	been found with the search by course filter.	
Typing “Claudia” into the first search bar where “Claudia” does not exist.	Table should return a message saying that no entries have been found.	Verifies that the table recognises no entries have been found.	Pass
Pressing “1” in the year filter button group.	Table should return all students who are in their first year of study.	Verifies that the filter by year functionality works.	Pass
Pressing the “Clear Filters” button	All filters should be cleared. The text inputs should be returned to their default state and the year button group should be cleared of a selection.	Verify that the button successfully triggers a function that clears all filter states.	Pass

Appendix G.5 Class Profiles

Coursework Tests			
Activity Tested	***Expected Output***	***Description***	***Result***
Click on ‘Academic Info’ Tab	User is displayed with a screen showing a table of statistics with personalised messages.	Checks that a personalised report is returned to the user	Pass
Click on ‘Export Report to PDF’	A PDF file with the same information from the profile is downloaded and opened, with today’s date.	Checks that the PDF functionality works and uses the correct information from the profile.	Pass

Appendix G.6 Coursework

Coursework Tests			
Activity Tested	***Expected Output***	***Description***	***Result***
Clicking on ‘Manage Coursework’ for a given class where coursework exists.	User is displayed with a table of courseworks for the class.	Verifies that an action is dispatched and a list of coursework for the class is returned	Pass
Clicking on ‘Manage Coursework’ for a given class where no coursework exists	User is displayed with a message stating that no coursework exists.	Verifies that the application will not return an empty table.	Pass
In ‘Add Coursework’ for a given class, try and submit a coursework with empty field(s)	Empty field(s) should be highlighted in red with an appropriate	Verifies that the application recognises empty fields and the coursework is not	Pass

	message.	added with empty fields.	
In ‘Add Coursework’, input a non-numeric value in the percentage field.	Percentage fields will be highlighted in red with an appropriate message.	Verifies that the application recognises that a non-numeric value has been entered.	Pass
In ‘Add Coursework’ input a value over 100.	Percentage field is highlighted in red with an appropriate message.	Verifies the application recognises that an invalid percentage has been entered.	Pass
In ‘Add Coursework,’ submit a with valid inputs.	A message is displayed stating the coursework has been added with an ID number.	Verifies the application has dispatched an event and the coursework has been saved.	Pass
Add two courseworks with weight 51%	A message should be displayed to the user during the second addition stating that the weight is too high.	Checks if the user can add courseworks so that the total weight can be over 100%	Pass
In ‘Edit Coursework,” press delete.	A message should be displayed asking the user to confirm their selection	Verifies the two-step system for deleting coursework	Pass
Click “No” in the two-step deletion process	The ‘Edit Coursework’ page should revert back to its state before pressing ‘Delete Coursework’	Checks that the deletion process is successfully cancelled.	Pass
Click “Yes” in the two-step deletion	A message is displayed stating	Checks that the deletion process is	Pass

process	the coursework has been successfully deleted.	confirmed after the two-deletion process.	
In ‘Edit Grade’ for a given student, try and submit a grade with empty field(s)	Empty field(s) should be highlighted in red with an appropriate message.	Verifies that the application recognises empty fields and the grade is not added with empty fields.	Pass
In ‘Edit Grade’, input a non-numeric value in the percentage field.	Percentage fields will be highlighted in red with an appropriate message.	Verifies that the application recognises that a non-numeric value has been entered.	Pass
In ‘Edit Grade’ input a value over 100.	Percentage field is highlighted in red with an appropriate message.	Verifies the application recognises that an invalid percentage has been entered.	Pass
Enter valid input and submit in ‘Edit Grade’ for a given student.	Returning to the coursework page, this grade should be displayed in the table.	Verifies that the application has successfully saved the student’s grade.	Pass
In ‘Upload CSV”, upload a valid CSV file.	The user should displayed a success message and a list of the attendances	Verifies that the application has successfully parsed the CSV file	Pass
In ‘Upload CSV”, upload a non CSV file.	The user should be displayed an error message stating the file must be a .csv file	Checks if an invalid file type can be uploaded and used to parse attendance	Pass

Appendix G.7 Attendance

Attendance Tests			
Activity Tested	***Expected Output***	***Description***	***Result***
In ‘Attendance’ for a certain class, submit a class with empty field(s)	Appropriate error message should be displayed to enter a time/date	To verify user input before taking a register.	Pass
In ‘Attendance’ enter an invalid date/time.	When clicked off, the input should be removed from the input field.	To verify that it is not possible to enter an invalid date.	Pass Note: If the user has inputted a valid date, then replaced with an invalid date it will revert back to the previous date. Otherwise, the input field will return to blank.
In ‘Attendance’, enter and submit a valid time and date.	The user should be displayed with a success message, and a list of students in the class with the attendance taking options.	To verify that the application recognises valid date and time.	Pass
Submit attendance via the checklist with a mixture of boxes checked and unchecked.	The user should have a success message and displayed a list of the attendance taken.	Verifies the attendance has been recorded successfully and accurately.	Pass
In ‘Generate QR Code’, scan the QR code.	The user should be directed to a form to enter and submit their user	Verifies the integrity of the QR code.	Pass

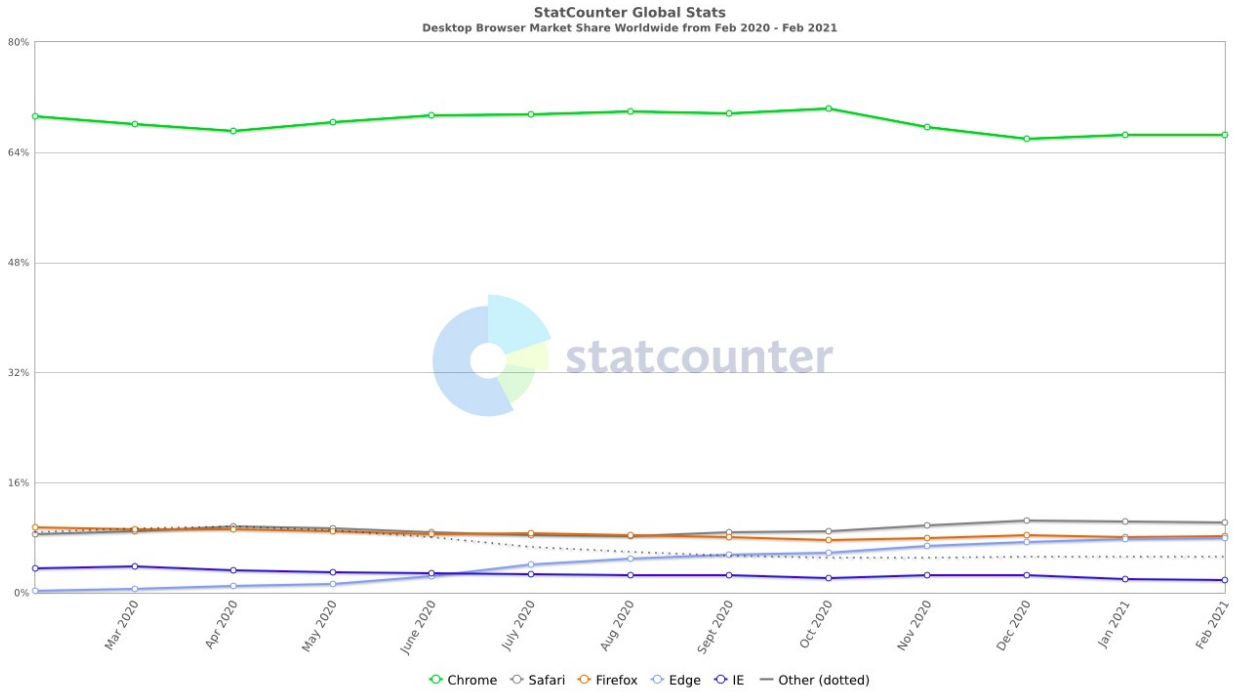
	ID.		
In ‘Upload CSV”, upload a valid CSV file.	The user should displayed a success message and a list of the attendances	Verifies that the application has successfully parsed the CSV file	Pass
In ‘Upload CSV”, upload a non CSV file.	The user should be displayed an error message stating the file must be a .csv file	Checks if an invalid file type can be uploaded and used to parse attendance	Pass
In ‘Upload CSV’ upload a CSV file with invalid headers.	The user should be displayed an error message stating their file must contain appropriate headers.	Checks if a file with invalid headers can be uploaded and used to parse attendance	Pass
In ‘Upload CSV’ upload a valid CSV file with appropriate headers	The user should be displayed a table showing the names in the file as present and names not present but enrolled in the class as not present	Checks if the CSV parser works correctly and not present students are accounted for	Pass

Appendix G.8 Extenuating Circumstances

Circumstances Tests			
Activity Tested	***Expected Output***	***Description***	***Result***
In ‘Extenuating Circumstances’ form, navigate forwards with empty fields.	Appropriate error message should be displayed under empty fields	To verify user input.	Pass
In ‘Extenuating Circumstances’, input a non-date into the date field	Input should automatically change to a blank input field.	To verify that the user cannot put a non-date into the field and consequently the database.	Pass Note: If the user has inputted a valid date, then replaced with an invalid date it will revert back to the previous date. Otherwise, the input field will return to blank.
In ‘Extenuating Circumstances’ input a date in the future	Input field should automatically revert to today.	To verify the user cannot attempt to log circumstances in the future	Pass
Select ‘Physical Health’ and navigate forwards.	User should be recommended services assosicated with Physical Heath	To verify that appropriate recommendations are outputted	Pass
Input valid information and navigate to the ‘Review’ page	All information entered should be present in the review table.	To verify that the information is stored correctly before being sent for submission	Pass

Appendix H – Browser Compatibility

Below is a graph that plots browsers by market share. From this graph, it was decided that the application would be tested in Safari, Firefox, Edge and IE.



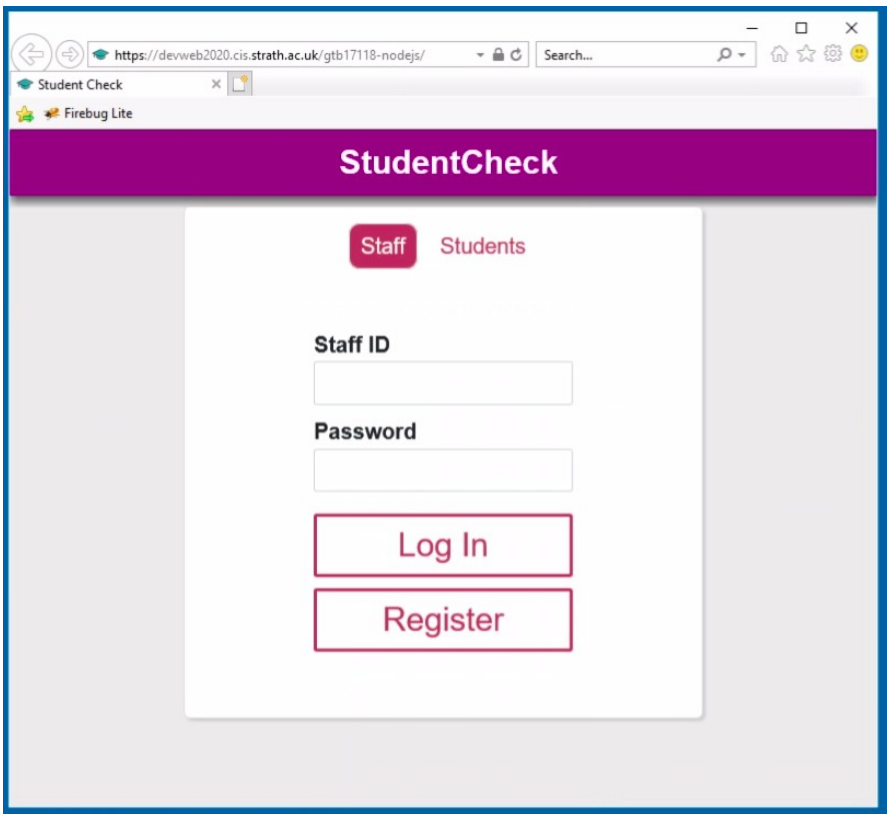
Appendix H : Browsers by Market Share Source : StatCounter

The actions carried out in the black box testing were applied in all four of these browsers and compared to the browser that was used for development, Chrome. Older versions of Chrome were also tested. The results are as follows:

Browser	Version	Result
Firefox	81 (Latest)	No issues
Firefox	80	No issues
Safari	12 (Latest)	No issues

Microsoft Edge	89 (Latest)	No issues
Microsoft Edge	88	No issues
Chrome	88	No issues
Chrome	87	No issues
IE	11 (Latest)	Major issues

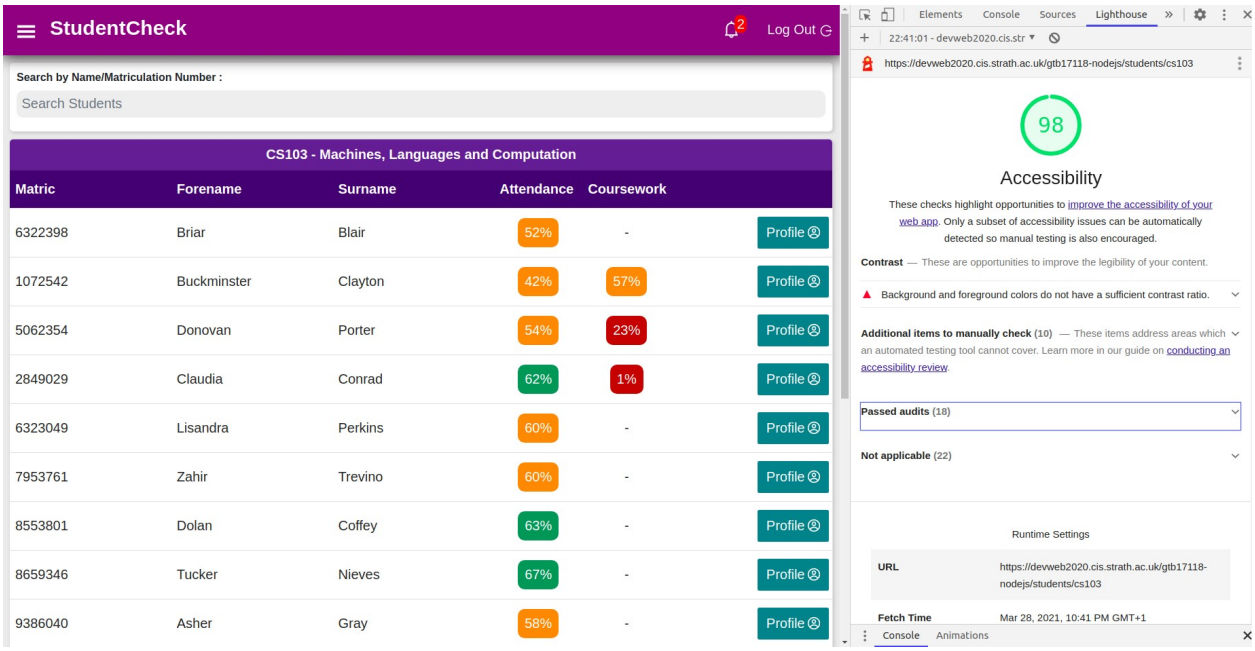
After using polyfills, the application was able to be run on IE.



