A Web Application for The School of Computing Science Exam Boards

Hannah Hughes

School of Computing Science

Sir Alwyn Williams Building

University of Glasgow

G12 8RZ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at the University of Glasgow

02/09/2022

Abstract

This project develops a web application for The University of Glasgow School of Computing Science. The web application is designed to replace an existing system that manages student grades in preparation for exam board meetings. The current system is slow and has an outdated design, making it frustrating for Class Heads to use and since no other software can cater to the unique and complex grading rules, a new web application needs to be developed.

Interviews were conducted with users to collect requirements for the new system and after review, Django and React were chosen for development. To evaluate the success of the project, users were given a demonstration and completed a survey. These results showed that the design of the web application was a significant improvement to the current application, but the speed of the web application needs to be improved.

This project has successfully created a starting point for a new web application for exam board preparation and will need to be developed further to fully replace the current application.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic form.

Name: Hannah Hughes Signature: 

Acknowledgements

I would like to thank Dr Gerardo Aragon Camarasa for being my supervisor and providing me with this project. His guidance and support have been incredibly valuable and enabled me to fully explore and enjoy this project.

Contents

Chapter 1 Introduction

Glasgow University currently has a web application for managing student grades in preparation for exam board meetings and external validation. However, this system has become antiquated and has several bugs, which makes it frustrating for the Class Heads to use. Since its conception, it has been modified for various special cases and has mutated from its original purpose with some permutable functionality hard coded. It was built on a legacy framework and, as a result, is slow and has an outdated design.

1.1 Aims and Objectives

This project aims to create a new web app that will replace the Exam Board Preparation page on the old existing system. The new app should have an up-to-date user experience and provide all the important information that the old system provides more intuitively. Furthermore, since the new app will be passed on to a maintenance and support team for the university, the backend should be understandable and written in Python. The old system had several points in the overall process that relied on one person for progression. This needs to be avoided with the new system, removing single points of failure.

1.2 Report Structure

Chapter 2 discusses the process taken to gain a list of requirements for this project and provides the prioritised list.

Chapter 3 details design decisions and introduces low-fidelity wireframes outlining basic design features.

Chapter 4 discusses the methods used to test the application during development and at the end of the project to evaluate its success.

Chapter 5 concludes this report with a discussion on possible future work and a personal reflection.

Chapter 2 Background: User Requirement Capturing

2.1 The Existing System

The current web application was initially built in the early 2000s when the number of computing science students was relatively low. In recent years, however, the number of students enrolling with the school has rapidly increased, and as a result, the current system has had to expand.

Although it has been refactored several times, decreasing its loading time, loading student grades for one cohort (approx. 180 students) takes almost 3 minutes. This can be especially frustrating when trying to locate a particular piece of information and having to load multiple pages.

In addition to fashion changes in application design, the application has been incrementally mutated and, as a result, has become cluttered and busy. There is no single theme for the design, and consequently looks disjointed.

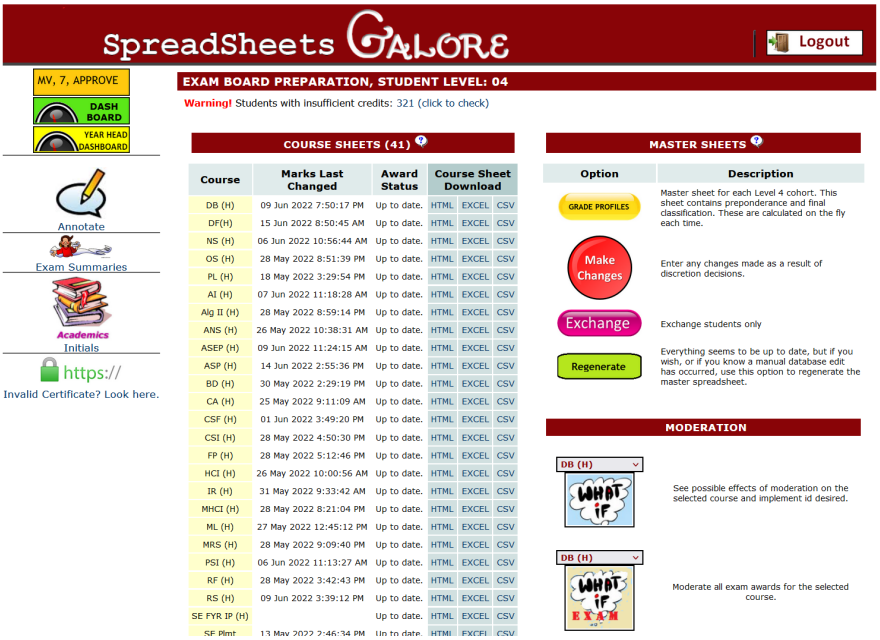

Figure 1: Design of Existing System

As university policy changes, the rules used to calculate student grades have also changed. The current system has no way for users to edit these hard-coded rules, so all changes must go through the system owner. This can be time-consuming and frustrating, especially if the system owner is unavailable.

There is often a tight turnaround time between exams finishing and the exam board meetings. During this time, Class Heads must review and consolidate all the decisions made regarding student progression and degree classifications. This application should be fast, easy to use and provide all useful information for decisions made in the meetings.

2.2 Similar Software

Grade management software is commercially available but mainly optimised for high schools.

Much of the software researched, including Alma[1], Thinkwave[2], JumpRope[3], Schoology[4], Rediker[5] and GradeBookWizard[6] had an outdated design with simple HTML table views. These are also not equipped to comfortably display and manage the volume and complexity of Glasgow University's School of Computing Science students and courses.

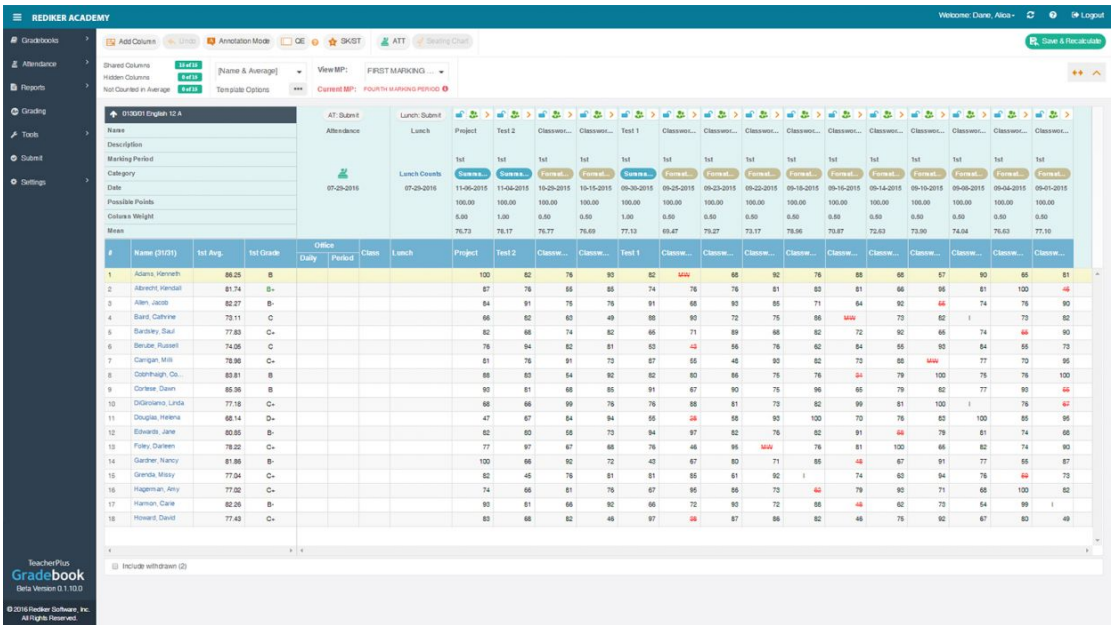

Figure 2: Rediker Software Example[5]

The university currently uses Moodle for some student management. This however, like these other commercially available software options, is not able to manage the complex and unique grading policies to calculate overall grades and final classifications. For these reasons, a singular tailored web application is required.

