Assessment Assistant

A dissertation submitted in partial fulfilment of

The requirement for the degree of

MASTER OF SCIENCE in Software Development

In

The Queen's University of Belfast

By

Paul Joseph Gault

29th April 2019

Paul Gault - 40126005

SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER SCIENCE

CSC7058 – INDIVIDUAL SOFTWARE DEVELOPMENT PROJECT (Part-Time)

A signed and completed cover sheet must accompany the submission of the Individual Software Development dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

Declaration of Academic Integrity

Before signing the declaration below please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.
6. Software and files are submitted on a memory stick.
7. Journal has been submitted.

I declare that I have read both the University and the School of Electronics, Electrical Engineering and Computer Science guidelines on plagiarism - http://www.qub.ac.uk/schools/eeecs/Education/StudentStudyInformation/Plagiarism/ - and that the attached submission is my own original work. No part of it has been submitted for any other assignment and I have acknowledged in my notes and bibliography all written and electronic sources used.
I am aware of the disciplinary consequences of failing to abide and follow the School and Queen's University Regulations on Plagiarism.

Name: (BLOCK CAPITALS) _____

Student Number: _____

Student's signature _____ *Date of submission* _____

Acknowledgements

The completion of this project would not have been possible without the continued support of Dr Richard Gault. His suggestions and ability to act objectively as a stakeholder has time and again proved to be a crucial aspect in driving the progress of this project - having a supervisor who fits in to the target audience really has been a huge benefit.

Secondly, thanks must also be given to the Citi Rates eTrading team in Belfast for support throughout this project. With special thanks given to Gerard Donnelly for allowing me the flexibility to work on my project when it was necessary.

Lastly, a special thanks must be given to the user acceptance testers who provided critical feedback on the acceptability of the project.

Abstract

The core purpose of this project was to improve upon the currently existing e-Assessment tools by facilitating the automatic marking of students' solutions; providing an efficient and easy-to-use way of assessing students mathematically; improving the consistency of marks given to students in a large class and quickening the speed with which students receive feedback from markers. A web-application was built in order to satisfy this that made use of technologies such as Java, Spring-boot, Angular and MySQL. With a requirement completion ratio of 91%, an average user acceptance score of 8.4/10, a developer productivity of 31.4loc/h and 99% code coverage to ensure stability - it has been concluded that this system has adequately achieved its goals.

Table of Contents

List of Figures

List of Tables

List of Equations

Introduction

This report documents the entire execution of the CSC7058 Assessment Assistant project which seeks to improve upon the current e-Assessment facilities that are offered. This section introduces the coming chapters, outlining the key sections of the project discussed within them.

Chapter 1 gives a critical evaluation of the existing online e-Assessment tools - especially for mathematical assessment. The resultant strengths and limitations of the current literature and software tools informs the investigation of a novel Assessment Assistant. In particular, the consideration of the adverse effects on the consistency of marking across large student groups. In addition, this chapter provides information on the target audience for the system.

Chapter 2 develops the proposed solution, at a relatively high-level, from the perspective of the student to satisfy the problem. This chapter provides a justification of the chosen Agile development model and identifies some of the design details that correspond to this model.

Chapter 3 outlines the requirements elicitation process. Documented here are the requirements, in the form of user stories, for the project alongside an explanation of how they were converted retrospectively to functional and non-functional requirements.

Chapter 4 outlines the key design decisions made throughout the process. Starting with a discussion in to the architecture of the system the chapter then moves on to look at the design of the various components and sub-components of the system.

Chapter 5 describes the methodology used in the development of this project as well as describing in detail the key functions the system performs and the algorithms that make those functions possible. The testing strategies including black-box, white-box and user acceptance testing are also discussed in this chapter.

Chapter 6 provides an evaluation and conclusion for the project as a whole. Evaluating the success of the project against the criteria set out in Chapter 3. The languages and technology used to deliver the end product will also be evaluated. Potential opportunities to further this project will also be discussed and outlined here.

NB - Details on how to access or run the application can be found in Appendix P.

Chapter 1: Problem Specification

1.1 - The Original Problem Specification

How does one ensure consistency when marking assessments for one hundred or more students in Higher Education? Many factors contribute to inconsistencies. For example: in medium to large classes a Lecturer will need to eventually bring in an assistant to aid with the marking - how can all tests be marked evenly if they aren't marked by the same person? Even in a class where the tests can be marked by the same person - factors such as: distractions, fatigue, or a change in opinion may lead to inconsistencies in marks within and across different questions. The emergence of e-Assessment tools has aided in overcoming this issue, resulting in tutors feeling "that they mark more consistently, reliably and efficiently" (Campbell, 2005).