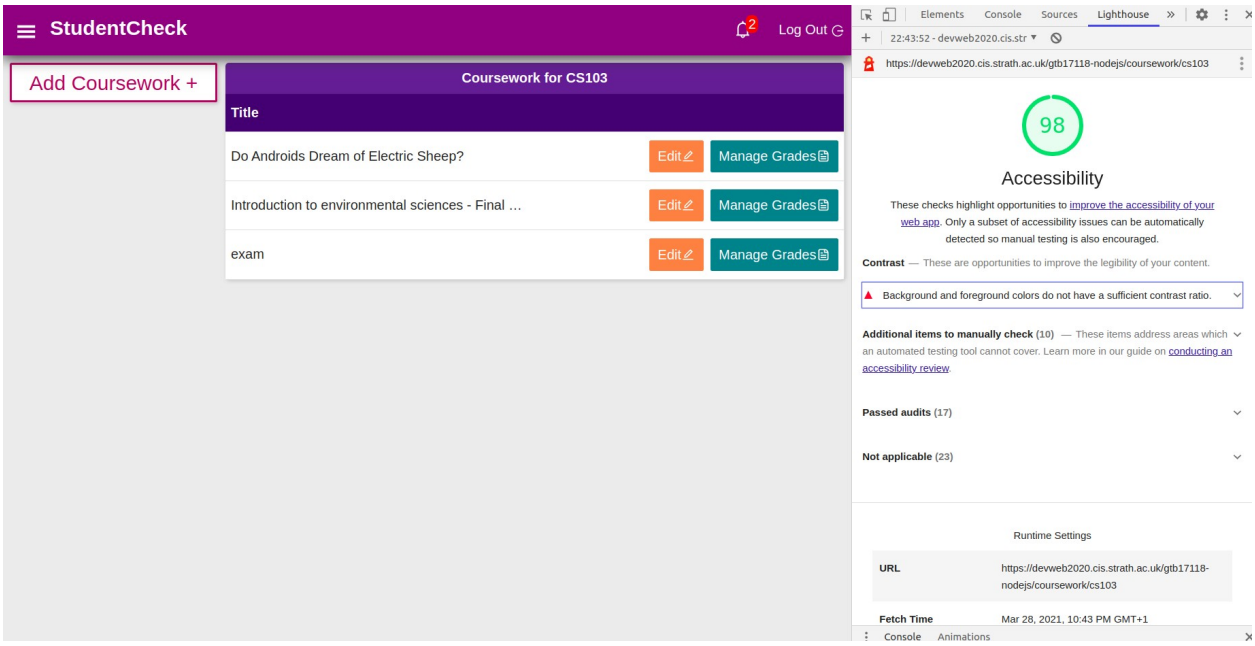
Appendix H: Application running on IE 11

Appendix I - Accessibility Testing

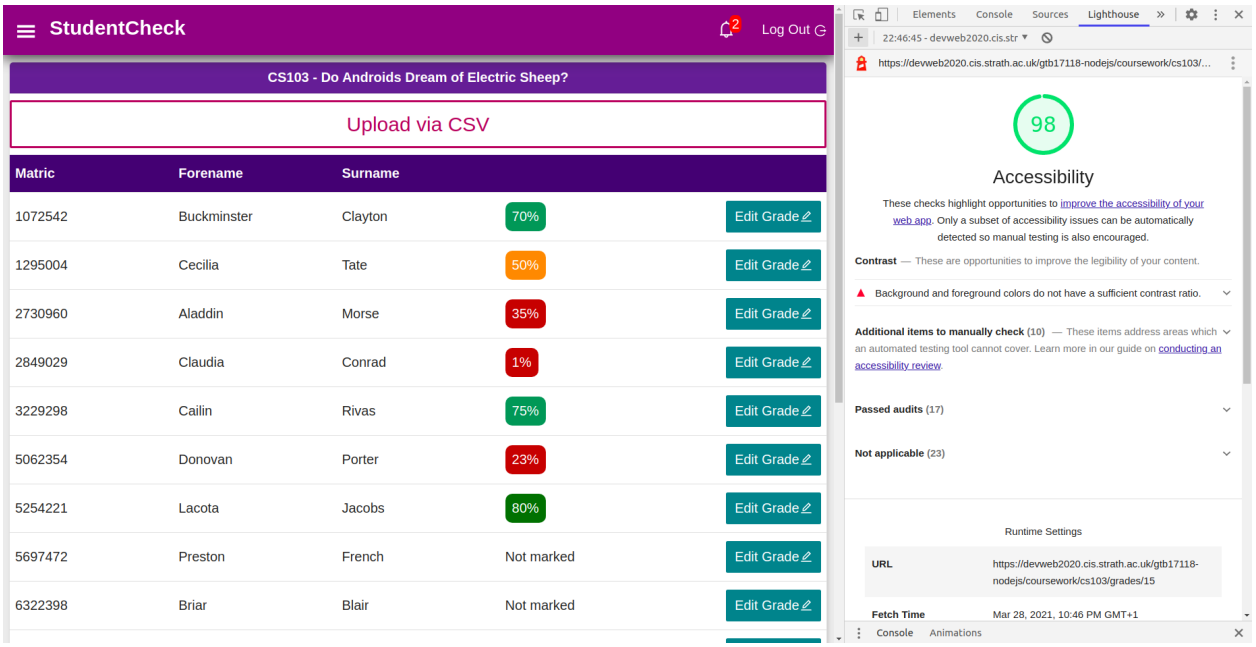
Below are some screenshots of the LightHouse Tests to evaluate accessability.



Appendix I.1 - LightHouse Test on Student Table

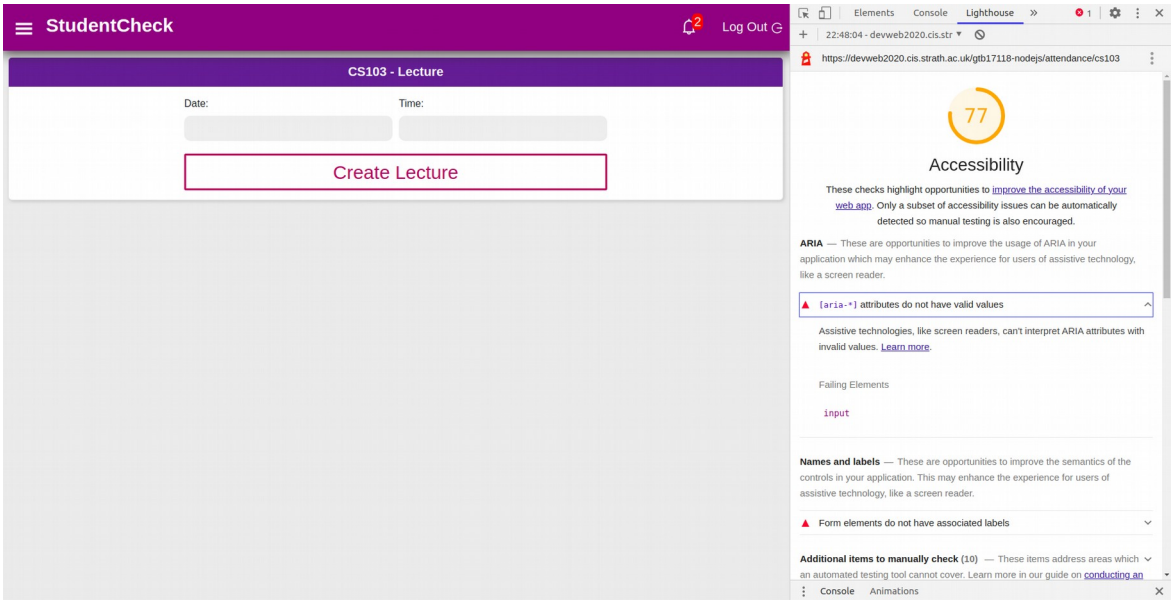


Appendix I.2 - LightHouse Test on Coursework Page



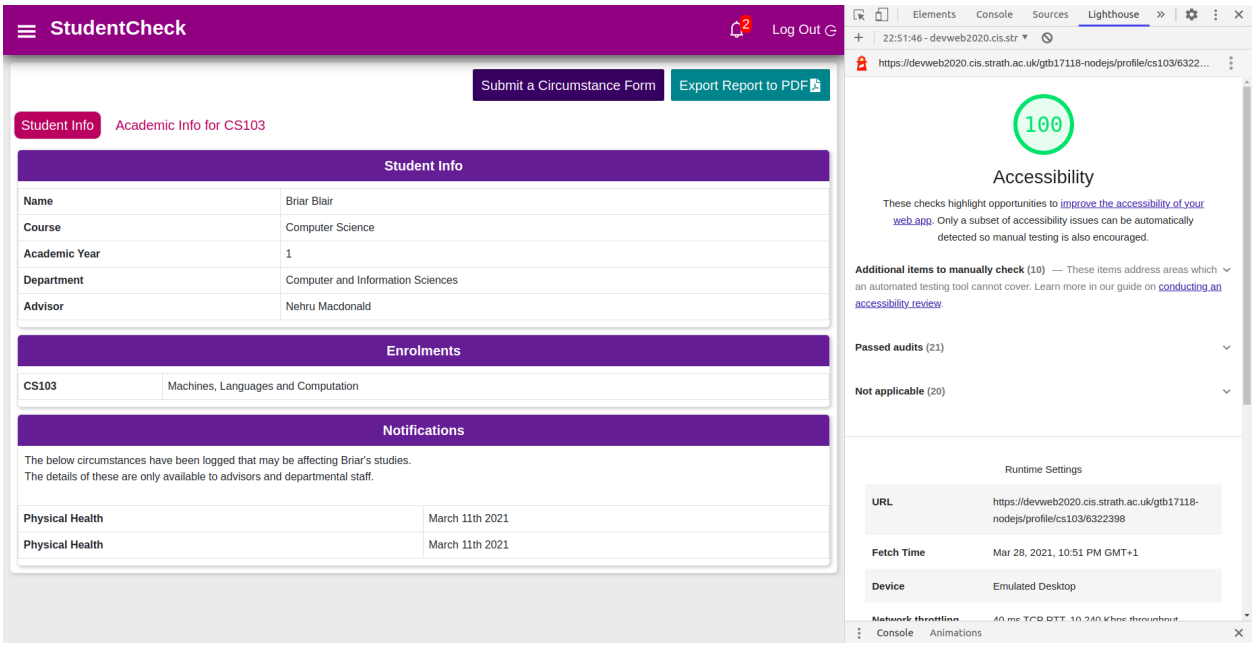
Appendix I.3 - LightHouse Test on Coursework Grades

In the above pages, the issue was the orange colour for the warning and edit buttons, which did not have sufficient contrast as the font-colour was white and the font size being smaller.



Appendix I.4 - LightHouse Tests on Attendance Page

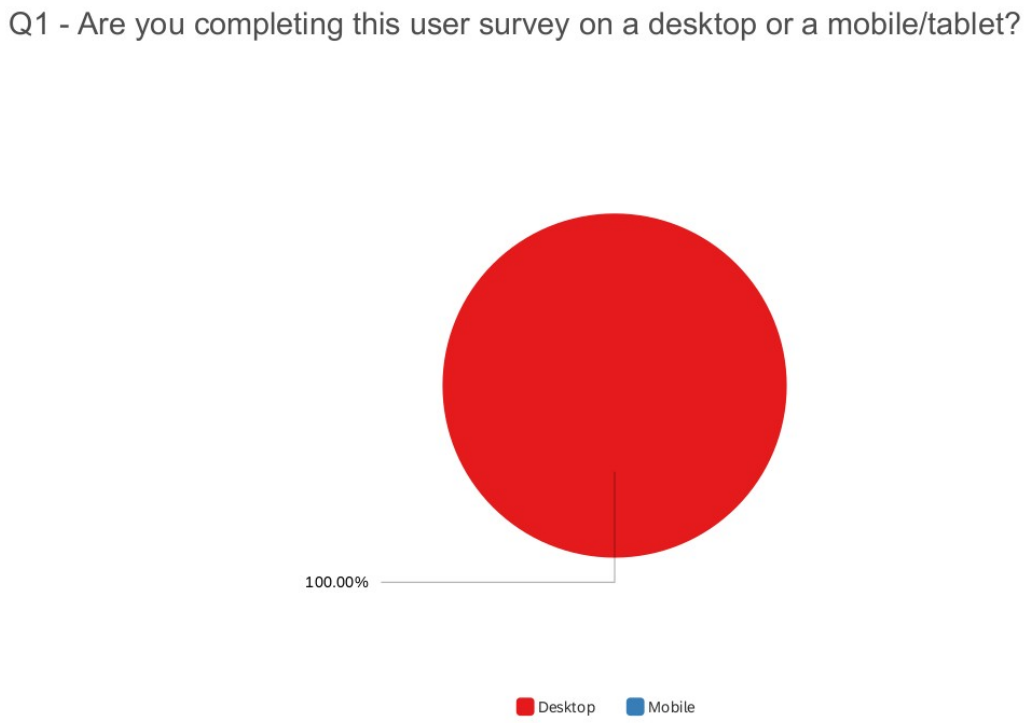
This score was due to the ***React-DatePicker*** component. The component has aria-labels that do not pass the accessibility audit. The documents were consulted to try and change the aria-labels however it would still fail the audit. Therefore, in further developments, another datepicker component should be researched.



Appendix I.5 - LightHouse Tests on Profile Page

Appendix J – User Evaluation Results

For the user evaluation, university staff were contacted to complete the survey as they are the main users of the application. The user was asked to login to the application and carry out a series of tasks. They would then evaluate the process and provide any feedback about the process.

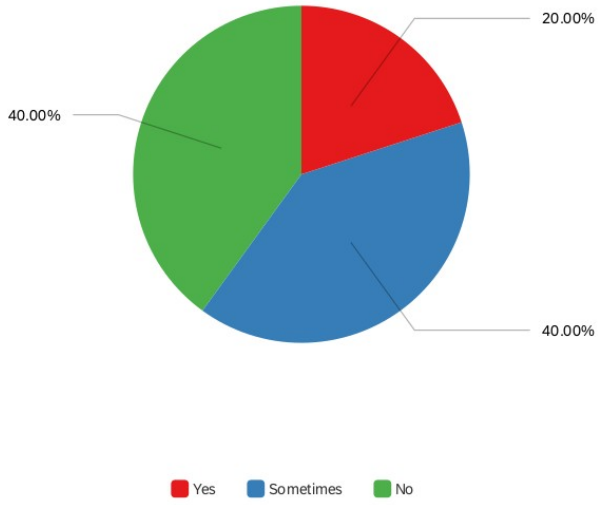


#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Are you completing this user survey on a desktop or a mobile/tablet?	1.00	1.00	1.00	0.00	0.00	5

This was to see if data could be gathered about the usability on a variety of devices. However, all partipants completed the survey on the desktop. This led to responsiveness testing during the testing phase where it was discovered that the application worked well across a variety of devices. The application was also

developed with screen-width flexibility in mind.

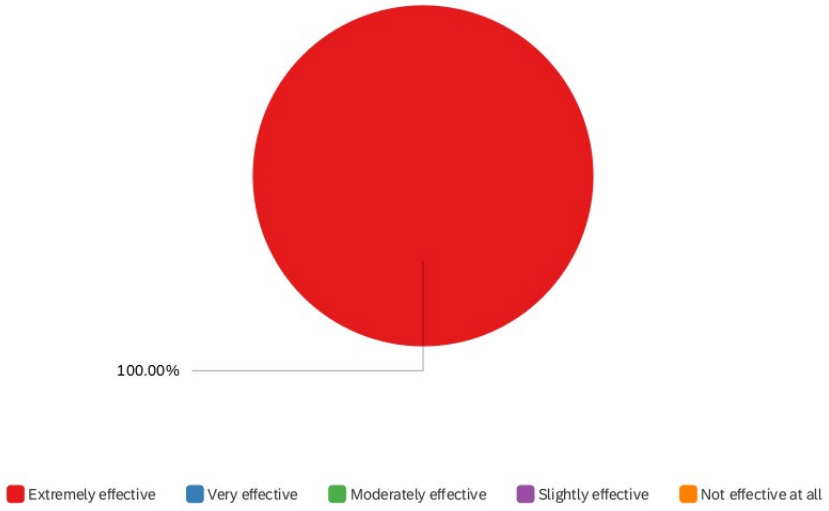
Q2 - Do you currently use any platform to monitor student progress?