2.3 Requirement Gathering Interviews

Owing to confidentiality of student information, the old system could not be used in depth to create requirements. Instead, three users were interviewed with a standardised list of questions which can be found in Appendix B. These interviews lasted 30 minutes each and from them a list of requirements was created.

2.4 User Requirements

Requirements were gathered, duplicates removed, and then prioritised using the MoSCoW system[7]. The resulting list was as follows:

Must Haves:

- As a User, I want to view all the grades for a student across all their courses.
- As a User, I want to view all the Exam grades, split into questions, for a particular course.
- As a User, I want to view the weighting of assignments and exams for a particular course.
- As a User, I want to view all the student grades for a given course.
- As a User, I want to view all the students in a cohort.
- As a User, I want to view the aggregated grade for a student which includes taught class grades and their project grade.
- As a User, I want to view the taught grade for a student.
- As a User, I want to view the project grade for a student.
- The System must hold at least 1000+ students per year.

Should Haves:

- As a User, I want to view the percentage of students with a mark below band D3.
- As a User, I want to export information in a way that can be taken into exam board meetings.
- As a User, I want to highlight or easily identify students with preponderance.
- As a User, I want to uplift or lower the marks for a course.
- As a User, I want to uplift or lower the marks for an exam question.
- As a User, I want to view the course summary given by the lecturer.
- As a User, I want to mark a student's preponderance.
- As a User, I want to quickly identify students who take an incorrect number of credits.
- As a User, I want to view the rules used to calculate a student's degree classification.
- As a User, I want to view degree classification of a student if they are in an exit year.
- As a User, I want to search for students or courses by name or id.
- As a User, I want to view graphs to compare a course with other courses.
- The System must load all views and pages quickly.

Could Haves

- As a User, I want to compare the grades for a course to the grades from the previous year.
- As a User, I want to view the weighted percentages of grades at each band for a student.
- As a User, I want to highlight or select a row in a view.
- As a User, I want to quickly look at a particular course's information and grades overview from a student view.

- As a User, I want to quickly look at a particular student's information and grade overview from a course view.
- As a User, I want to know the percentage of students in each grade band in a particular year.
- As a User, I want to view a graph to compare student grades on questions in an exam.
- As a User, I want to identify students that are not able to progress to the next year.
- As a User, I want to leave comments on a student's information.
- As a User, I want to edit the rules used to calculate degree classifications.
- As a User, I want to view the GPA of a student across each of the years that they have completed.
- The System must support at least 20+ users simultaneously.

Would Haves

- As a User, I want to use either student ID as digit only or digit and letter.
- As a User, I want to export an excel sheet of all students and student grades.
- As a User, I want to quickly identify students who may be eligible to submit a good cause.
- As a User, I want to hide columns from views.
- As a User, I want to quickly identify grades that are likely to be erroneous.
- As a User, I want to hide or censor students who should not be considered for exam boards
- As a User, I want to check that all students have been given a mark or preponderance for each coursework or exam question.

Chapter 3 Design and Implementation

3.1 Frontend Design

Once logged in, the user is presented with a dashboard. The dashboard contains useful information about the cohorts that the user manages. On the left-hand side, there is a Sidebar that is constant throughout all pages for primary navigation through the application. Additionally, on the dashboard, there is a search bar that can be used for quick navigation to students or courses.

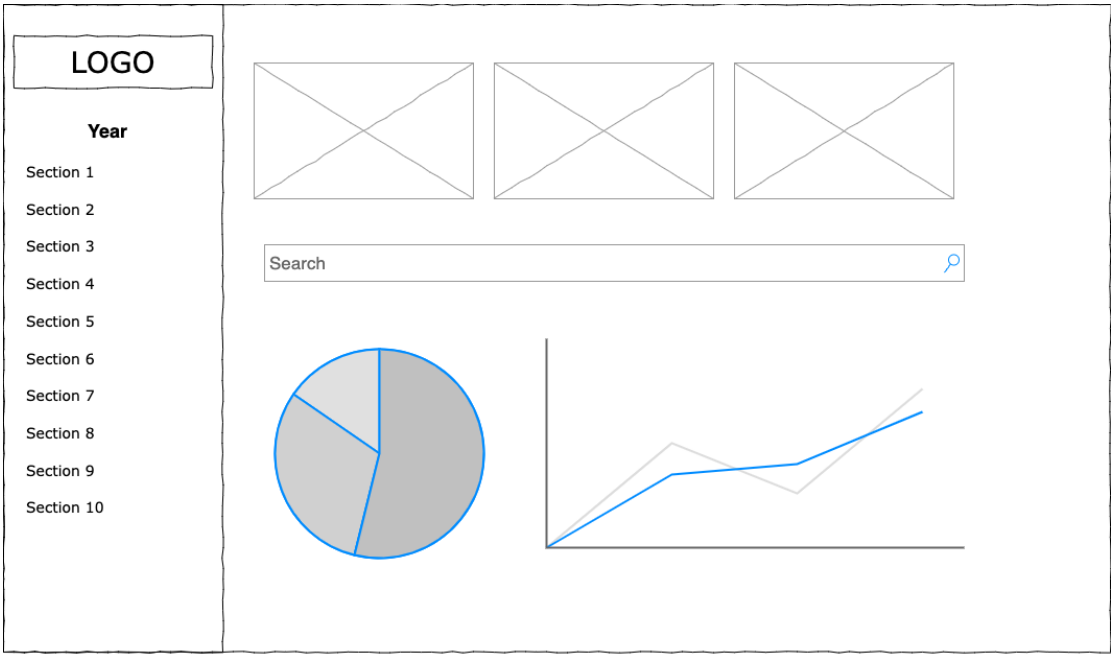

Figure 3: Dashboard Wireframe

The course page shows the lecturer comments and information followed by a useful scatter chart and grade distribution. At the bottom of the page, students and their associated grades for this course are displayed in a table. Clicking on one of these students opens that particular student page in a new tab.

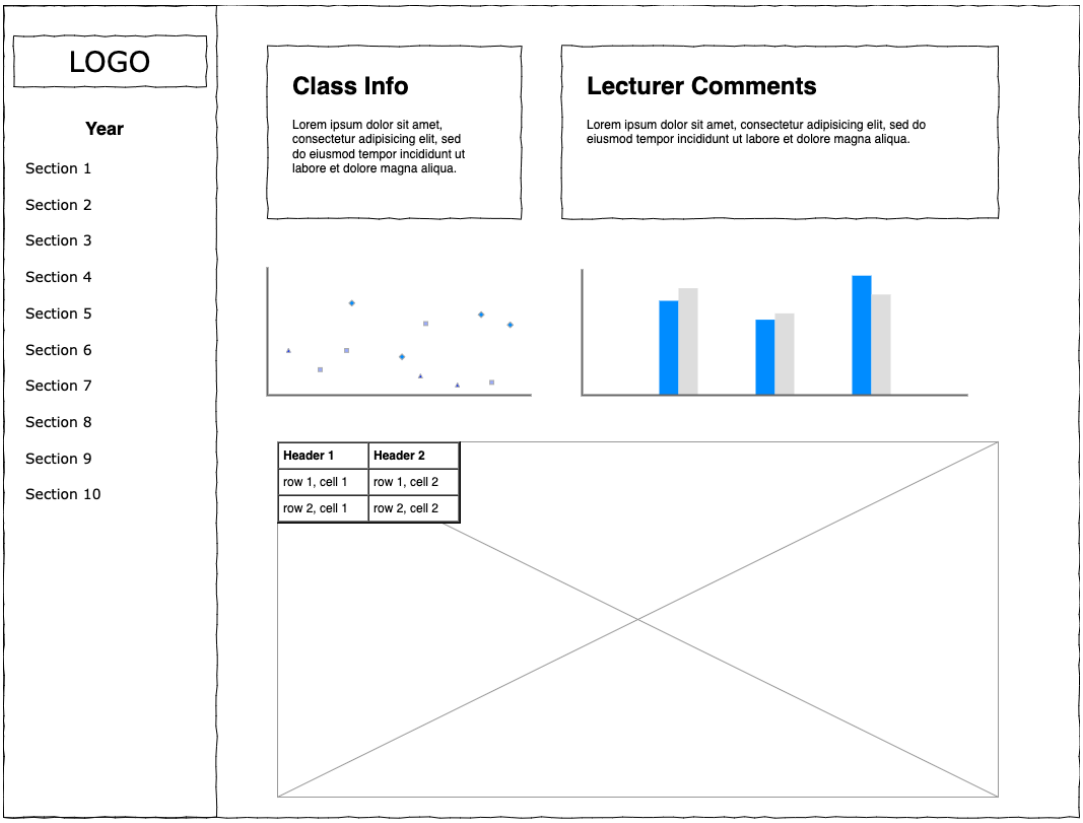

Figure 4: Course Page Wireframe

The student page contains all the grades across all the completed years for that student. Additionally, users can leave comments to a student page about any changes to marks or other identified issues, creating a record that can be referenced later.