In existing e-Assessment tools, Lecturers are very limited in the scope of question types that they are able to ask, which (Cook, 2010) found to be due to the "limited range of interaction types" available. Essentially the only medium the students can use to answer questions is through typing, which means questions that require drawing or complex mathematical scripting can be difficult to include. For this reason, e-Assessment tools are primarily limited to basic text-based questions or multiple-choice questions. Hence this can be restricting for some Lecturers, particularly those assessing topics which require students to provide complex algebraic solutions. This could encourage these lecturers to stick to a paper-based approach when assessing students, and as a result lose out on the benefits that e-Assessment can offer.

Finally, there is also the issue of reviewing students' performance which typically consists of the mean, maximum, minimum scores. When an assessor marks a paper-based assessment they then have to begin calculating performance-based metrics manually, (Campbell, 2005) found that e-Assessment removes this. With a technology-based approach, the production of useful reports can be included in to the application itself making it fast and easy to calculate and distribute feedback in order to help the student in their development. In (Wiggins, 2012) timely feedback is listed as one of the seven keys to effective feedback.

1.2 – The Target Audience

The target audience for this project are students, educators and assessors in Higher Education. Especially Lecturers who are in charge of modules that have a class size of one hundred people or more and may have the desire to delegate some of their marking to their teaching assistants. Another subset that is a particular target are those in a Science, Technology, Engineering or Mathematics (STEM) area. However, the system will not be limited to this staff. The students themselves will benefit from many of the features available on the system such as the generated feedback they can automatically receive when performing practice tests and the facilities in place to ensure fair and consistent marking. The system is not unique to Higher Education and would be relevant for those in post-primary education. In particular, schoolteachers could easily use the system as a way to gather and mark students' homework while issuing feedback and benefitting from the available performance metrics.

1.3 – Existing Solutions

This Section will discuss some of the e-Assessment tools currently used by Lecturers to address the challenges of quick, accurate and consistent feedback. Two main tools will be of focus here: Questionmark Perception (Questionmark, 2019) and Gradescope (Gradescope, 2019).

The Questionmark Perception is a “complete assessment management system that enables you to create questions and organize them into exams, quizzes, tests, or surveys” (Questionmark, 2019). Other features listed include scheduling assessments and the ability to report on and analyse results through reports. Thus streamlining the paper-based assessment process in an online format.

However, one could argue that they are simply the fundamentals and that there is some major room for improvement. One such feature that an e-Assessment tool should have is the ability to assist with the marking of answers. Currently, Perception offers marking assistance for questions that have limited scope of entry, for example: true-false questions, multiple-choice and yes-no questions. This is a good start but the ability to auto-mark or at least assist with questions that allow open-response is a feature that would significantly reduce the time taken to mark and provide feedback. As outlined in the previous chapter (Wiggins, 2012) lists timely feedback to be one of the seven keys to effective feedback. While Questionmark offers a large range of question types, it is has no functionality to allow Lecturers to ask questions that require even basic algebra, for example: Euler’s formula (Euler, 1748) which is outlined below. While there is a way for lecturers to add this to the question content, the index on the left-hand side would be difficult for students to insert without compromising readability. Due to the fact that it only allows students to answer via basic text input.

$$e^{ix} = \ cosx + isinx$$

Equation 1 - Euler's formula

Gradescope allows the user to grade paper-based exams, quizzes, and homework. In addition, Gradescope enables the user to automatically, or manually grade programming assignments (Gradescope, 2019). The idea with this system is that Lecturers can issue paper assignments to their students. Upon completion of their assignments, the Lecturer scans the students’ submissions and enters the files in to the Gradescope system where the Lecturers can mark them and have any marks they give or comments added saved for reuse on any submissions they are yet to mark.

This approach to assisted marking means the Lecturer, in theory, can ask any question type they want without need for any special configuration. However, one could argue that this paper-based approach is also the biggest weakness of the Gradescope system. The administration time required by Lecturers to correctly arrange and scan in the submissions is unnecessary and is time taken away from teaching their students or grading their submissions to return feedback as quickly as possible.

An online version of Gradescope is currently under development (Gradescope, 2019), which would in theory eliminate any weaknesses of Gradescope that arise due to their paper-based approach. Upon review of limited information provided of their online system, which is still in test, one can see that there are still some areas of concern with their approach. Firstly, questions are created within the system on a per assessment basis - the issue with this is that questions cannot be reused in other assessments and thus answers cannot be banked up. In theory, the marks and feedback given to an

answer can be reissued automatically on identical answers, hence making questions reusable can ensure consistent and fair marking on those questions across many assessments spanning any period of time. Secondly, while the Gradescope online system would allow tutors to add mathematical content to the questions they create, it does not seem as though there is functionality to allow students to provide mathematical content. This limits the students from being able to demonstrate an ability to communicate in mathematical questions which was characterised as one of the limitations of using multiple-choice questions in mathematical e-Assessment in (Azevedo, 2015). Lastly, similar to Questionmark, the Gradescope online system provides automatic marking for limited scope of response questions but does not provide it for open-response questions, providing this would lead to more timely feedback.