#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Do you currently use any platform to monitor student progress?	4.00	6.00	5.20	0.75	0.56	5

This question was asked to determine if after the survey, whether participants who already use applications to monitor student performance would consider using StudentCheck as well, or if participants who did not use any kind of application would consider starting by using Student Check.

Q4 - How would you describe the log in process?



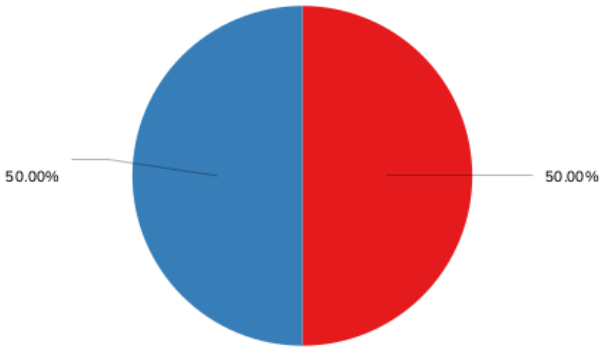
#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	How would you describe the log in process?	1.00	1.00	1.00	0.00	0.00	5

Q5 - Please provide any additional feedback about the login process.

Please provide any additional feedback about the login process.
It was simple and straight forward
Everything I could've asked for in a login screen

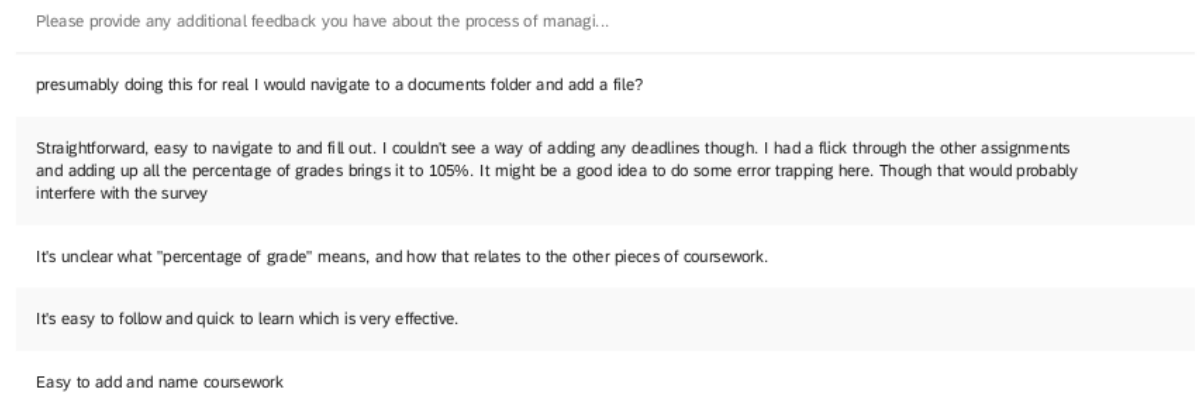
The participants were provided with login credentials. As above, it can be seen that there were no issues highlighted with the login system.

Q6 - You wish to add a new coursework for your taught class CS103. Please try to add this (You can call the coursework anything you like, however this will be visible to other users taking the survey). How would you describe the process of adding a coursework to a particular class?



#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	You wish to add a new coursework for your taught class CS103. Please try to add this (You can call the coursework anything you like, however this will be visible to other users taking the survey). How would you describe the process of adding a coursework to a particular class?	1.00	2.00	1.50	0.50	0.25	4

Q7 - Please provide any additional feedback you have about the process of managing coursework on the website.



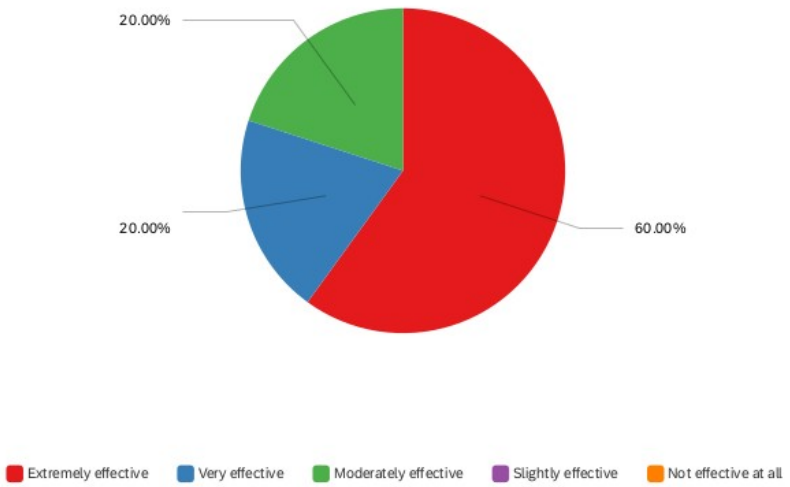
Please provide any additional feedback you have about the process of managi...
presumably doing this for real I would navigate to a documents folder and add a file?
Straightforward, easy to navigate to and fill out. I couldn't see a way of adding any deadlines though. I had a flick through the other assignments and adding up all the percentage of grades brings it to 105%. It might be a good idea to do some error trapping here. Though that would probably interfere with the survey
It's unclear what "percentage of grade" means, and how that relates to the other pieces of coursework.
It's easy to follow and quick to learn which is very effective.
Easy to add and name coursework

From the results, it can be seen that there were some issues with the clarity of the addition of coursework.

The feedback regarding the coursework bringing the total up to 105% was a valid point, as the calculation to prevent this was disabled for the purpose of the survey and it should have been made clearer that this was the case.

Feedback stating that a file should be uploaded, assumably means that the participant thought that the purpose of the application includes coursework distribution to students. While this is not a feature that was considered for this development student, the ability to add the coursework as a file for students to download is a feature that could certainly be considered. Another feature from the feedback that should be added to the application is having catagories for coursework. For example, Labs are worth 50% of the coursework grade, and each lab is work 10% each.

Q8 - For a coursework in CS103, you wish to upload a grade for a student. Please try and complete this (You can choose any coursework and any student you like). How would you describe the the process of adding coursework grades?

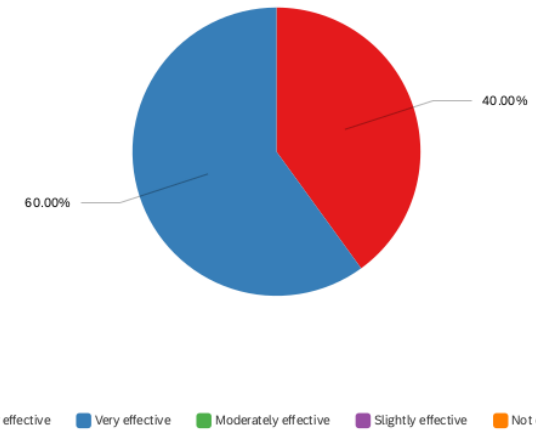


#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	For a coursework in CS103, you wish to upload a grade for a student. Please try and complete this (You can choose any coursework and any student you like). How would you describe the the process of adding coursework grades?	1.00	3.00	1.60	0.80	0.64	5

The feedback was overall positive and highlighted features that could be included in future developments. These features include addition of GPAs and qualitative scores. This was not included in this development cycle as GPAs and qualitative scores usually will have a percentage equivalent, therefore adding by percentage was prioritised.

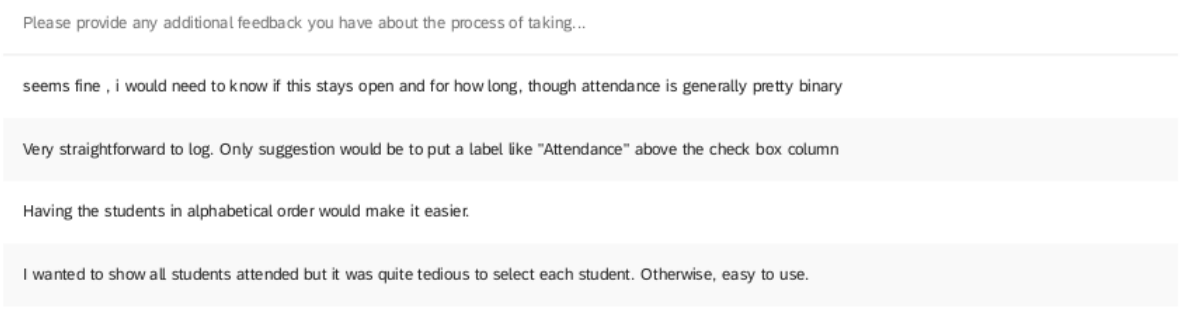
There was feedback to include the ability to upload coursework marks all at once. This feature is included as the user can upload grades through a CSV file in bulk. However, the ability to bulk add grades through a table could be implemented in the case the user does not wish to create a CSV file.

Q10 - You want to take the attendance of a lecture in CS103. Please try and complete this (The QR option requires student input therefore I would suggest the Manual Option) How did you find the process of logging attendance?



#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	You want to take the attendance of a lecture in CS103. Please try and complete this (The QR option requires student input therefore I would suggest the Manual Option) How did you find the process of logging attendance?	1.00	2.00	1.60	0.49	0.24	5

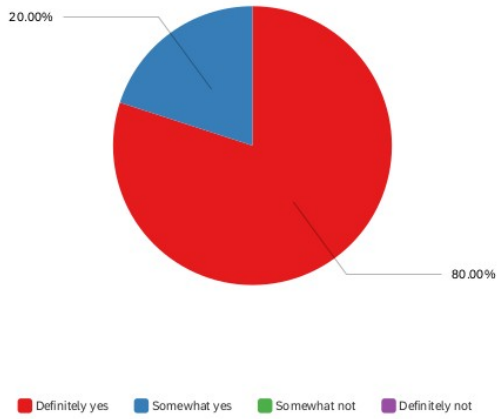
Q11 - Please provide any additional feedback you have about the process of taking attendance on the website.



The feedback for this feature was also positive, however there were also some issues that could be addressed regarding clarity. The attendance label is a very easy feature that could be implemented and would add clarity to the process.

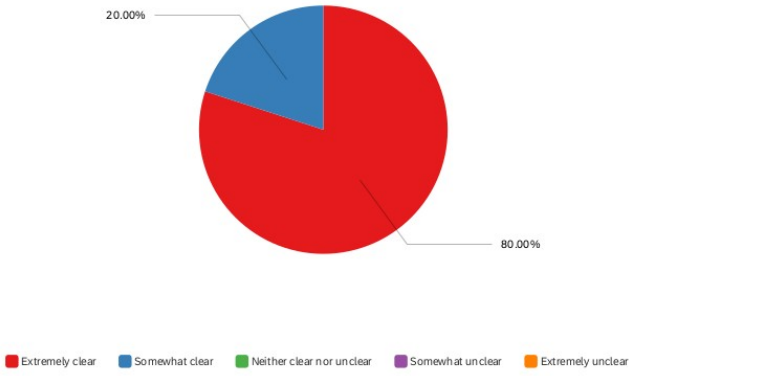
There is functionality with the application to order the student by forename and surname. This can be done by clicking the appropriate headers in the table. All entries can also be selected by clicking the checkbox in the header of the table. However, even though this functionality is included, it was missed by the participants therefore labels should be added to make the process more clear.

Q12 - Please navigate to the students page of CS103. Can you easily identify which students that may have concerning attendance or grades?



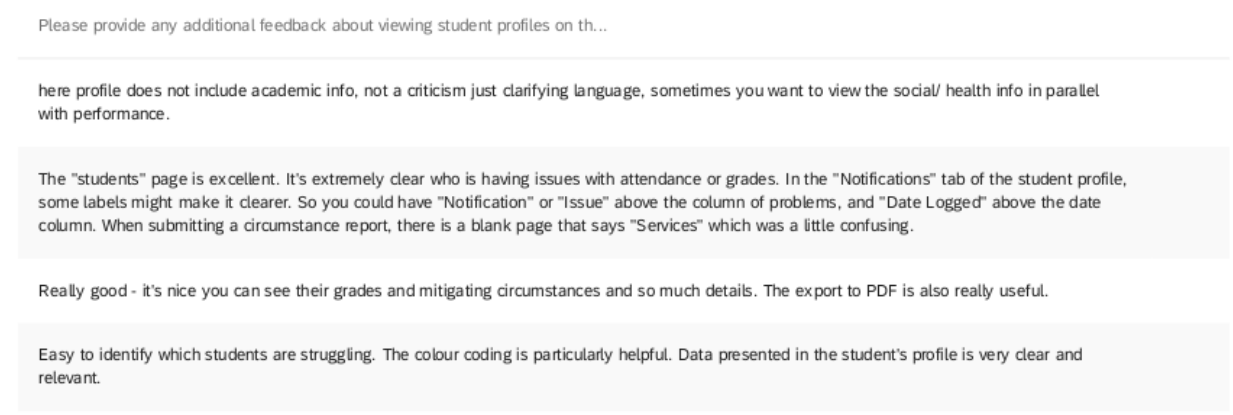
#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Please navigate to the students page of CS103. Can you easily identify which students that may have concerning attendance or grades?	1.00	2.00	1.20	0.40	0.16	5

Q13 - Please navigate to a student's profile. How would would you describe the data presented in the profile?



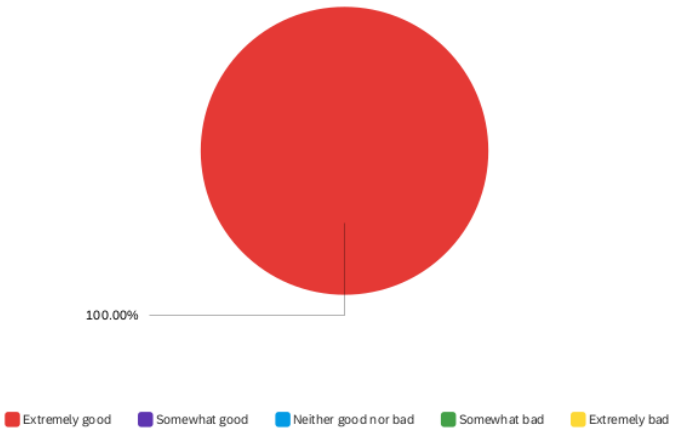
#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Please navigate to a student's profile. How would would you describe the data presented in the profile?	1.00	2.00	1.20	0.40	0.16	5

Q14 - Please provide any additional feedback about viewing student profiles on the website.



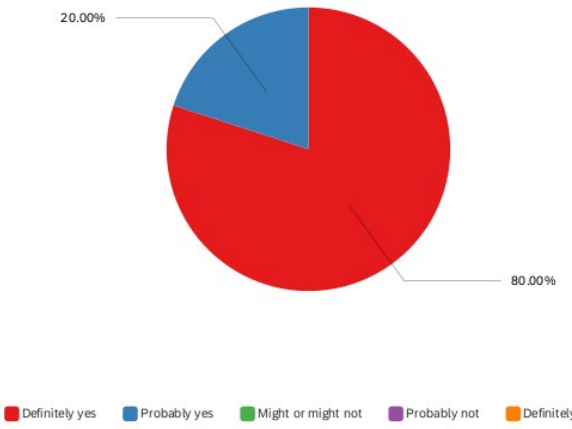
The profiles recieved excellent feedback with issues that can be easily mitigated in the next development cycle. The issue with the second feedback with regards to the empty “Services” tab was an unfortunate oversight when mitigating the table from working locally to deployment. The junction table linking student services and resources had been left empty so this led to an empty page. This however led to the addition of a default message if no resources are available, referring the user advice to contact the student’s advisor.

Q15 - How would you describe the look and feel of the application? Please feel free to navigate around the website and test the functionality.