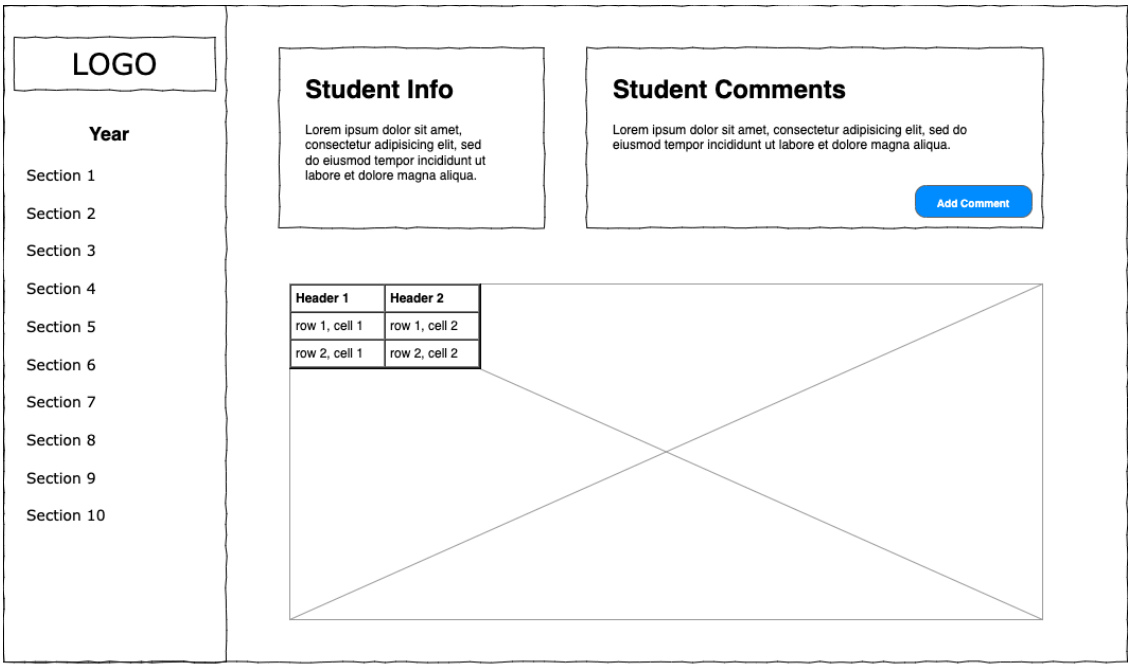

Figure 5: Student Page Wireframe

The colour theme throughout the web app was chosen from the university brand colour palette[8]. This was so there is a harmonious integration with other university systems. In keeping with the modern and simplistic design, there is not much use of colour, mainly on graphs, icons, and text. However, when using the exact match for colours from the palette, they were too dark and difficult to distinguish in some areas. For this reason, colours from the secondary palette were used. To create a full palette with 5 distinct colours for the pie chart, two extra colours were generated. To do this, coolors.co[9] was used with three original colours, Turquoise, Rose, and Pumpkin.

3.2 Database Design

Below is an entity relationship diagram for the project. Identifying the entities and their relationships between each other is essential for good database design. Each entity also has associated attributes, these can be found in Appendix A

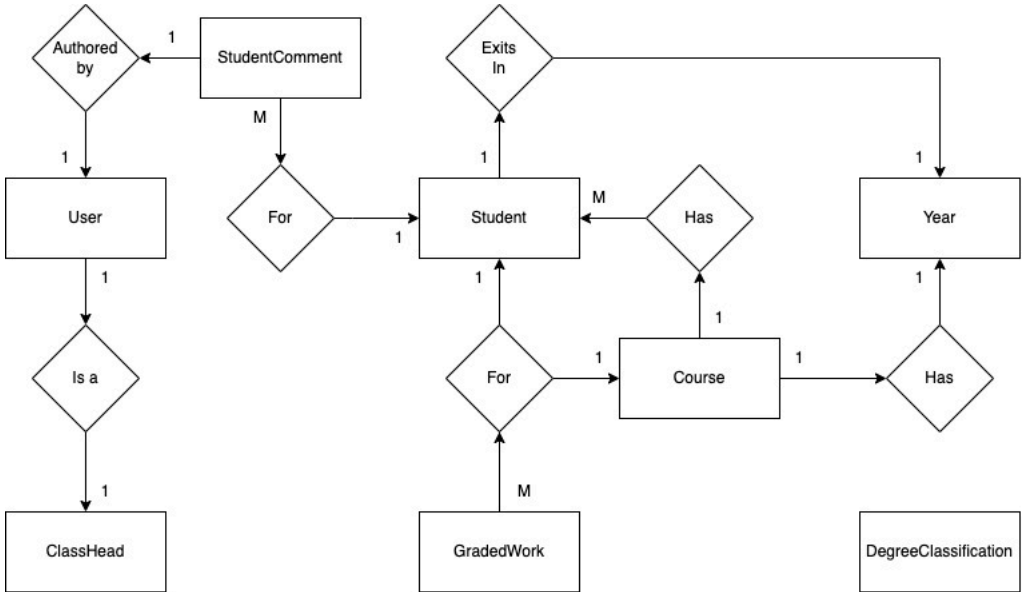

Figure 6: Entity-Relationship Diagram

3.3 System Architecture

The following figure shows a system architecture diagram for the web application.

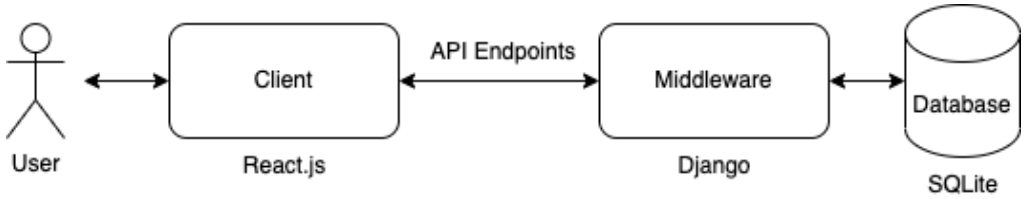

Figure 7: System Architecture Diagram

3.3.1 Django

Several frameworks were considered for the development of the web application. Since the maintenance team are largely familiar with Python, it was requested that the framework be Python-based. Among consideration were Django, Flask and FastAPI.

Django is well established and provides a highly structured layout. However, for this reason, it can be considered cumbersome and has a risk of becoming antiquated [10]. Since it is well established, it has a large online community for support and learning. Flask encourages a simplistic and flexible environment which can improve performance but also results in reliance on third-party extensions for some functionality [10]. This could be detrimental to the security of the application. Finally, FastAPI is the newest and fastest framework considered[11]. However, since it is only 3 years old, there is limited online support.

Django was chosen to develop this application because the high level of documentation will enable maintainability and understandability for a wide range of users. The prescribed structured format will allow easy integration with future apps, and its established nature heightens reliability and security. The application was created with the latest version, Django 4.0.4, which is compatible with Python 3.10 [12].

3.3.2 React.js

React [13] was chosen for the development of the frontend mainly because of its modular structure. Parts or components can be reusable, allowing for easy scalability and fast development. This can be achieved by management of props, read-only variables, and state, objects that store data about the component, passed from parent components down to child components. React has a well-established community and a large variety of libraries that can be used to create high-quality pages with improved user experience. One such library, Material UI (MUI)[14] was chosen for data tables, icons, and buttons. Similarly, Recharts[15] was chosen for the visual representation of the data in the form of graphs and charts.