In summary, for the reasons outlined a good e-Assessment tool should**;** streamline the paper-based assessment process in an online format, assist with the marking of answers even open response questions, allow user to ask questions with complex algebra, and allow for reuse of questions. Currently, there is no such system which can provide all of these benefits. Chapter 2 will outline the proposed solution that seeks to remedy this as well as justifying the chosen agile methodology to be used in development.

Chapter 2: Proposed solution and justification of the development model

2.1 - The Proposed Solution

The overall purpose of this project is to create an application capable of housing assessment, feedback and marking all under one roof, while aiding the Lecturer throughout the assessment process to avail of automated marking. If successful, a solution would; streamline the paper-based assessment process in an online format, assist with the marking of answers even open response questions, allow user to ask questions with complex algebra, and allow for reuse of questions. As outlined in the previous chapter neither Questionmark Perception (Questionmark, 2019) nor Gradescope (Gradescope, 2019) provide all of these features.

The proposed solution was to use an older version (v1) of Angular (Angular, 2010-2019) that did not use the Angular Command Line Interface (CLI). This system was intended to be developed using Thymeleaf (Thymeleaf, 2018). This is a template engine which serves out data generated by templates in the backend code as static files or parts of a static file. Some AngularJS would then be used to try and improve the responsiveness of the code.

The reasons for this system being deemed unsuitable are as follows:

- *Coupling of front and backend systems*: Storing both components within the same code base means neither can be reused without the other, due to the fact that the backend code must serve up the frontend code by generating templates from strings. This means that code will be more difficult to maintain as developer must know the entire system to change just one component (Hokstad, 2008).

- *Multi-page application*: This is less responsive and slower than a single-page dynamic application (RubyGarage, 2018).
- *Slowed development*: Since the static files are stored as part of the spring-boot application, making even the smallest of changes to front-end code, such as an update to a CSS style, required the entire spring-boot application to restart. Which can take several minutes depending on specifications of the machine being developed on.
- *Developed using outdated technology*: The only version of Angular which is developed without the Angular CLI (v1) is several versions out of date and thus the system will not be able to avail of features of the framework that have since been added such as; production builds which packages files for deployment and development mode which allows the developer to easily add live updates.

For these reasons a solution which includes the following would be desirable:

- *Decoupling of front and backend components*: Doing this means changes can be made to one without need to rebuild and re-test the other, it also means either component can be taken and reused elsewhere. Creating backend and frontend components that are separate entities can be done using Spring-Boot (Pivotal, 2019) and Angular.
- *Ease of development*: Use of the Angular CLI greatly streamlines the development and deployment phases, this will be discussed in more detail later in Chapter 5.
- *Access to spring framework*: Provides access to spring security, ability to create a REST API and other useful external jars.
- *Creation of a single page dynamic web-application*: Significantly improves speed and responsiveness as outlined in (RubyGarage, 2018) and can be achieved through use of Angular.

Instead, the proposed solution was a web-application with a backend component to perform the system's logic and communicate to the database, alongside a fronted component to serve content to the end user in an elegant and responsive user interface.

The backend component will be written entirely in Java and make use of the Spring-Boot framework for configuration purposes. Alongside this, a MySQL (Oracle, 2019) Database will be built in order to persist some of the data required for the running of the application alongside content generated by the users. Lastly, the frontend component will be developed with Angular using the Angular CLI to generate the required components of the Angular application. Each Angular component contains a HTML file, a TypeScript file and a CSS file available for editing.

2.2 – Justification of the Chosen Development Model

The project used an agile development model. Agile software development is an umbrella term for a set of practices based on the principles expressed in the Agile Manifesto (Agile Alliance, 2019).

It is therefore that the development ethos aligns with the principles outlined in the Agile Manifesto (Kent Beck et al, 2001). A select number of these principles will now be examined.

- "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software" - This was upheld by planning each set of features for the sprint at meetings in line with the stakeholder's needs and demonstrating them once complete.
- "Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale" - In this project delivery was on the preferred shorter side, demoing the working software every two weeks.
- "The most efficient and effective method of conveying information to and within a development team is face-to-face conversation" - Almost all stakeholder communication pertaining to features of the software was carried out in the fortnightly face-to-face meetings.
- "The best architectures, requirements, and designs emerge from self-organizing teams" - All deliverables on the work slate for each sprint were implemented at the developers own pace throughout the duration of the sprint.

The developer considered the SCRUM (Scrum.org Team, 2019) method for agile development as a potential candidate for the chosen development model. However, given the fact that there is only one developer for this project it was decided that certain key principals of SCRUM such as daily scrum meetings and social motivation by volunteering for work in front of peers could not be upheld.