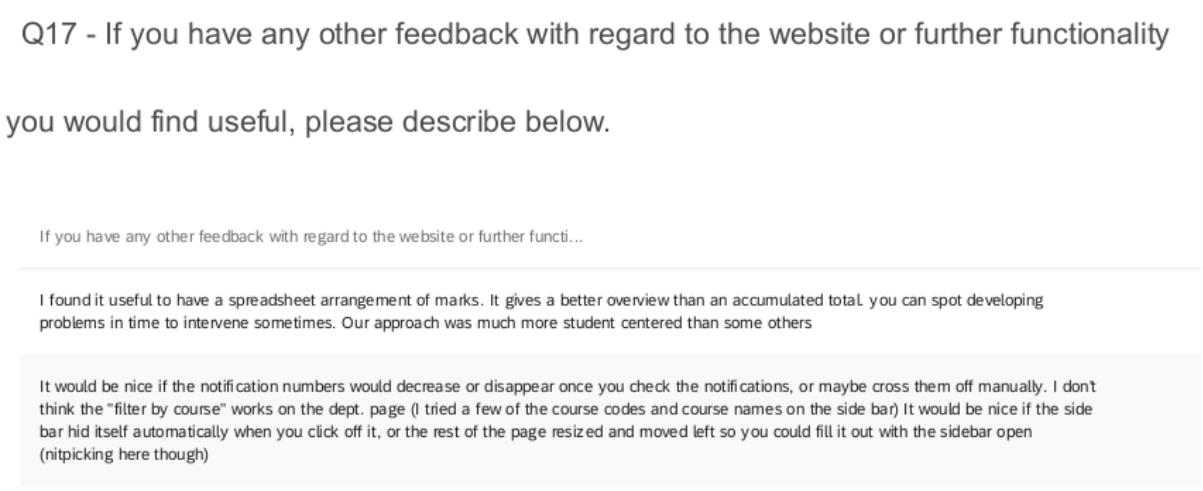


#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	How would you describe the look and feel of the application? Please feel free to navigate around the website and test the functionality.	1.00	1.00	1.00	0.00	0.00	5

Q16 - Is this an application that you would consider useful in your role as university staff?



#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Is this an application that you would consider useful in your role as university staff?	1.00	2.00	1.20	0.40	0.16	5



Overall, the participants state that the look and feel of the website was good, and that it was an application that they would consider using in their role. The feedback was useful as it reinforced some of the advanced functionality that could not be implemented in the time frame would be useful, for example exporting to CSV.

The issue with filtering is also something that can be addressed in the short term as highlighted earlier in the report.

Appendix K - User Manual

Appendix K.1 Introduction

The application is available at https://devweb2020.cis.strath.ac.uk/gtb17118-nodejs/. The application here is configured to run with a database on DEVWEB.

Please note: for the purposes of demonstrating the functionality of the application, accounts that have been set up are:

Staff :

Melyssa Crawford – Department Head, Teacher and Advisor
ID – 9853953
Password – password

Daniel Gross – Teacher
ID – 9350882
Password - password2

Student :

Buckminster Clayton
ID – 8650591
Password – password3

Cailin Rivas
ID – 3229298
Password – password4

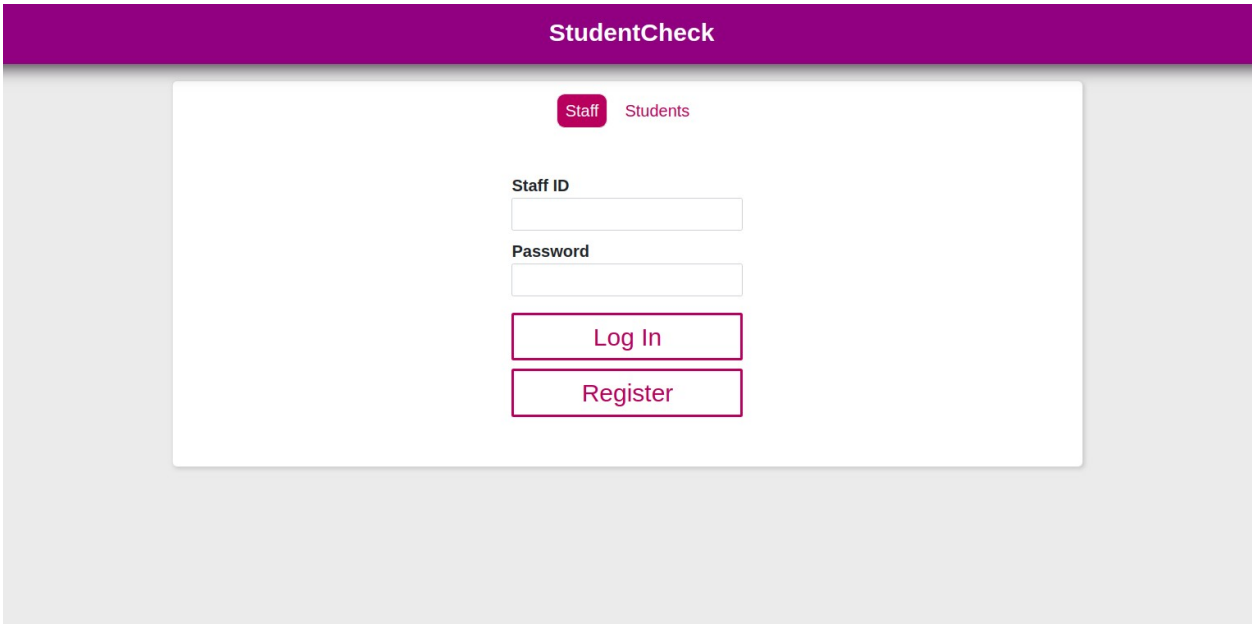
Please consult the .readme file for accounts that have been set up on the system. All information on the database is entirely fictional. There will

also be details on how to set up this application to work with your own database.

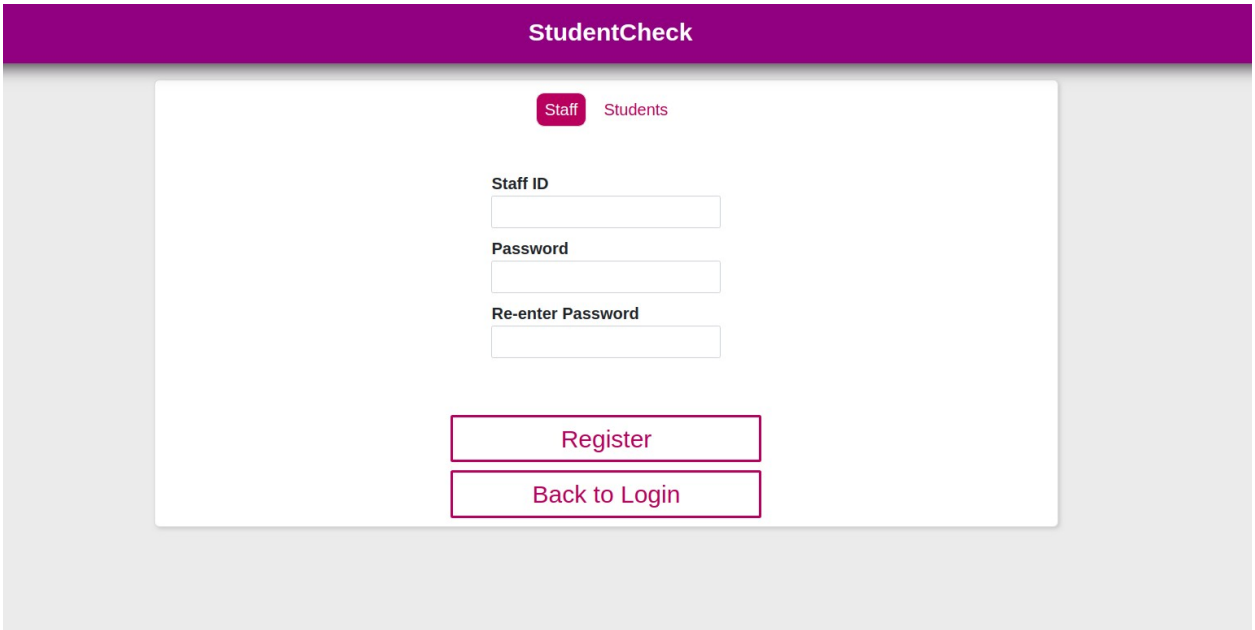
Appendix K.2 Login Page

The login page allows the user to enter their credentials so that they can view the information available for them. There is a tab so that the user can switch between logging in as a student or logging in as staff.

If the user has not registered yet, the user can click on, “Register”, for the respective login page. The user can then enter their ID and desired password. If the ID is present in the database and the password passes the verification checks, then this password is encrypted and saved to the database, and can be used to login.



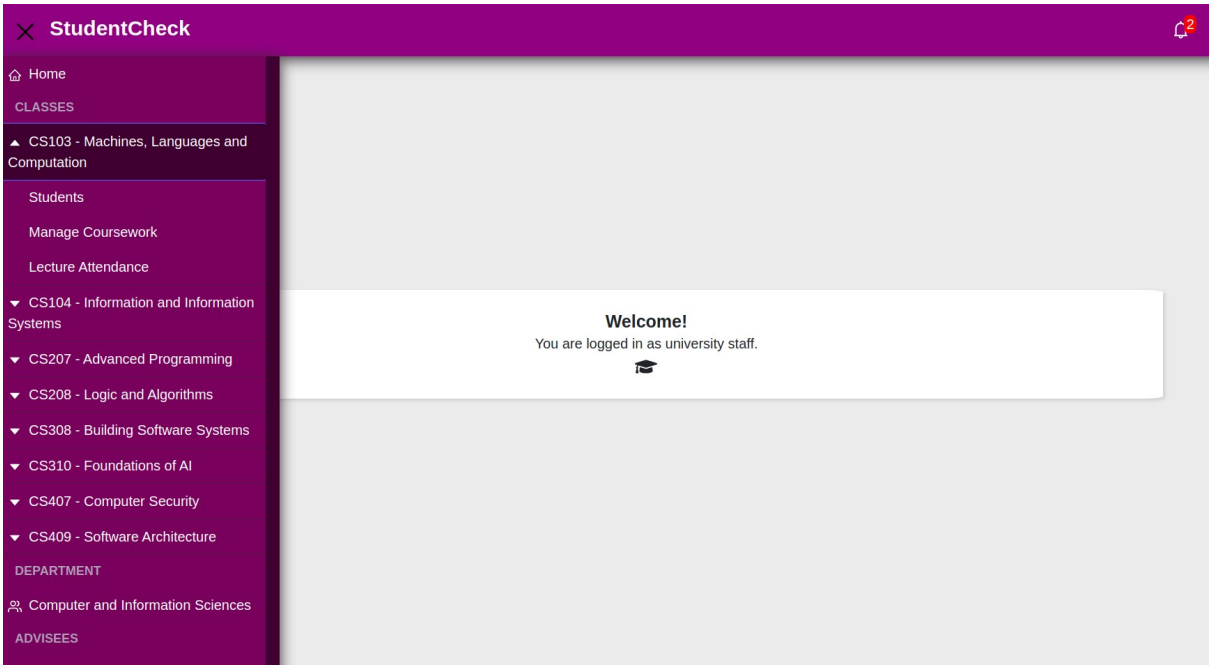
Appendix K.2 - Login Page



Appendix K.2 - Register Page

Appendix K.3 Staff Homepage/Navigation

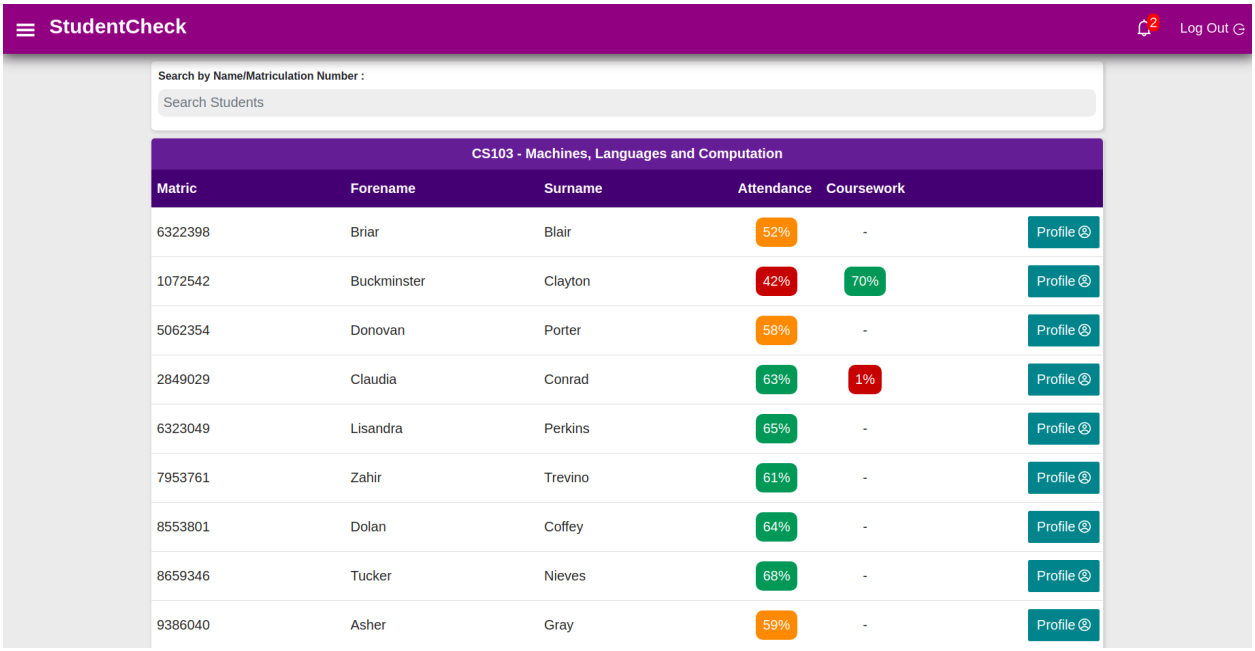
Upon successful registration and/or login, the user will be redirected to a homepage. The header contains a ‘hamburger’ icon in the top left of the page. When clicked, the navigation side-bar will open. Clicking on a class will expand the sub-menu for that class.



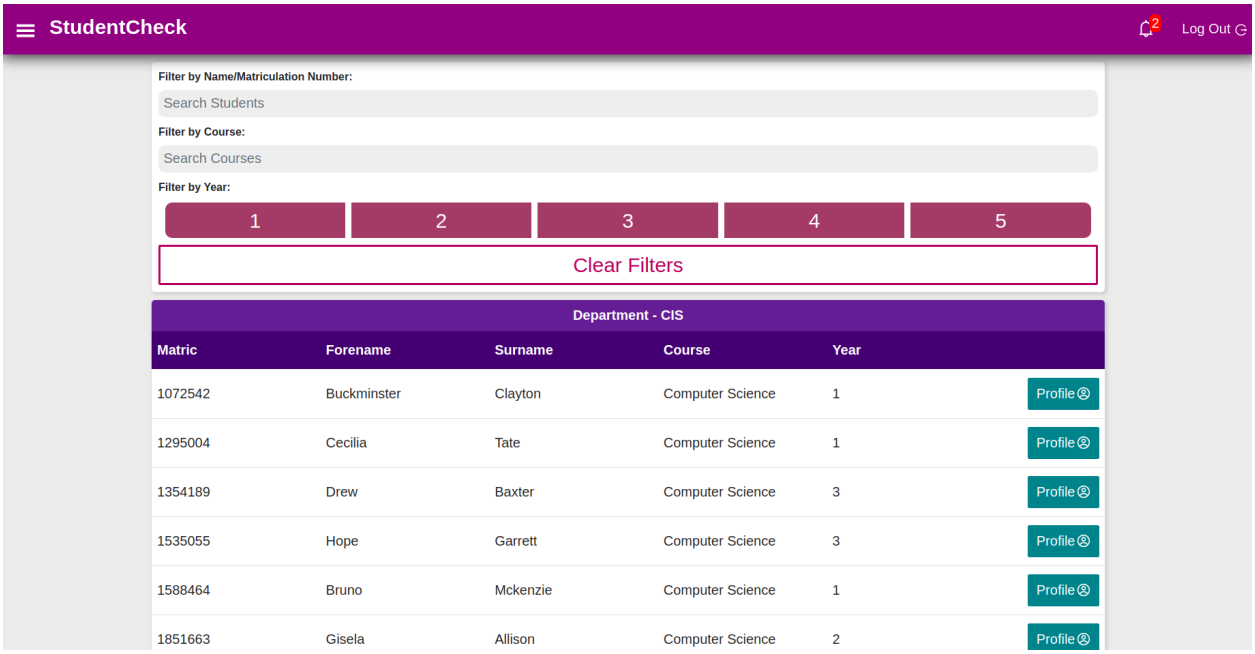
Appendix K.3 - Staff Navigation

Appendix K.4 Viewing Students in Classes/Departments/Advisees.