To achieve this, Django REST Framework [16] was used to create API endpoints which in turn were used by React to handle views.

3.3.3 Database

By default, Django uses SQLite as a database. Since this web app will be only tested with a small amount of mock data, using this default database is acceptable. In its final integration, the app will use the current university databases which will be more suitable for managing larger datasets.

Data for this app was provided via CSV files and uploaded through a temporary ‘upload’ page. In its final implementation, data input would be done through connection with the relevant database. However, for the purpose of this project, inputs provided via these files and the ‘Upload page’ remains proof of concept for future uploading functionality.

3.3.4 Authentication

Two methods of user authentication are token-based and session-based authentication. In session-based authentication, the server-side stores the authentication, whereas with token-based it's stored on the client side. Generally, session-based authentication is more secure, but tokens are considered faster and more easily scalable. Since this application will be used behind a VPN, JSON Web Token (JWT) authentication is acceptable. Token refresh and blacklisting will also be implemented to further increase security and avoid the possibility of a token being stolen and used by an unauthorised person [17].

3.4 Page Specifics

The top section of the dashboard shows the number of students that the user manages, the total number of courses, the count of instances of preponderance among their students and the average grade for these students as a band.

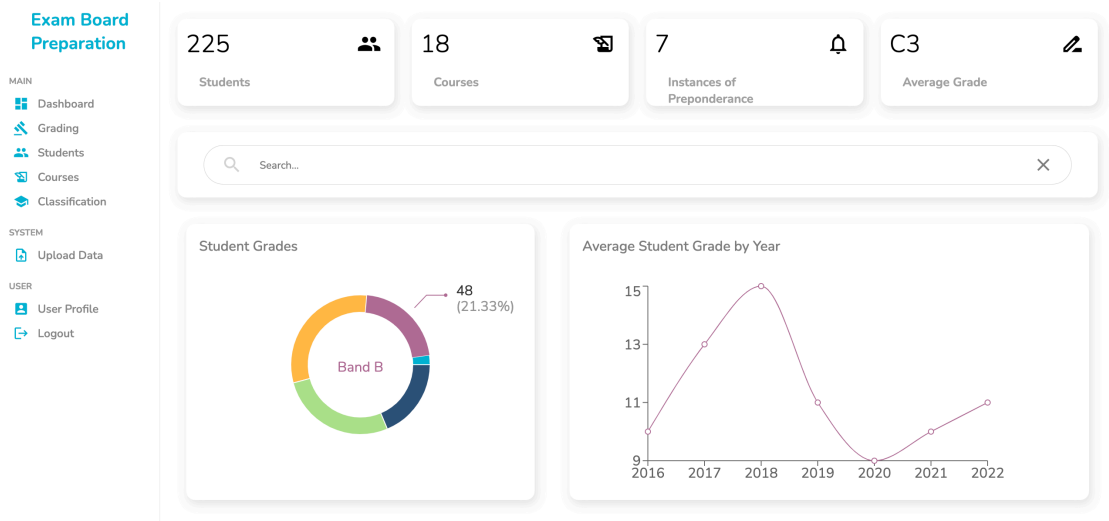

Figure 8: Dashboard

On the grading page, there is a reminder of the grades required for each band. Additionally, there is a section to view and edit the rules used to calculate a student's degree classification. Since these classification rules are often changed, a model in Django was created, Figure 9, so that this information can be stored and updated regularly. In the future, a Year foreign key could also be added to this model, allowing users to view the rules used in previous years.

```
class DegreeClassification(models.Model):
    classificationName = models.CharField(max_length=255)
    lowerGPAStandard = models.FloatField()
    lowerGPADiscretionary = models.FloatField()
    percentageAboveForDiscretionary = models.IntegerField(default=50)
    charBandForDiscretionary = models.CharField(max_length=5)

    def __str__(self): return self.classificationName
```

Figure 9: DegreeClassification Model

A new feature in the student view, that was not in the original app, is adding comments to a student page. This was identified in user requirement capturing interviews and is useful if a student has any issues or if their grades might be unusual in any way since users can leave a comment explaining the situation. These comments are automatically dated and marked with the name of the user who left them.

From the student page, where grades are shown against courses, there is a link to the course page and vice versa. This allows easy navigation for the user to get more relevant information allowing them to build a big picture quickly.

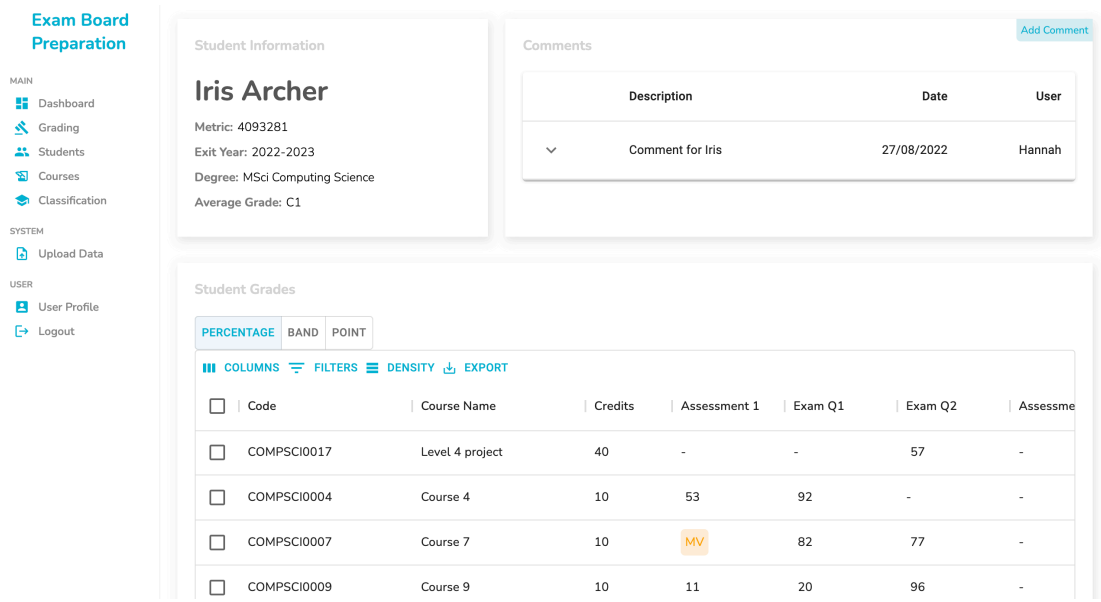

Figure 10: Student Page

Student preponderance can be marked at the bottom of the student page and is marked as yellow on the table of grades so that they can be easily identified, as seen in Figure 10. Marking a preponderance, for example, a Medical Void does not delete the original grade obtained for that coursework item. Instead, the grade is discounted from overall class grade calculations and the 'MV' label is shown instead of the grade mark. This is so that if the preponderance is removed or if needed for reflection, the original grade obtained remains unchanged.

On an individual course page, there is a doughnut chart, similar to the one on the dashboard, that shows the distribution of grades at each band. Additionally, there is a scatter chart that displays a point for each student and their mark for that class versus the mark that they got for all their other classes. This graph would ideally be linear and identifies how the class difficulty compares against other classes at that level.

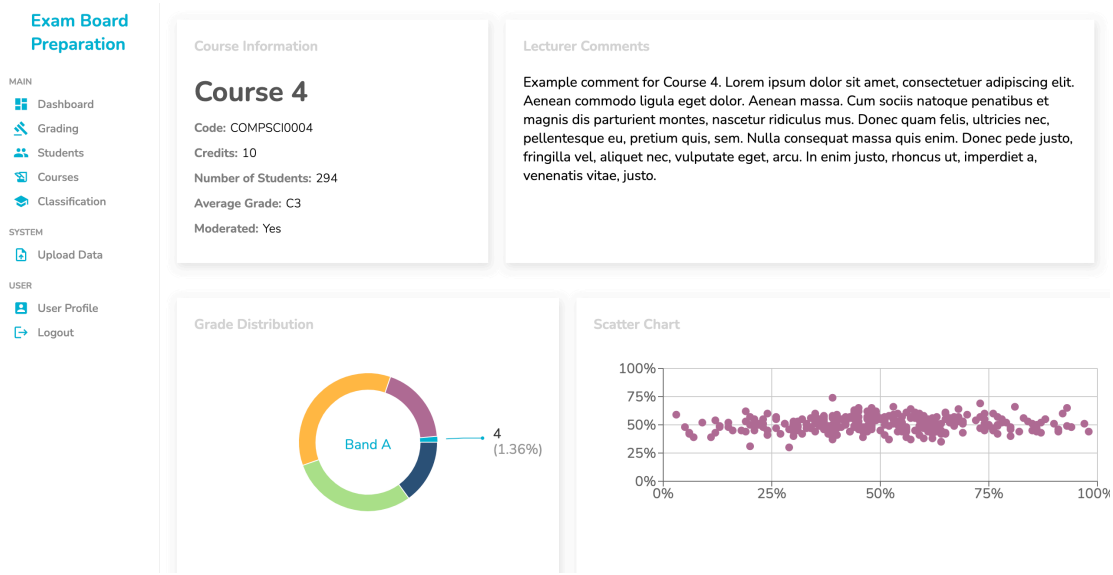

Figure 11: Course Page

If a user identifies a course or specific coursework that has unusually high or unusually low marks, they can moderate it. This augments the grades by the specified factor. Again, the original grade is stored separately from the moderation so that the moderation can be removed, and the original grade is always available.

The master view, or 'Classification' page on the web app, displays a table containing all the students that belong to a cohort, labelled only by matriculation. This is so that the student is anonymous when the information is exported. The table shows the cumulative credits they achieved at each graded band, their weighted grade for taught classes, their project grade and an aggregate grade that combines their taught and project grades. Finally, this view displays the degree classification a student has achieved using the classifications from the DegreeClassisification model.

1. Find the final GPA for the student and set it as *final*
2. Find the number of credits that the student has at each band, *bandCredits*
3. Sort list of classifications, *c*, into descending order by the standard GPA boundary.
4. For each classification in *c*;
    a. If *final* is more than the standard GPA boundary for the classification, return the classification name and terminate.
    b. If *final* is more than the discretionary GPA boundary for the classification;
        i. Calculate the required credits at the specified band for the classification's discretionary rules, and set this to *required*
        ii. If *bandCredits* has more credits at the specified band than required, return the classification name, and terminate.
    c. Go to next classification in *c*

Figure 12: Classification Algorithm

Chapter 4 Testing and Evaluation

4.1 Unit Testing

Model tests were written to ensure the integrity of the system when objects are created or deleted. Additionally, tests were written to check that the APIs successfully retrieved data to pass to the front end.

There were a total of 27 tests, which were run at regular intervals to ensure that new changes had not compromised or broken existing functionality.

```
(ebp) Han@laptop exam_board_preparation % python manage.py test
Found 27 test(s).
Creating test database for alias 'default'...
^[System check identified no issues (0 silenced).
...........................
----------------------------------------------------------------------
Ran 27 tests in 0.122s

OK
Destroying test database for alias 'default'...
(ebp) Han@laptop exam_board_preparation %
```

Figure 13: Unit Tests

Since the web application has less than 10 pages, it was deemed unnecessary to write frontend testing, this could be done manually. However, it would be good practice to write these in the future to allow quick and thorough testing of all aspects of the application, especially if new functionality and pages are added. Additionally, coverage tests could be added to ensure the breadth of testing is sufficient.

4.2 User Testing

To evaluate the success of the web application and receive user experience feedback to build upon the list of future work, a user demonstration and survey was conducted with 9 out of 15 users of the current system.

The users each were given a standard briefing and loosely guided tour of the web application. They had up to 20 minutes to fully explore the web app. Once they were satisfied that they had viewed everything, they were invited to complete a survey.

The list of survey questions was designed to be short, less than 10 minutes to complete, and mainly closed-ended with a bipolar rating to reduce cognitive effort and biases[18]. Only the final two questions of the survey were open-ended, allowing users to express what they like and dislike about the web application without any prompt or restriction. The full list of questions can be found in Appendix C

4.3 Evaluation

All participants of the user evaluation demo and survey were users of the current exam board preparation application. Most users use the current application monthly, while few use it yearly.

Overall Impressions

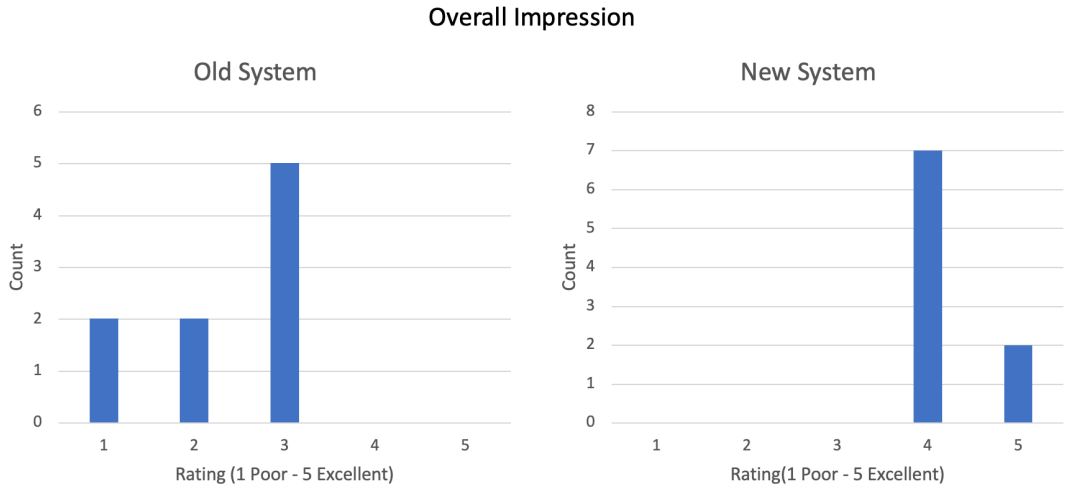

Figure 14: Overall Impression Results

As suspected, the overall impression of the current, old application was low, Figure 14 left. No responses indicating that it is 'Above average' or 'Excellent'. However, the most common response was that the application is 'Average'. This indicates that although the app needs refurbishment, it is still useful for the exam board meetings and provides helpful information.

The overall impression of the new web application, Figure 14 right, was unanimously positive, showing that the web application has been successful in improving the current process. All users consider the new web application to be better than the current application.

Content

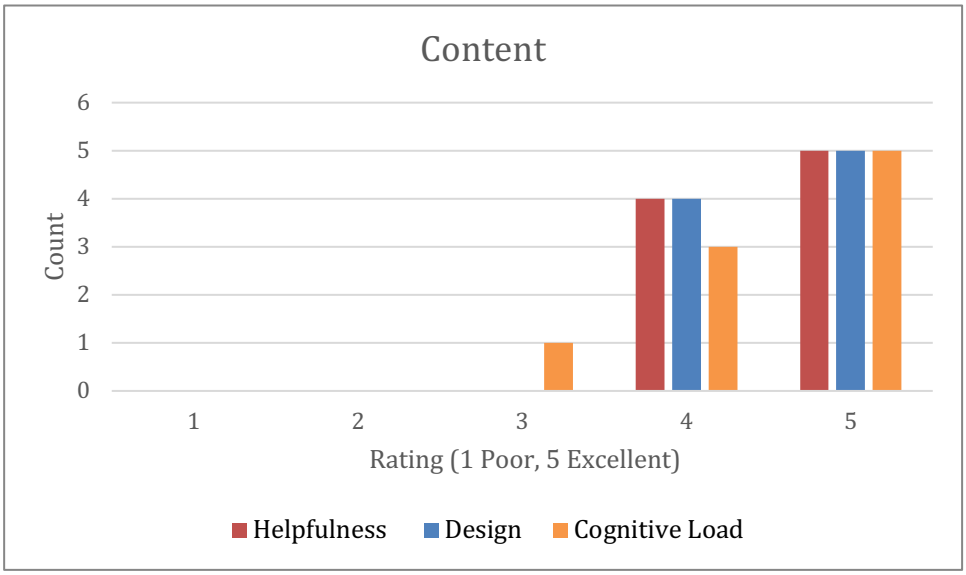

Figure 15: Content Results

Results show that users found the information displayed to be both helpful and visually appealing. When designing pages on the web application, care was taken to show only information that would be used in decision making. In addition, a simple, clean theme was chosen for the design. These also aid in reducing the cognitive load for the user when using the application.