The user is presented with a table with a list of students in the class. There is a search bar at the top of the page. When a term is entered into this bar, it will filter the table with entries that contain the search term. Clicking on the forename, surname or matriculation number columns will sort the table into ascending/descending order.



Appendix K.4 - Viewing Students in Class



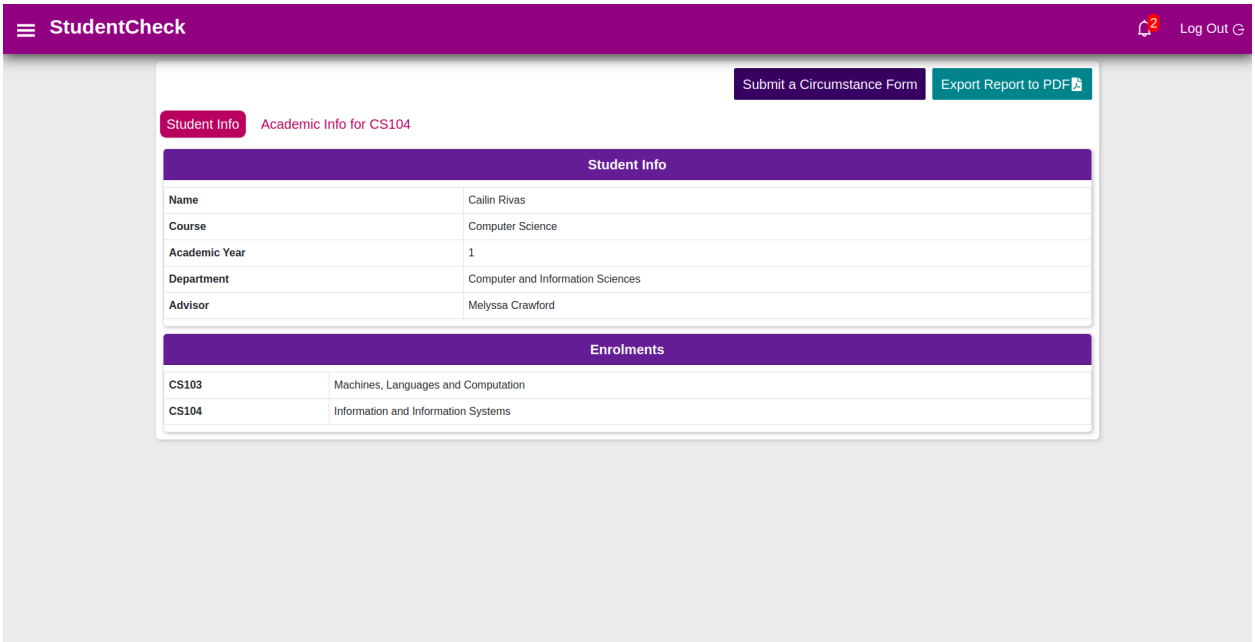
Appendix K.4 - Viewing Students in Department

For Departmental Heads and Advisees, their table will include more filters as the table will contain more information such as year and course.

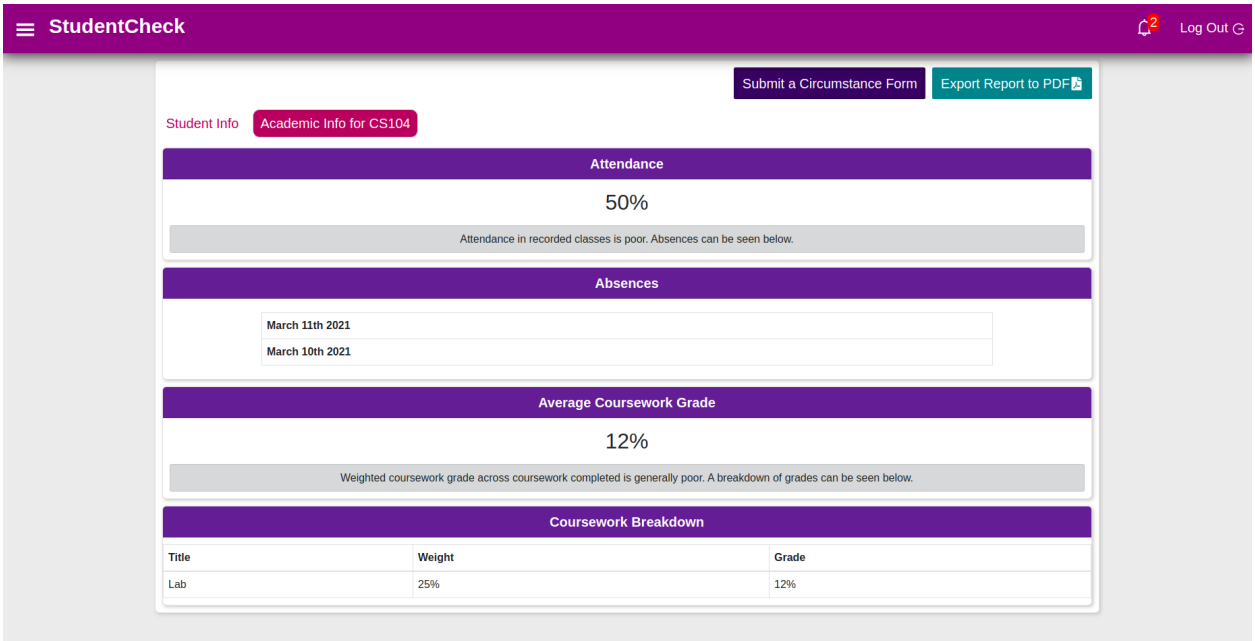
Appendix K.5 Viewing Profiles

Class profiles will show the general information about the student in the Overview page. Clicking on the Academic Information tab will present the user with attendance statistics and coursework statistics with their corresponding messages.

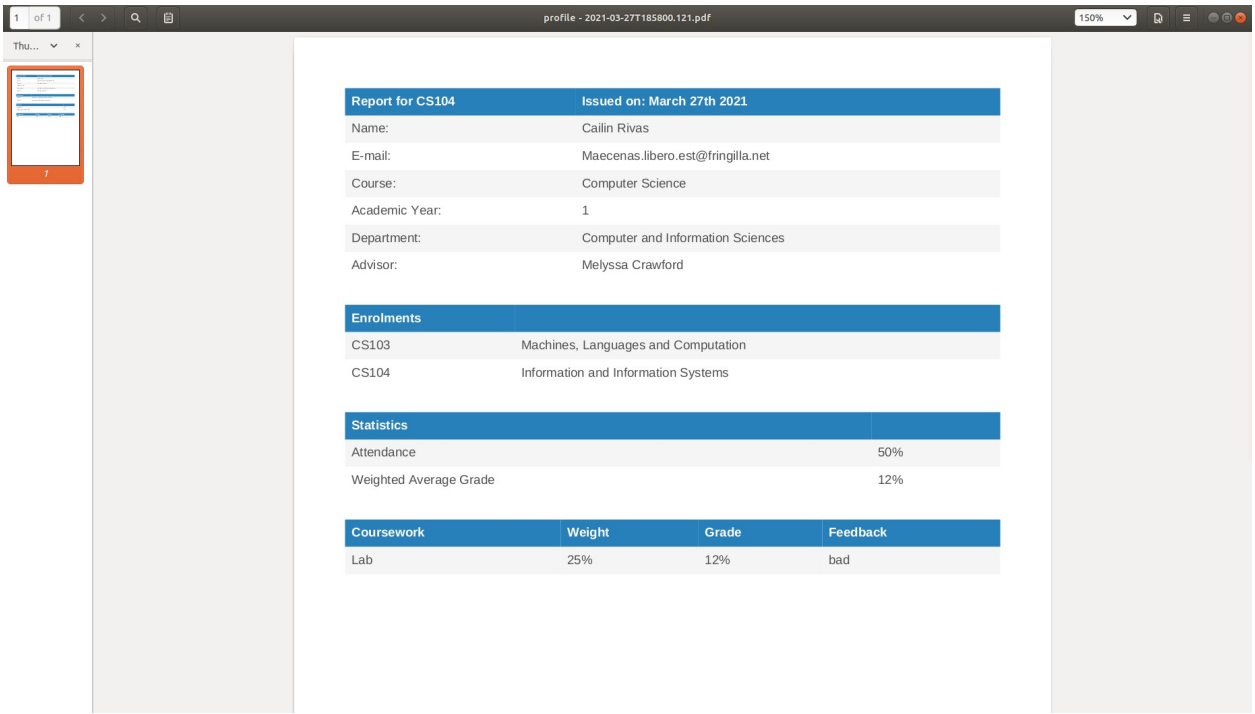
Clicking on the export button will download a PDF document which will contain the information in the profile.



Appendix K.5 - Viewing Class Profile



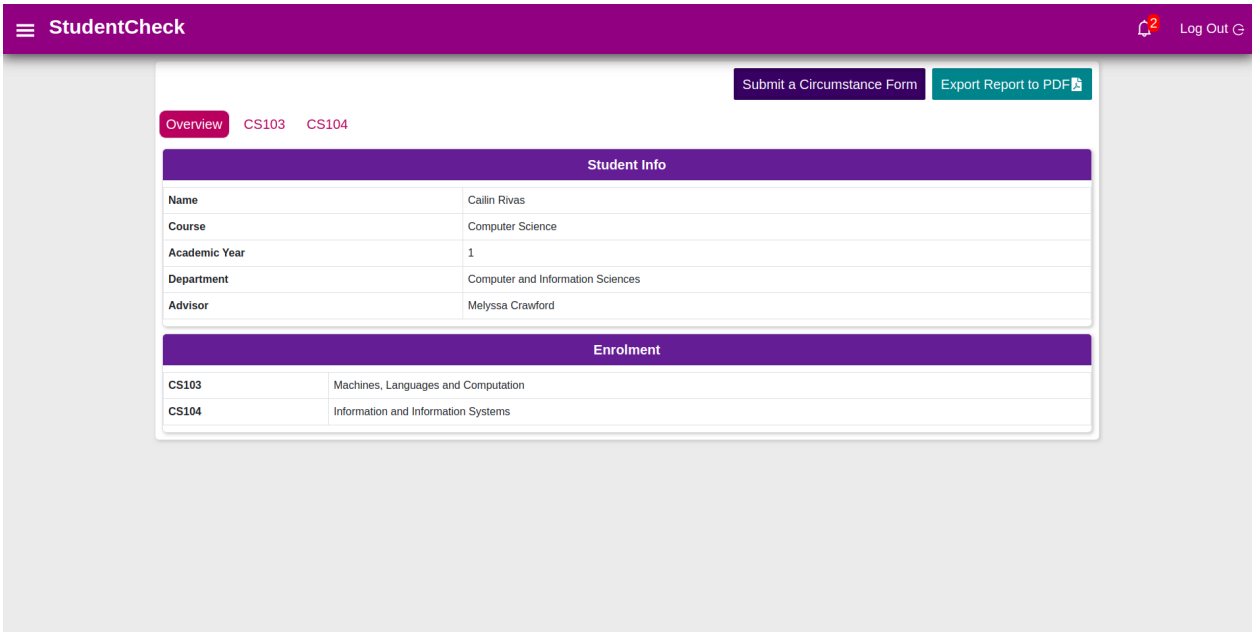
Appendix K.5 - Academic Info Tab



Appendix K.5 - PDF Generated from Class Profile

Departmental and Advisee profiles will contain a toggle for the classes so that the

user can see all cross-module data for the student.

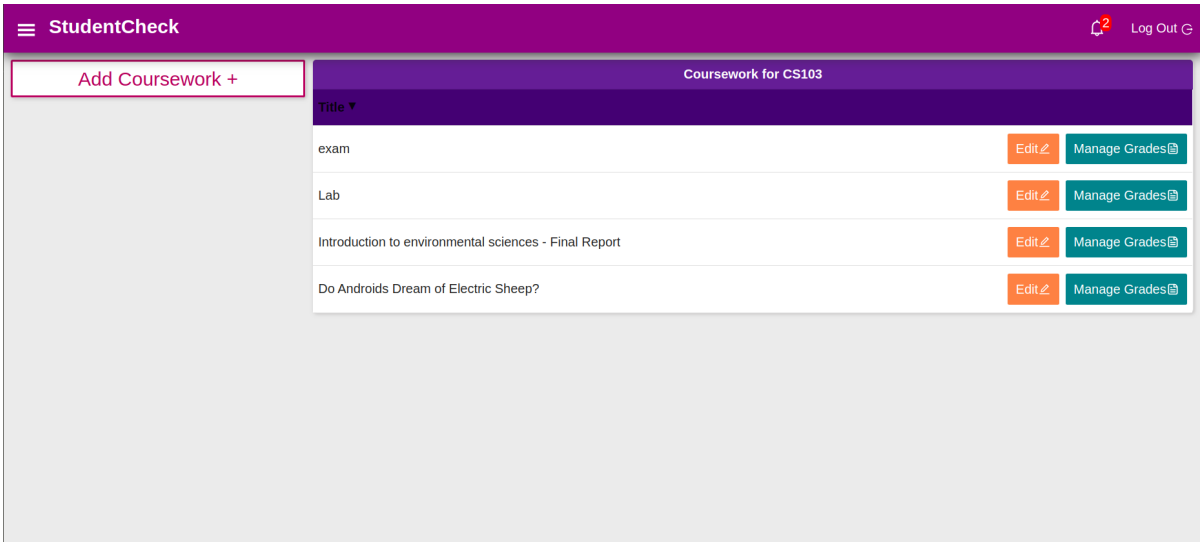


Appendix K.5 - Viewing Department Student Profile

Appendix K.6 Coursework Management

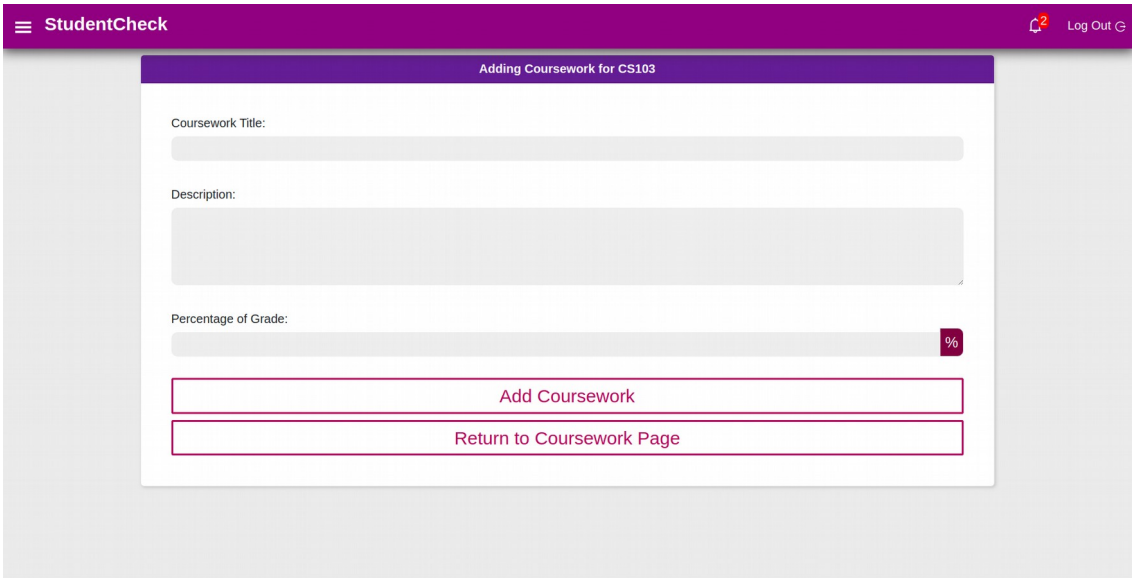
The coursework management pages consist of tables of courseworks added for a class. By clicking the Title column in the table, the coursework titles will be sorted into ascending or descending order.