Most user responses in the survey indicated that they want a print to pdf button in addition to the export to CSV functionality. In the survey, 67 % said 'Yes' to this feature, 33 % said 'Maybe' and no-one said that they would not want it.

Usability

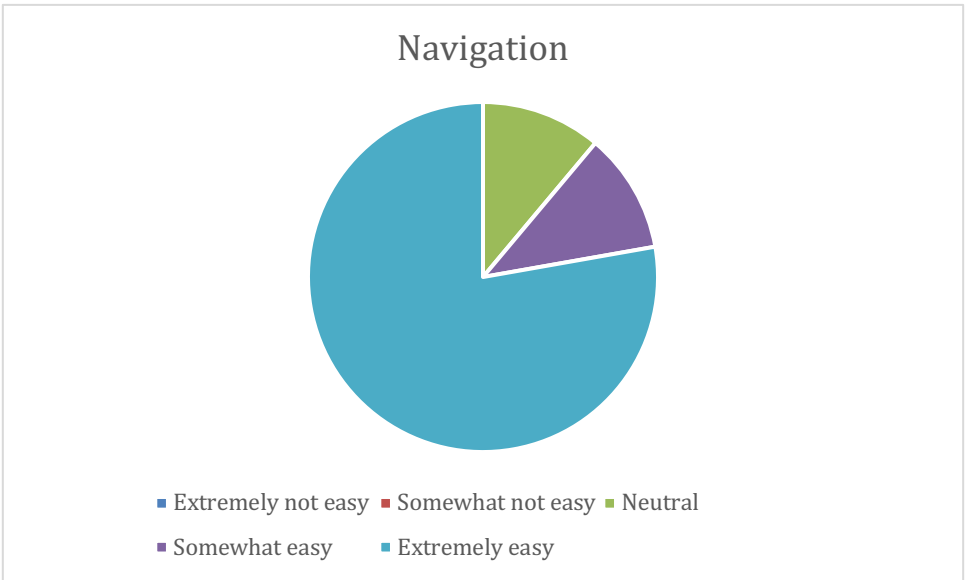

Figure 16: Navigation Results

Survey results show that most users found navigation in the web application to be extremely easy. In all the data tables on the web application, whenever a student or course is shown, there is a button to directly view that individual student or course page. There is also a standard sidebar through all pages on the web application and no page is more than two pages deep.

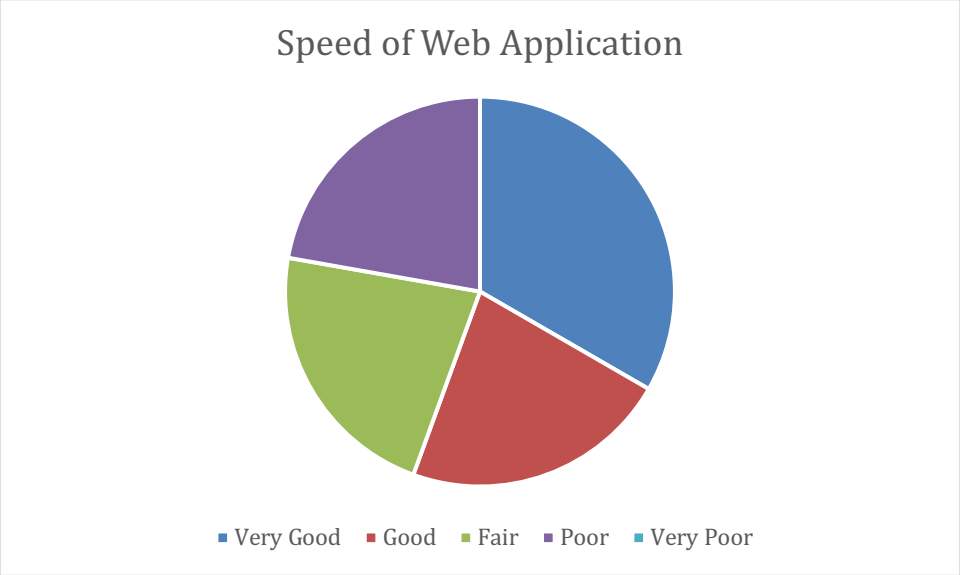

Figure 17: Speed of Application Results

Responses for how the user would rate the speed of the new web application varied a lot. This is likely because, as one user indicated in the open-ended questions, they compared the speed of the web application to the current application, from which there is a significant improvement. And others considered the speed without this comparison, for which the wait is noticeable and needs improvement.

The old web application could load information for 179 students in approximately 3 minutes. The new web application loads 225 students in 9 seconds, Figure 18. This is the longest load time on the new web application.

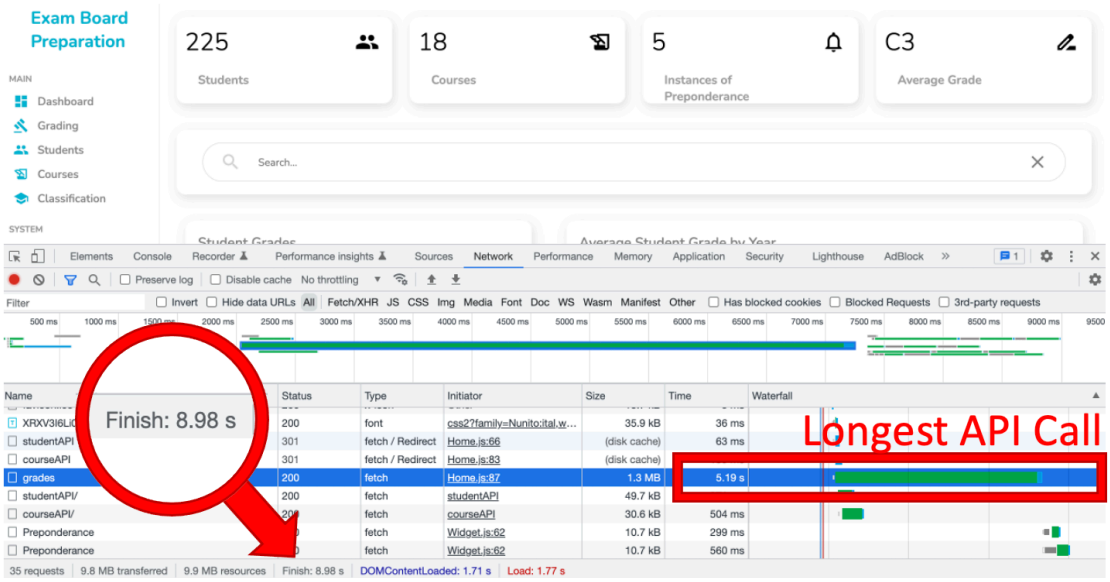

Figure 18: Speed of Longest Page Load

From the network analysis, most of the 9 seconds is from the API call. By using a faster database, or by improving the efficiency of the API call, this time could be reduced.

Open Questions

In the open-ended comments, most users noted that they liked the new simplistic design.

> *"Appealing design"*
>
> *"clean interface"*
>
> *"straightforward presentation"*
>
> *"very appealing"*

Figure 19: User Comments on Design

And others reiterated the ease of finding and displaying useful information without having to copy-and-paste identification numbers across multiple systems.

"easy to navigate"

"intuitive"

"user-friendly view of all crucial information"

"not clogged, easy to use"

Figure 20: User Comments on Information Accessibility

There were many ideas for future work. One user said they would not use the new exam board preparation app. On closer inspection of their open feedback, this is because there is not enough focus on pages for management of non-exit years in the application. This functionality should be added in future work.

Two users also mentioned that it would be useful to compare course grades with the grades of that course in previous years. This could be added with new yearly-view functionality in both individual student and course pages.