Clicking on the Add Coursework button will navigate the user to the page to add a new coursework for the class. The Edit button will navigate the user to the page for editing an existing coursework, and the Manage Grades buttons will take the user to the page for adding and amending grades for that coursework.



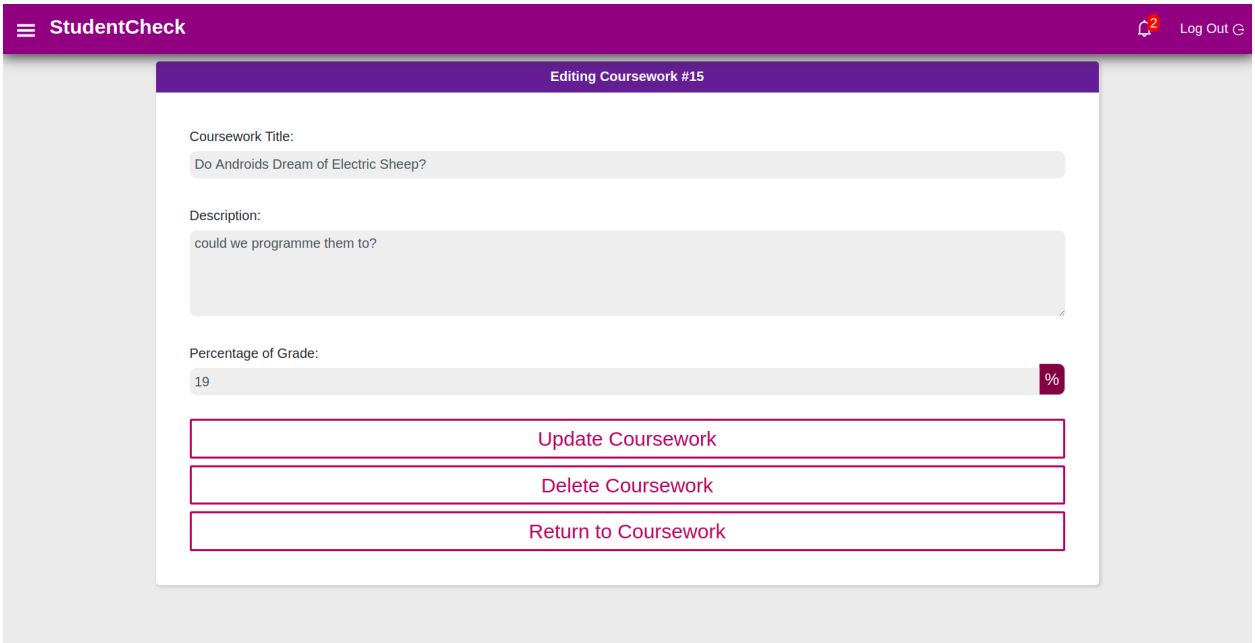
Appendix K.6 - Manage Coursework Page

The Add Coursework page presents the user with a form where they can add and submit a new coursework. Any errors that occur due to the user input will be displayed under the appropriate fields.



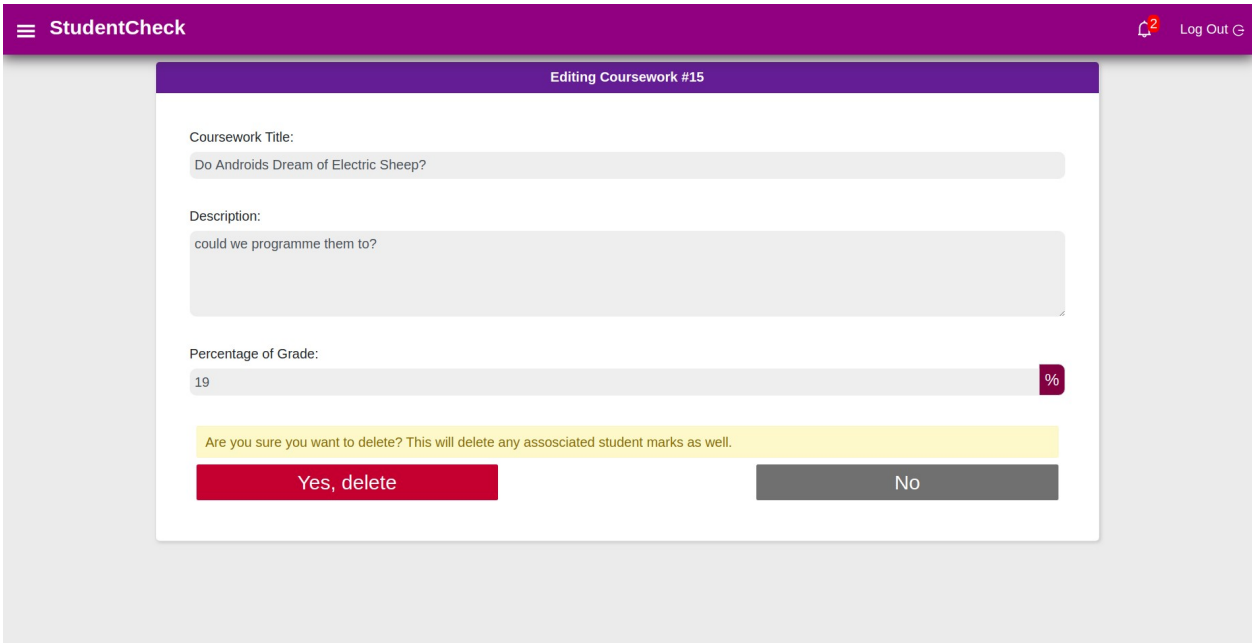
Appendix K.6 - Add Coursework Page

The Edit page for a coursework allows the user to change any of the values for the coursework. If any errors occur the user will be presented with an appropriate error under the corresponding input field.



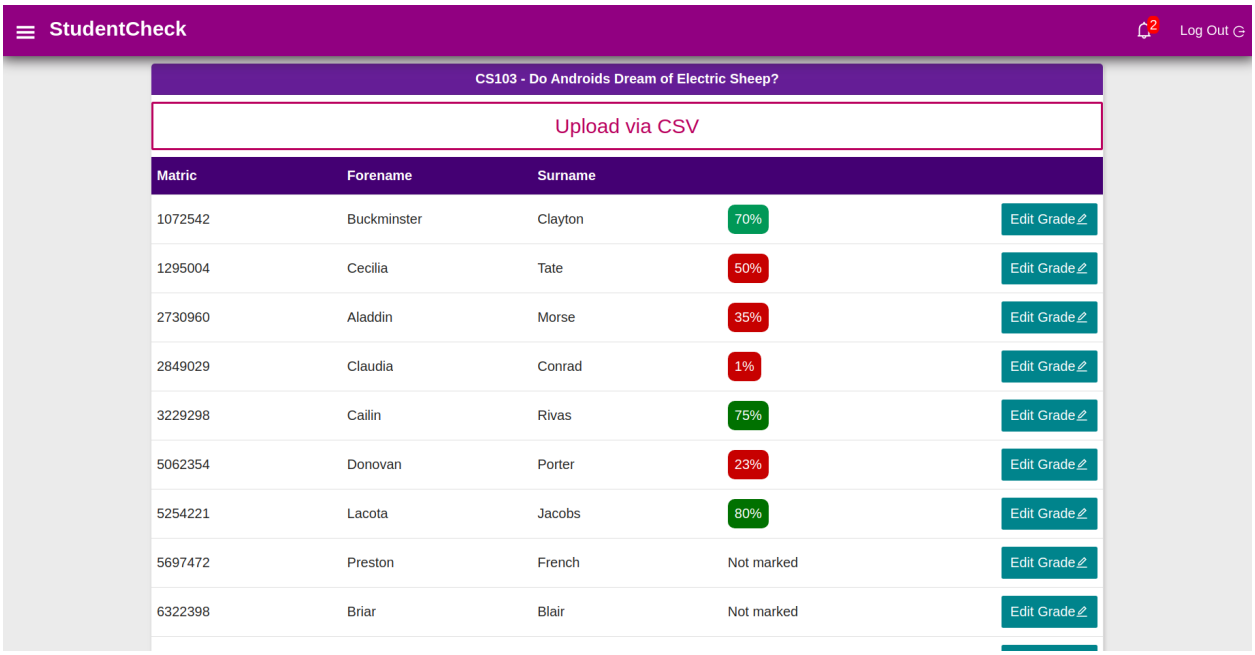
Appendix K.6 - Edit Coursework Page

Here the user can opt to update any of the values in the inputs or delete the coursework, which will result in a two-step verfication before the coursework is deleted.



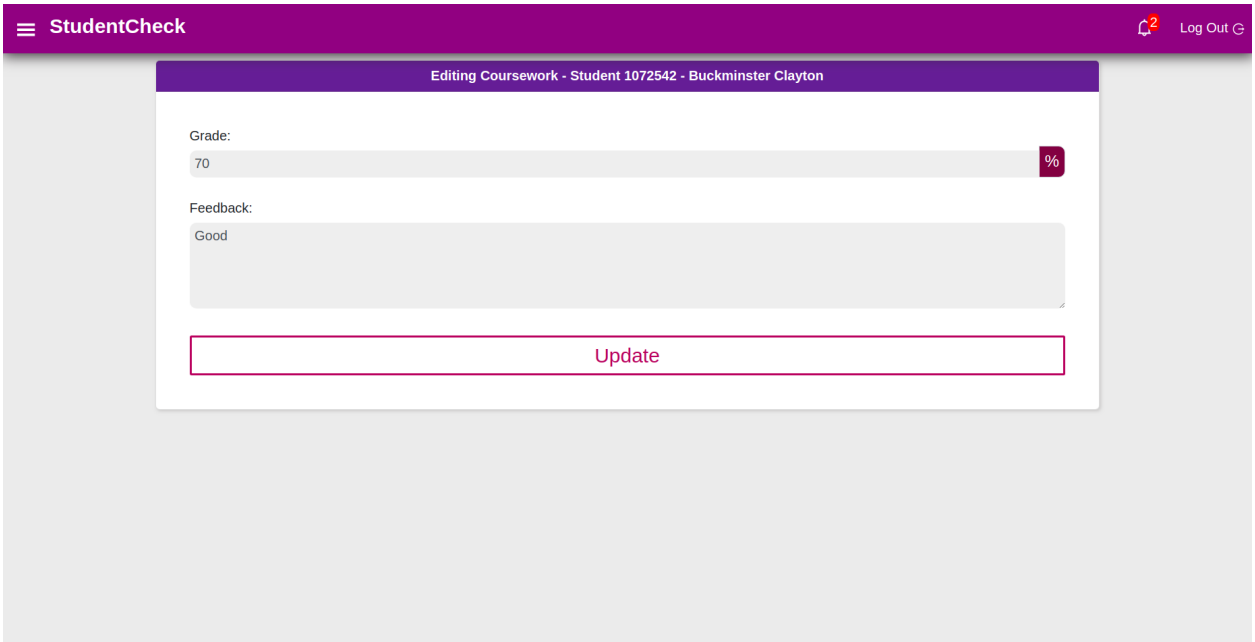
Appendix K.6 - Delete Coursework Option

From the coursework table, if the user selects to ‘Manage Grades,’ they will be displayed a list of students and their marks.



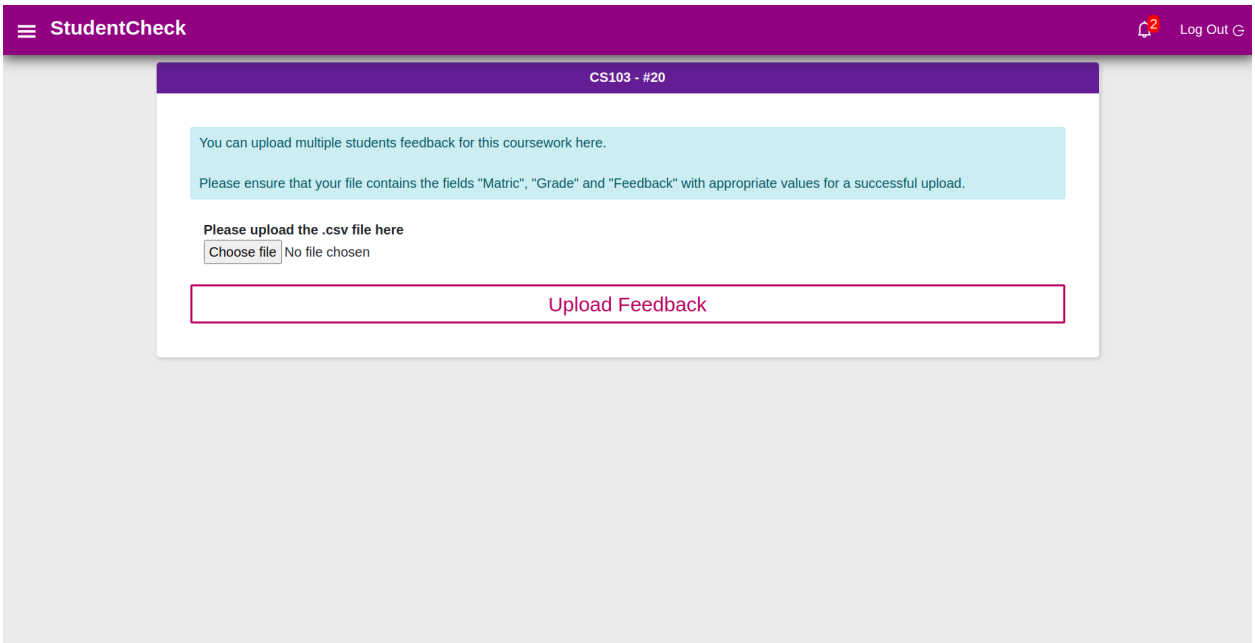
Appendix K.6 - Student Grades Page

The user can select a student's 'Edit Grade,' button to individually add a mark. This must pass verification checks before the mark is submitted. If a mark is already present, the details of the grade will be the inputs default values.