There was some contrast in opinions for the level of information that the web application should show. Some users wanted more statistics and more information to be displayed while others wanted less. This balance needs to be managed in future iterations, perhaps with customisable displays for user preference. One user mentioned that they would like all the information to be displayed on the screen without needing any horizontal or vertical scrolling. On the web application, there is currently an option for 'condensed' row density and the ability to hide columns. To remove scrolling as much as possible, column resizing could also be added. Finally, to create more space for tables, hence showing more columns, a sidebar collapse button could be added. This would hide the sidebar from the view when a user chooses.

One user mentioned that a 'help' button could benefit new users. To avoid cluttering the simple page design with surplus text, a small button could be included on the sidebar or at the top of every page where the user can click to receive a page overview.

Users also indicated their desire for the course modification section to be developed further. This would include the ability to test how a moderation would change grade distribution before the moderation submission. As well as a feature to add a note of more complicated moderations that are processed outside of the system.

More customisable grading options were also requested, specifically the ability to view and edit how course grades are rounded before weightings for overall classification calculations.

One survey response suggested the introduction of a feature to pin or star courses that are relevant to that specific user. Currently, in the course view, all courses are shown, but in the final implementation, this might be verbose. Including this

feature would allow users to only view and consider courses that are of interest to their role, reducing unnecessary information and cognitive load.

Finally, several minor changes were suggested that should be easily reflected in the web application. These included showing grades as bands instead of percentages initially, showing the student's name in the 'Mark a Preponderance' section for reassurance, sorting by final grade as default, and showing the student's degree on the classification page.

Chapter 5 Conclusion

This project has successfully laid the foundations for the creation of a web application to replace the Exam Board Preparation page on the old existing system. More work, especially regarding yearly views and non-exit year student management, is required to fully replace the current application.

As shown in the user evaluation surveys, this application has provided a modern and user-friendly interface with easy navigation and information access for users, a significant improvement to the current application.

The middleware for this application is written with Django, a Python-based framework as requested by the future maintenance team. Both Django and React are highly structured and have extensive online documentation allowing for code reuse and easy extendibility if future university systems are to be integrated.

By addition of the features to edit classification boundaries and add comments to individual students, single points of failure in the original process have been removed. Instead of requesting changes via the single system owner, multiple users can update classification rules. Additionally, users can save time for themselves or other users by adding notes to student files, eliminating the need for external communication to receive understanding and clarification. These features, especially editing grade rules could be further expanded to be more customisable and hence further reduce the issues.

5.1 Future Work

From the initial set of requirements, all of the Must Haves and Should Haves are complete. Less than half of the Could Haves and many of the Would Haves were not completed and have been added to future work.

9/9	Must Haves Completed
13/13	Should Haves Completed
7/12	Could Haves Completed
2/7	Would Haves Completed

Figure 21: Overview of Requirement Completion

Notably among these, is the ability to view previous years from the individual student and course pages. For a student, grades could be displayed per completed year and for courses, comparison to previous years could be included in addition to viewing historical data. This functionality was considered when designing the

structure of the web application and to facilitate this, Course objects have a Year attribute. By including year, courses that have the same code, but different years could be grouped. Student grades and classes could also be shown per year by grouping the courses which have the same Year attribute.

For students who cannot progress, it would be good if the system allowed management of decisions made. For example, if the student has to re-sit classes, or if they must re-sit a year, the system should have functionality for users to relay this.

Currently the system focuses on students in exit years. It would be beneficial to add another view under the 'Classification' page that shows and manages student progression. This should have its own rules, similar to DegreeClassification that dictate if a student is able to progress.

Additionally, the Level 2 user noted how the views were centred around exit years. This is likely because no low-level users were able to undertake interviews for requirement capturing at the beginning of the project. Should the project continue, these should be set up to ensure that the progression views contain all the important information that these users would require.

For exam board meetings, it would ultimately be useful to simply have a view in the application that could be used for all review and decision making. Perhaps this could be done by 'locking' pages and 'approving' decisions. However, as an intermediary step, the web application could simply generate a printable format displaying necessary information. Students that should not be considered in the exam boards could also be removed from these views, for example if a re-sit year is already agreed and confirmed.

This system has the potential to catch erroneous marks and identify students who may be eligible for good cause before the rush of exam board decisions. The web application could notify users if a student's grades seem uncharacteristically low in comparison to their other grades or their peers' grades. This would be an indication that follow-up is needed to identify the cause.

From the user evaluations, many users tried to click on the top widgets on the dashboard. This indicates that they should be interactive and clickable. The 'preponderance' widget could be especially useful since there is currently no way on the website to isolate students with preponderance.

Users indicated that they would like a 'Print to PDF' option. This could be integrated with the export views mentioned previously as a stepping-stone for gathering information for exam board meetings.

While the API calls have significantly improved the speed of page loading compared to the old application, there is still room for improvement. When the API calls for this new web application were initially written, they were 43 seconds long for 225 students. By changing the order and reducing nested attributes, these were reduced to 9 seconds. Further organisation and refactoring are likely to speed them up further.

Currently, the ClassHead model has a one-to-one relationship with the User model. This could be improved by extending the User model to include the

ClassHead attributes. This would reduce complexity in the database relationships. This change is shown below in Figure 22, where the light grey sections show the removed elements from the current ER structure.

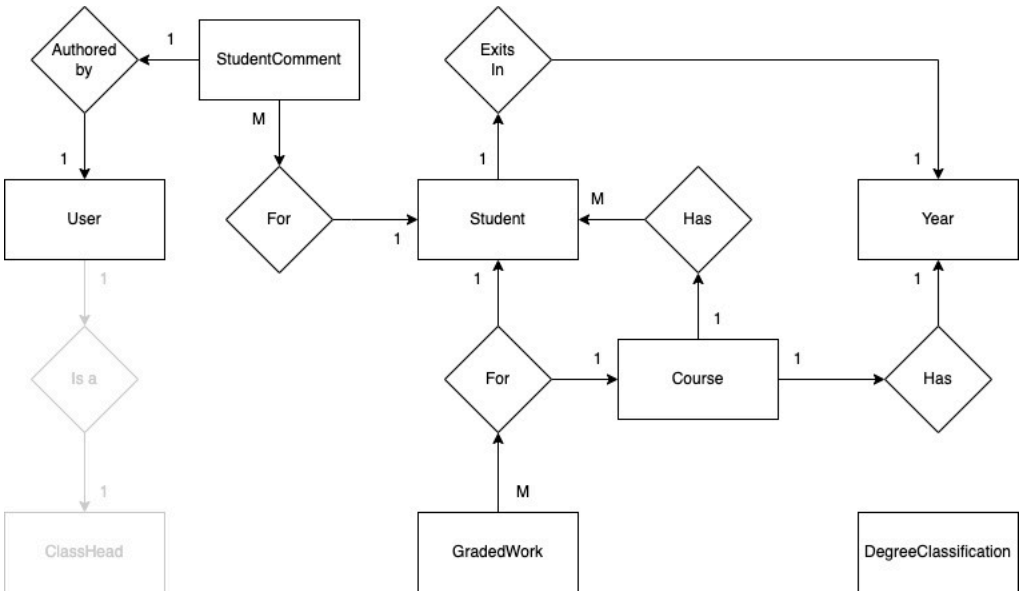

Figure 22: Ideal ER Structure

A rounding error was discovered near the end of the project for how the web application calculates degree classifications. The web application only rounds the final overall grade for the student's degree classification. However, in alignment with the current grading policy, each individual weighted course mark should be rounded before calculating the overall weighted grade. This is perhaps something that could be made more visible for users and editable if future rules change.

Finally, testing is vital for code maintenance. As functionality expands, it is imperative that more tests are written. These should also include coverage reports and front-end testing.

The full list of future work can be found in Appendix D

5.2 Personal Reflection

When nearing the end of the project, I realised that some of the agreed requirements were qualitative instead of quantitative, which made it very hard to know when I had achieved them. For example, one of the requirements is that pages on the web application load "quickly". It is difficult, if not impossible, to know if I have achieved this requirement. As a result, I asked my supervisor if the speed was acceptable to complete this requirement, but it would have been better if I set an unambiguous time, e.g. "All pages load in under 10 seconds".