Appendix K.6 - Edit Grade Page

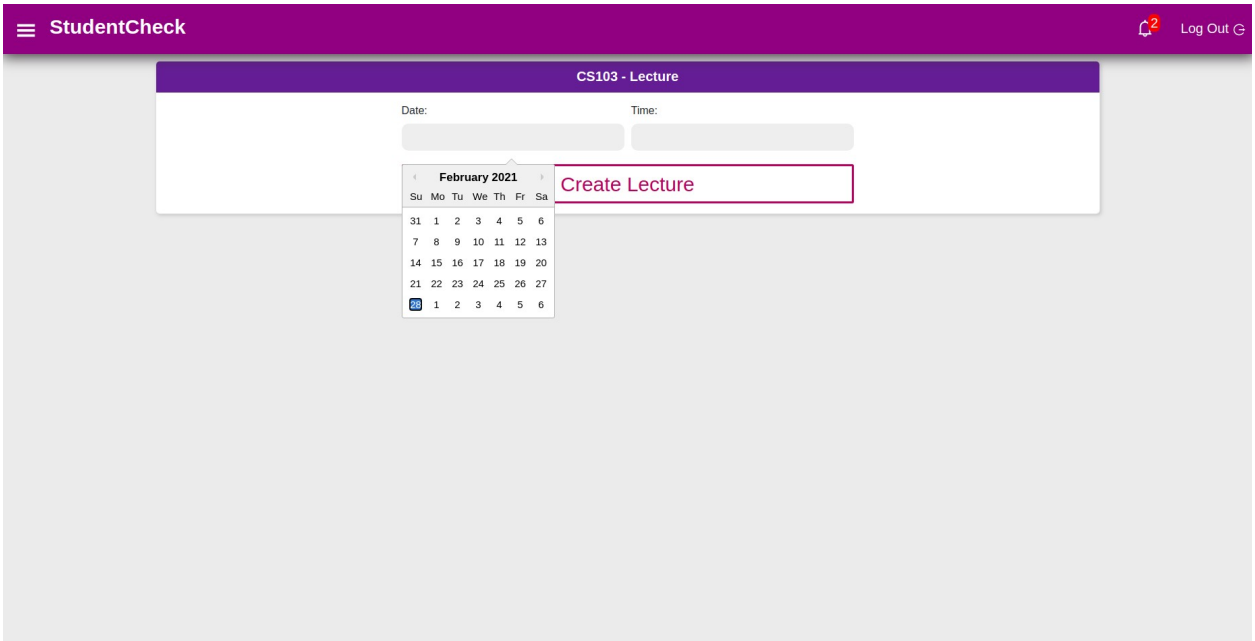
If the user clicks the 'Upload via CSV' button, they will be presented with a page to upload a valid CSV file. This CSV file must pass the file type check, and checks to see that the file has appropriate headers before it can be submitted.



Appendix K.6 - Upload Grades CSV Page

Appendix K.7 Attendance Management

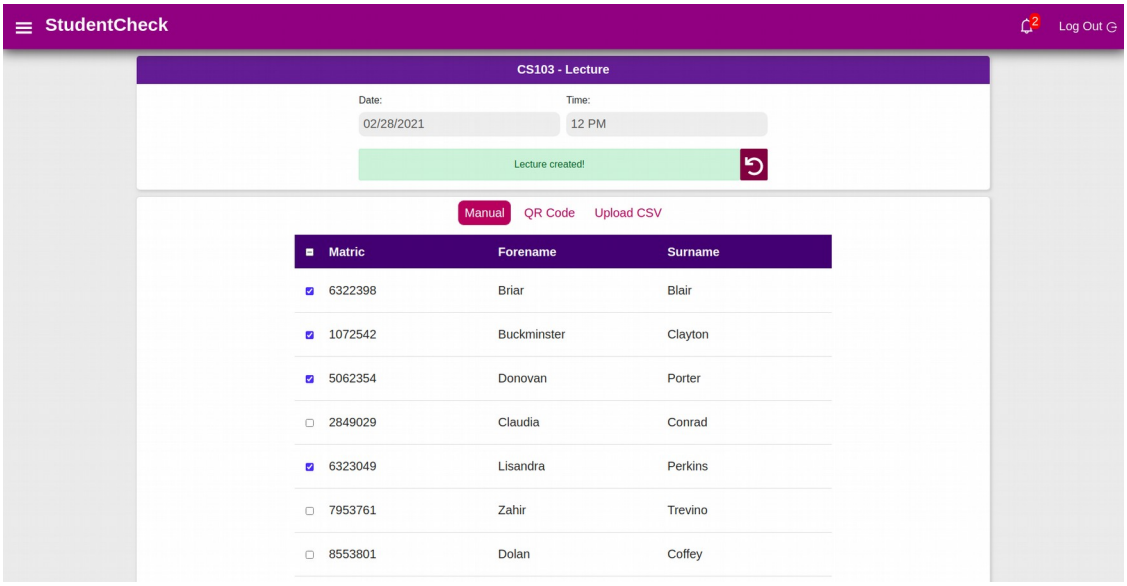
If the user navigates to an attendance page for a given class, they will be presented with two inputs to select a time and date. These are limited to a year before and the current date.



Appendix K.7 - Attendance Page

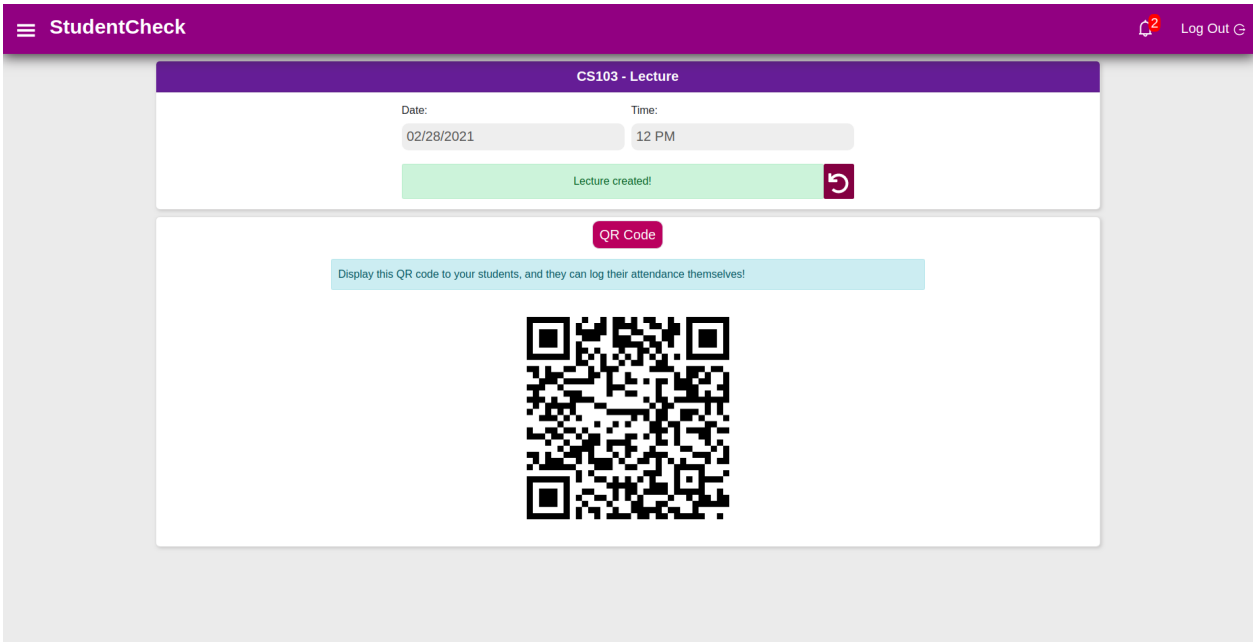
When a valid date and time is submitted, the user will be presented with Tabs to indicate the different methods of taking attendance.

The first is a table with selectable rows with a submit button at the bottom of the register.



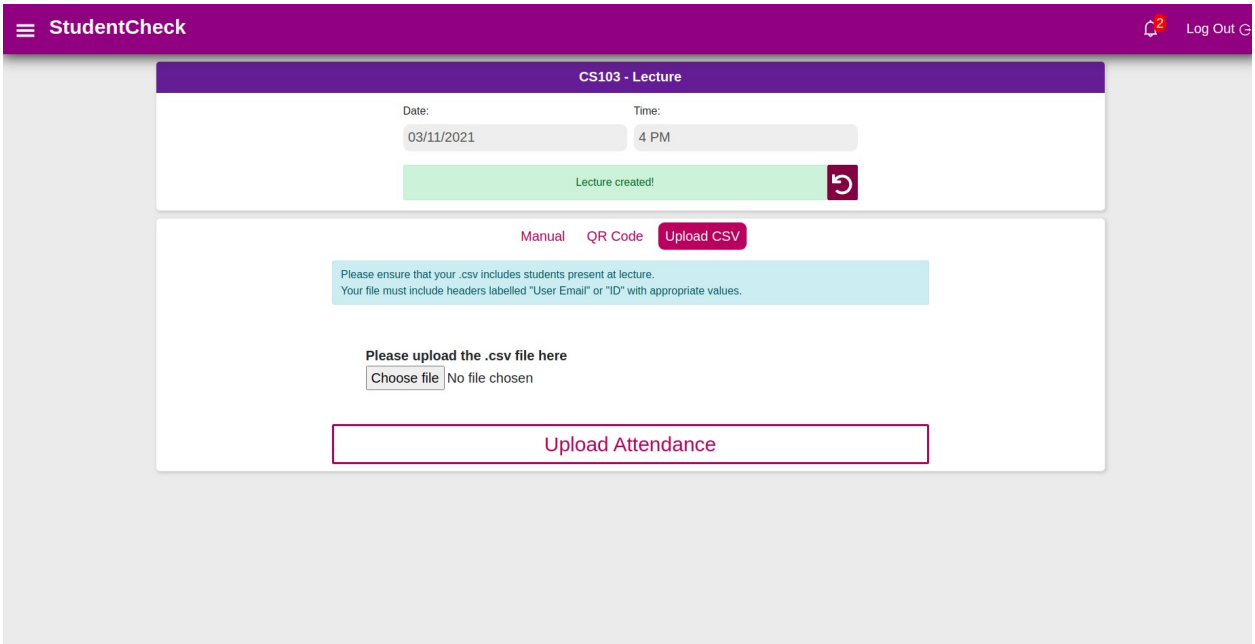
Appendix K.7 - Checklist Attendance Page

The second option is a QR code that can be scanned by students to upload attendance themselves.



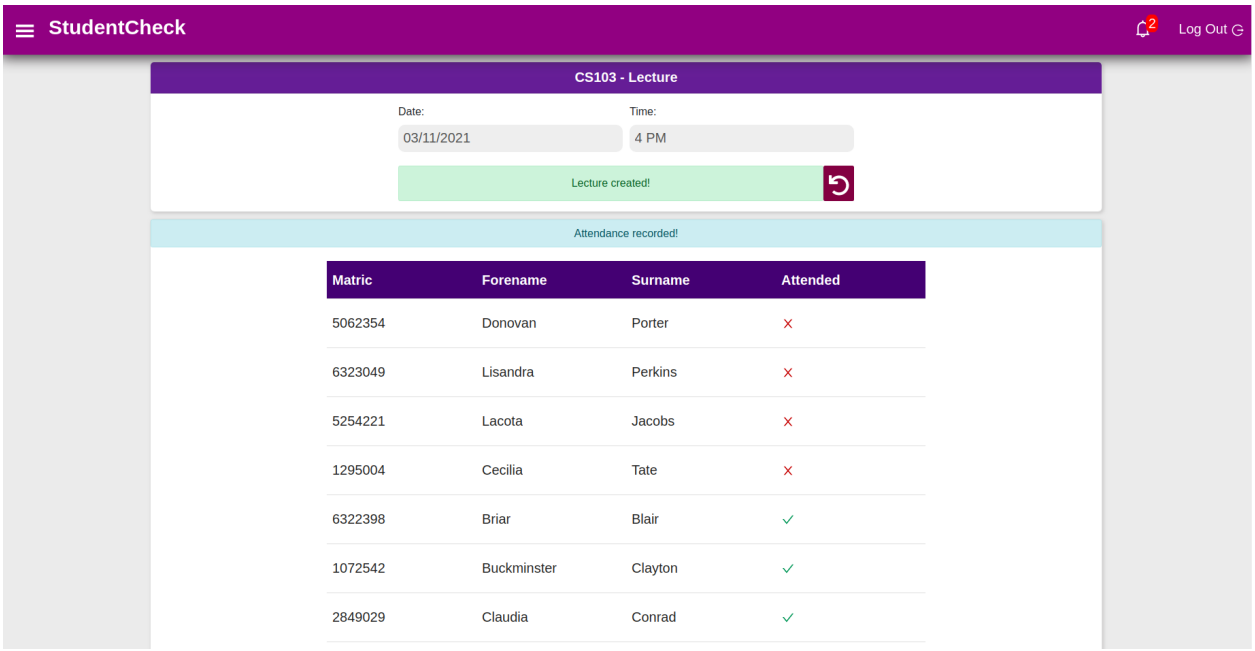
Appendix K.7 - QR Attendance Page

The final option is to upload attendance via CSV file. This must pass verification checks that it is of CSV format, and contains the correct headers.



Appendix K.7 - Attendance CSV Page

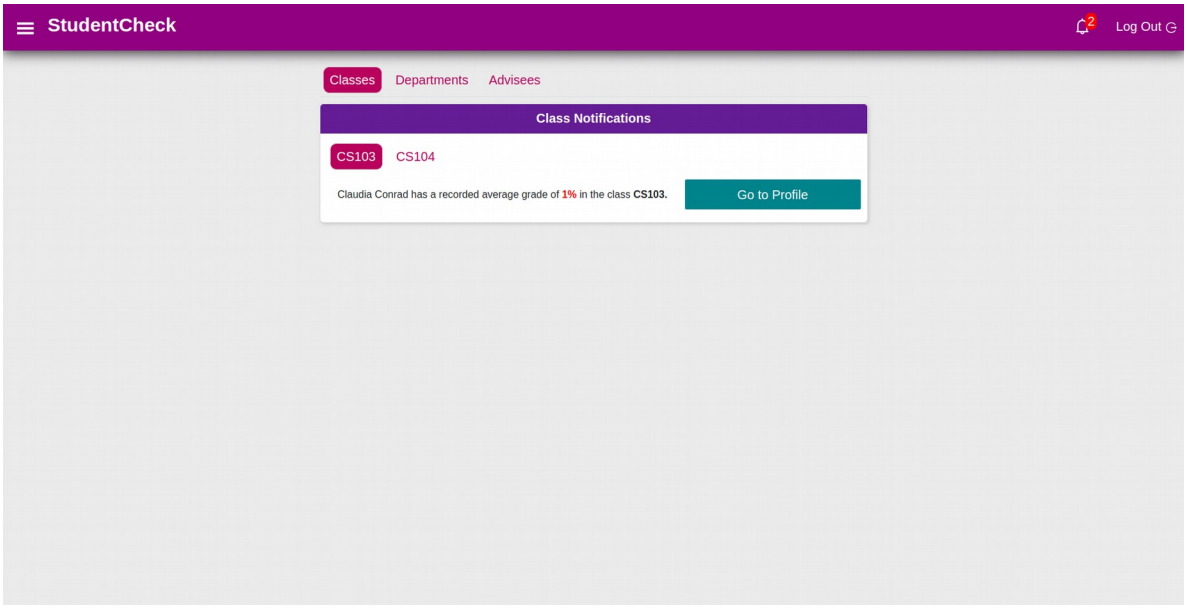
For the checklist and CSV methods, when attendance is submitted, the user will be presented with the register that has been submitted.



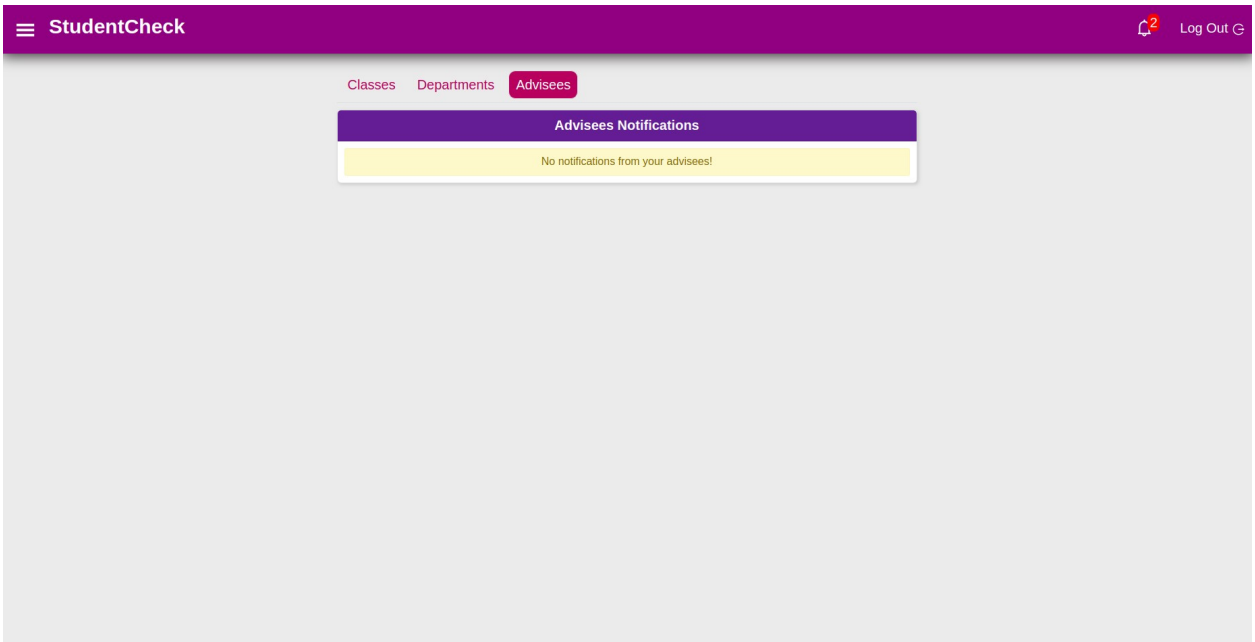
Appendix K.7 - Attendance Added Page

Appendix K.8 Staff Notifications

Navigating to the notifications page will show any students with very low attendance or weighted grades statistics. There are tabs for the different staff roles, and if there are no notifications a message will be displayed.



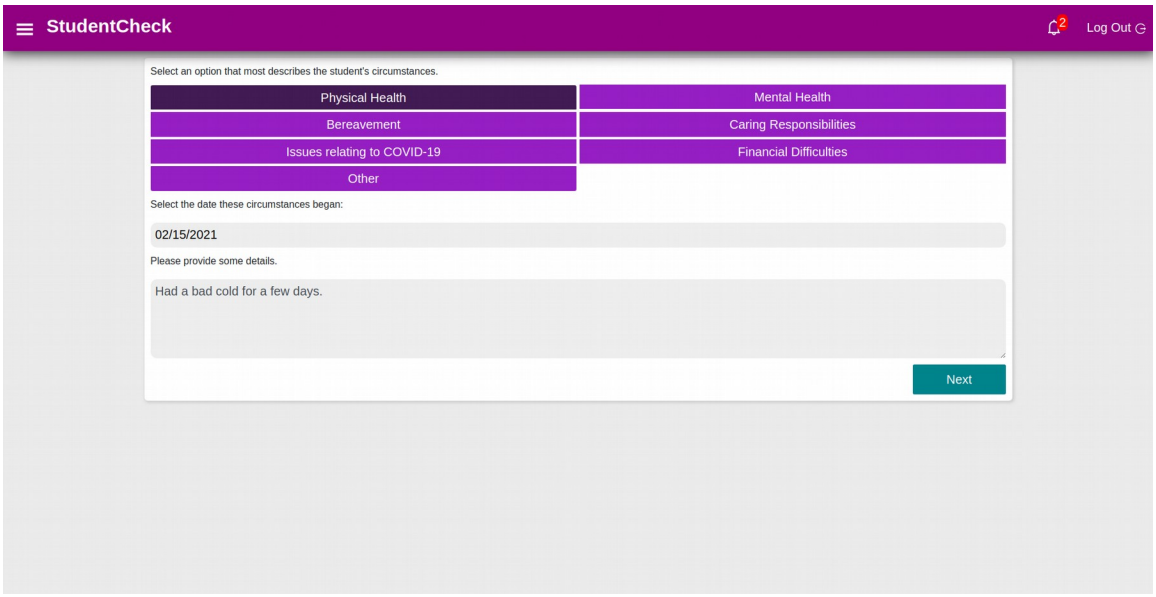
Appendix K.8 - Staff Notifications Page



Appendix K.8 - No Notifications Page

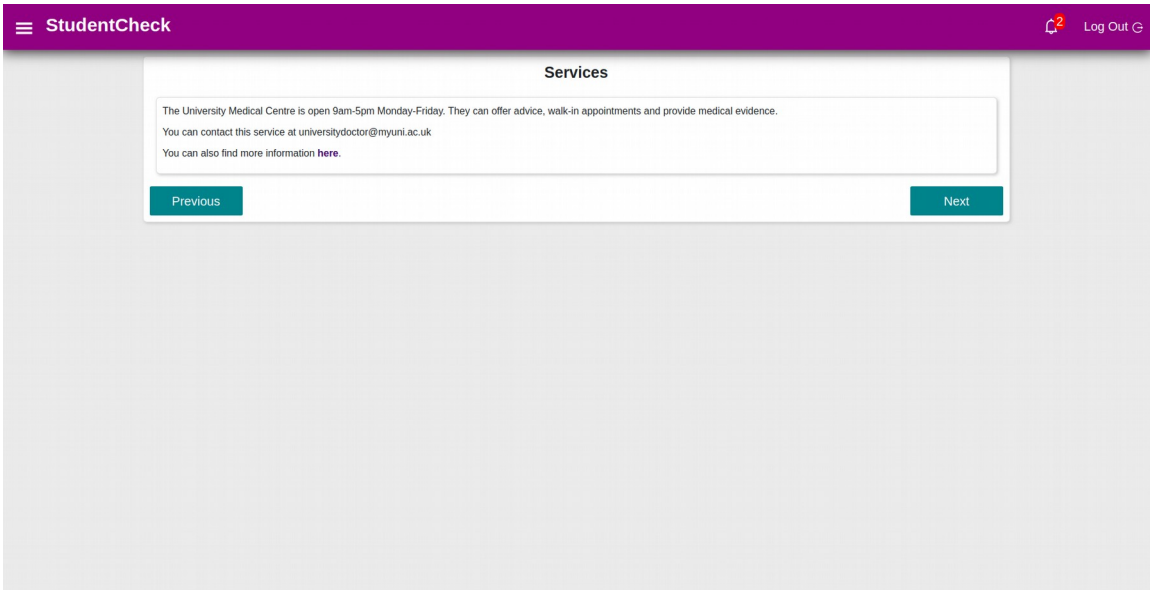
Appendix K.9 Circumstance Forms

These forms are available for both staff and student users. The first page will offer the user the option to input details about a students circumstances that are affecting their studies. The form must pass verification checks before the user can proceed to the next page.



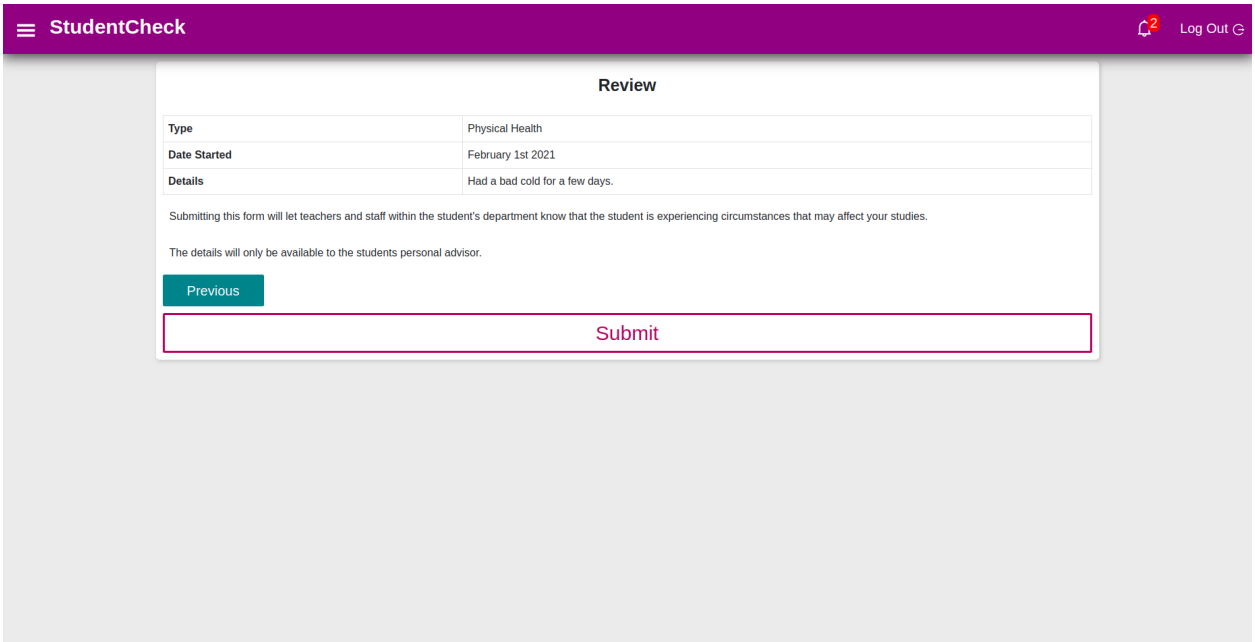
Appendix K.9 - Circumstance Form Page 1

After the checks are passed, the user can navigate forwards to be recommended any available services depending on their circumstance.



Appendix K.9 - Circumstance Form Page 2

The final page of the form will offer the user the chance to review their submission before submitting.



Appendix K.9 - Circumstance Form Page 3

Appendix K.10 Student Navigation

When a student successfully logs in, they will be presented with a sidear that can be controlled though a ‘hamburger’ icon in the top left. Their homepage will be a profile containing their cross-module information.

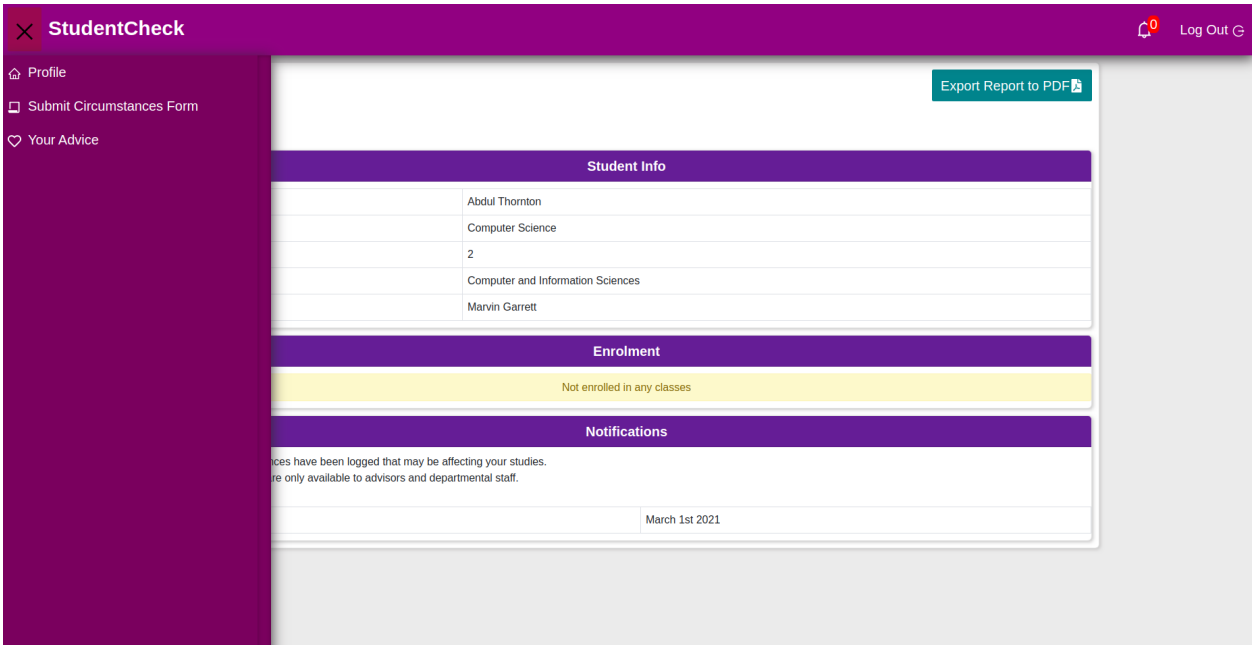
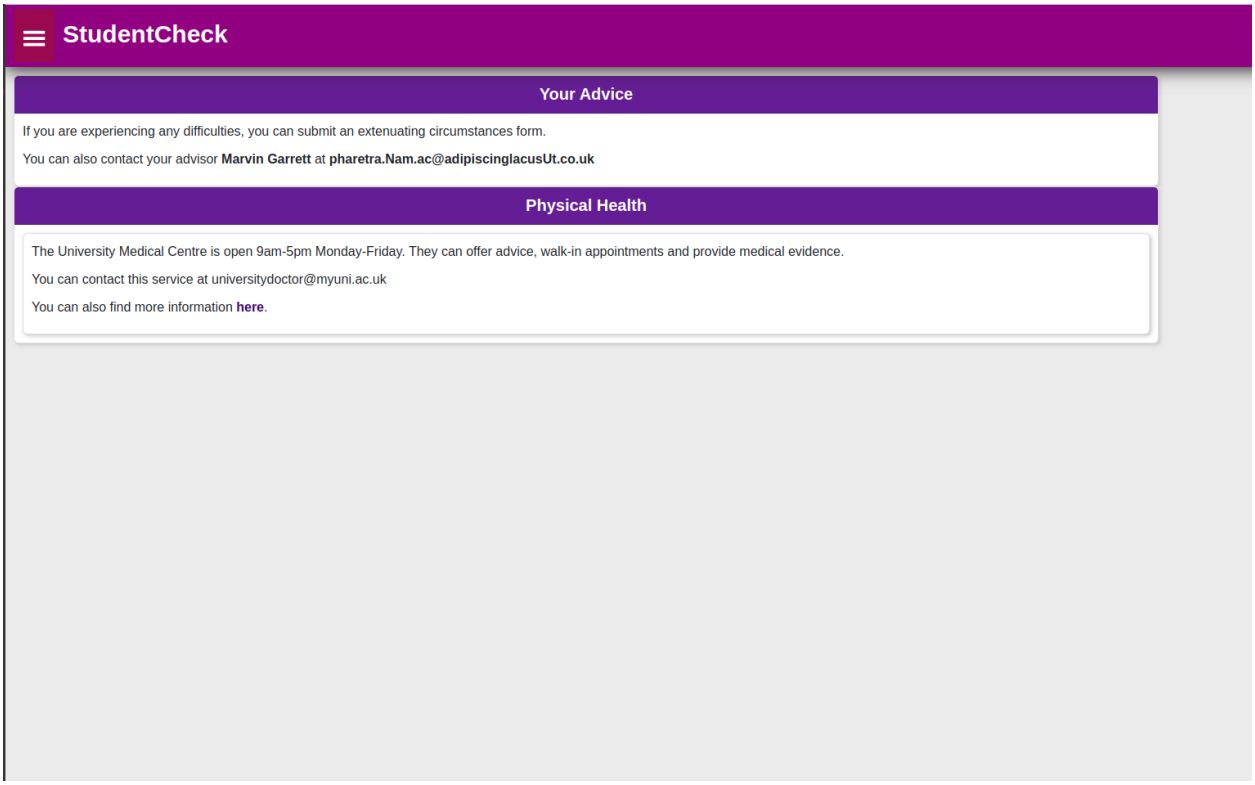


Illustration 2: Appendix K.10 - Student Navigation

Appendix K.11 Student Advice

The student advice page, accessible via the navbar will display any services available at the university based on the extenuating circumstances submissions the student has made.



Appendix K.11 - Student Advice Page