When I started this project, I had no previous experience in using React. The task was quite daunting, but I allocated time at the beginning of the project to complete a Udemy course on React basics [19]. I am glad that I did this because I think that it provided a solid foundation for understanding components and how to structure the application efficiently. Even with this course, I found it initially very frustrating to enact my ideas in React, but as the project progressed, I started to

notice how much I was learning. When creating some of the final functionality, I mentally noted that I did not have to look up structures or commands and that components that initially took me days to create were created in hours.

I am naturally very organised and independent. Throughout the project, I used a Notion page to plan iterations and track progress. I had set the page up before the project started so that it was very easy for me to use when the project got busier. However, perhaps partly owing to the enjoyment of a difficult problem and partly stubbornness to move on or ask for help, I initially found it difficult to move on quickly if there was something that I could not do. This was something that I practised and got better at as the project progressed. I tried to set strict boundaries for when to move on with a problem by giving myself "one more hour then do something else". Often the solution would come to me a day later when I was not fussing over it.

When there were delays in the project, I was good at shifting tasks to maintain momentum. I think this is down to having a quickly customisable plan on Notion, where I could simply drag cards to different iterations and view upcoming tasks quickly.

Chapter 6 References

[1] "Alma." Accessed: Aug. 27, 2022. [Online]. Available: https://www.getalma.com/

[2] "thinkwave." Accessed: Aug. 27, 2022. [Online]. Available: https://www.thinkwave.com/

[3] "JumpRope." Accessed: Aug. 27, 2022. [Online]. Available: https://www.jumpro.pe/

[4] "Schoology Learning." PowerSchool. Accessed: Aug. 27, 2022. [Online]. Available: https://www.powerschool.com/solutions/unified-classroom/schoology-learning/

[5] "TeacherPlus Web Gradebook." Rediker. Accessed: Aug. 27, 2022. [Online]. Available: https://www.rediker.com/solutions/student-information-system/teacherplus-web-gradebook

[6] "GradeBookWizard." Accessed: Aug. 27, 2022. [Online]. Available: https://secure.gradebookwizard.com/

[7] "MoSCoW Prioritisation," *Agile Business Consortium Limited*. https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation (accessed Aug. 08, 2022).

[8] "BRAND TOOLKIT," *University of Glasgow*. https://www.gla.ac.uk/myglasgow/staff/brandtoolkit/brandelements/colours/ (accessed Aug. 16, 2022).

[9] "Palette Generator." https://coolors.co/generate (accessed Aug. 16, 2022).

[10] J. Sandy, "Choosing between Django, Flask, and FastAPI," *Section.io*, Jan. 04, 2021. https://www.section.io/engineering-education/choosing-between-django-flask-and-fastapi/ (accessed Aug. 16, 2022).

[11] "Alternatives, Inspiration and Comparisons," *FastAPI*. https://fastapi.tiangolo.com/alternatives/ (accessed Aug. 16, 2022).

[12] "Django." Django Software Foundation, Lawrence, Kansas, Apr. 2022. Accessed: Aug. 08, 2022. [Online]. Available: https://www.djangoproject.com/

[13] "React." Meta, Jun. 2022. Accessed: Aug. 08, 2022. [Online]. Available: https://reactjs.org/

[14] "Material UI." Apr. 2022. Accessed: Aug. 08, 2022. [Online]. Available: https://mui.com/

[15] “Recharts.” Feb. 2022. Accessed: Aug. 08, 2022. [Online]. Available: https://recharts.org/en-US/

[16] “Django REST framework.” Dec. 15, 2021. Accessed: Aug. 08, 2022. [Online]. Available: https://www.django-rest-framework.org/

[17] “Simple JWT.” https://django-rest-framework-simplejwt.readthedocs.io/en/latest/ (accessed Aug. 17, 2022).

[18] J. S. Olson and W. A. Kellogg, Eds., *Ways of Knowing in HCI*. New York, NY: Springer New York, 2014. doi: 10.1007/978-1-4939-0378-8.

[19] R. Dhungel, “React JS Frontend Web Development for Beginners - Udemy.” Accessed: Aug. 08, 2022. [Online]. Available: https://www.udemy.com/course/react-tutorial/

Appendix A Entities and Attributes

Full list of entities and attributes.

User
- Email
- First_name
- Last_name

Year
- yearStart
- yearEnd

Student
- name
- matriculationNumber
- degreeTitle
- mastersStudent
- fastRouteStudent
- exitYear (Foreign Key – Year)

Course
- classCode
- className
- students (Many to Many – Student)
- year (Foreign Key – Year)
- isTaught
- lecturerComments

GradedWork
- name
- course (Foreign Key – Course)
- student (Foreign Key – Student)
- weighting
- gradeMark
- moderation
- preponderance

ClassHead
- user (Foreign Key – User)
- level

StudentComment
- student (Foreign Key – Student)
- user (Foreign Key – User)
- date
- subjectLine
- comment

DegreeClassification
- classificationName
- lowerGPAStandard
- lowerGPADiscretionary
- percentageAboveForDiscretionary
- charBandForDiscretionary

Appendix B Interview Questions

Questions used to conduct User Requirement Capturing interviews.

1. What is your role and why would you need a web app?
2. What are your responsibilities that this system should help you to achieve? What information is used to achieve these?
3. What outcome(s) do you want when you go to use this system?

4. What and when do you need to know about a particular student?
5. What and when do you need to know about a particular course?

6. What aspects of the current site are helpful?
7. What aspects of the current site are not helpful?

8. Is there anything that you think would be useful that isn't on the current site?

Appendix C User Evaluation Survey Questions

Questions completed by 9 users to evaluate the web application.

Question	Options and Result
How often do you use the current exam board website?	• Daily • Weekly • Monthly (56%) • Yearly (44%) • Never
What is your overall impression of the current exam board website?	• Excellent • Above average • Average (56%) • Below average (22%) • Poor (22%)
How easy or difficult was it to navigate through the new web app?	• Extremely easy (78%) • Somewhat easy (11%) • Neutral (11%) • Somewhat not easy • Extremely not easy
How helpful or unhelpful was the information that was displayed on the new web app?	• Very helpful (56%) • Somewhat helpful (44%) • Neither helpful nor unhelpful • Somewhat unhelpful • Very unhelpful
Would you use a 'Print to PDF' button when exporting data? (additionally to the CSV export)	• Yes (67%) • Maybe (33%) • No
How did you perceive the cognitive load when using the new web app?	• High (too much information displayed) • Above average • Average (11%) • Below average (33%) • Low (easy to digest) (56%)
How do you rate the speed of the new web app?	• Very good (33%) • Good (22%) • Fair (22%) • Poor (22%) • Very Poor

How was the visual design of the new web app?	• Very good (56%) • Good (44%) • Fair • Poor • Very Poor
Would you use the new web app?	• Yes (89%) • No (11%)
What is your overall impression of the new web app?	• Excellent (22%) • Above average (78%) • Average • Below average • Poor

What do you like about the new web app?
What do you think could be improved/ideas for future work?

Appendix D Future Work

Remaining tasks from initial requirements

- As a User, I want to compare the grades for a course to the grades from the previous year.
- As a User, I want to view the weighted percentages of grades at each band for a student.
- As a User, I want to view a graph to compare student grades on questions in an exam.
- As a User, I want to identify students that are not able to progress to the next year.
- As a User, I want to view the GPA of a student across each of the years that they have completed.
- As a User, I want to use either student ID as digit only or digit and letter.
- As a User, I want to quickly identify students who may be eligible to submit a good cause.
- As a User, I want to quickly identify grades that are likely to be erroneous.
- As a User, I want to hide or censor students who should not be considered for exam boards
- As a User, I want to check that all students have been given a mark or preponderance for each coursework or exam question.

From user evaluations, the following tasks have been added;

- Making the tiles on the dashboard clickable/interactive
- More custom views for non-exit years e.g., Level 2
- Coursework averages on an individual course page
- Cohort statistics on Classification page with comparison to previous years
- 'Help' button for each page
- Speed up API calls
- Pin or Star courses relevant to user
- Resize column functionality
- Further editing/customisable ability for calculating degree classifications
- Customisable displays, show or hide additional statistics
- Print to PDF functionality

Additional work;

- Split individual course and student pages into years
- Make DegreeClassifications year specific
- Write frontend tests
- Collapsible sidebar