Another such framework that follows agile practices is Iterative and Incremental Development (IID) (Larman, June 2003). The IID approach to software development is modelled around a gradual increase in feature additions. In incremental development, different parts of the system are developed at various times and are integrated based on their completion (Technopedia, 2019). This project will only have one developer, IID facilitates this as it encourages the system to be built up feature by feature. This also means that the code produce will be lean as it is intended to satisfy one feature at a time. This also leads to code that is easier to understand and manage. IID breaks the creation of the system down in to a series of iterations or sprints. In the timeline of this project it would be appropriate to work in 9 sprints, each lasting roughly 2 weeks. Fortnightly meetings and updates with the stakeholder, Dr Richard Gault, made it practical to align the sprints to these meetings. The meetings provided an opportunity to review the previous sprint and demonstrate progress to Dr Gault and for him to provide feedback on what was delivered. At these meetings a new set of deliverables were outlined for the next fortnight. Each sprint consisted of planning, developing and testing the software required to supply the agreed upon deliverables. Testing the code little and often ensured that defects were detected and remedied as early as possible. The Gantt chart outlining this timeline can be seen in Figure 1.

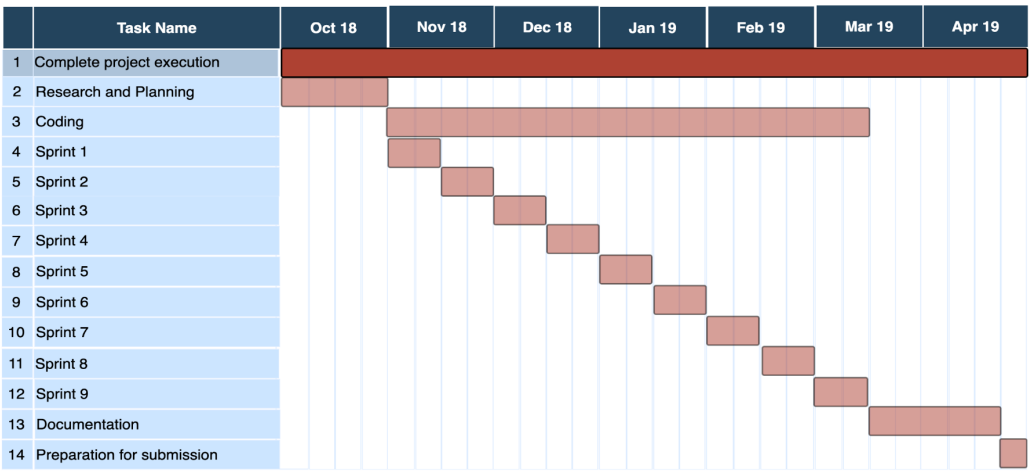


Figure 1 - The Project Gantt Chart detailing the area of focus at any given time throughout the project.

Throughout development a TeamCity continuous integration server was used to monitor the health of the builds and generate artefacts needed for deployment. This ensured that at any given time throughout the development lifecycle, if the project timeline was moved forward the last successful build of the application is ready to be deployed. A screenshot of the build artefacts on the TeamCity GUI can be seen in Figure 2.

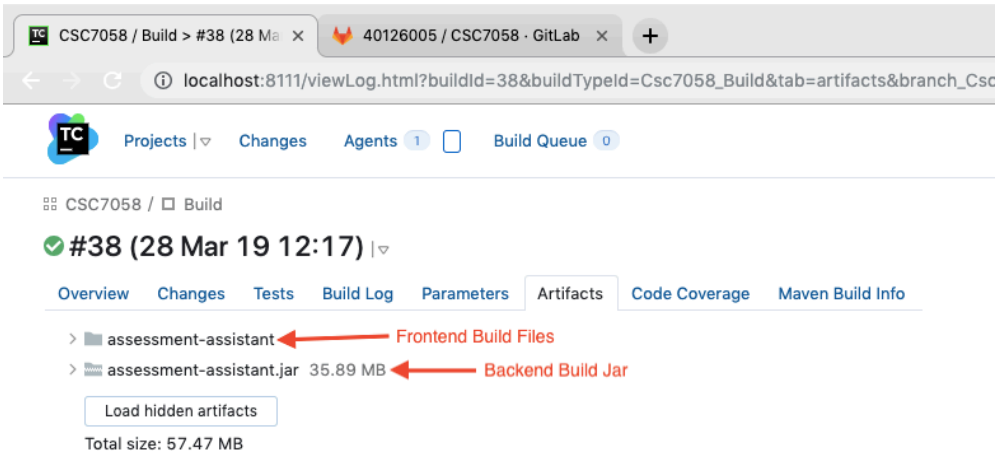


Figure 2 – The Team City Build Artefacts containing the files needed for deployment of front and backend systems.

In summary, the proposed solution was a web-application with a backend component to perform the system's logic and communicate to the Database, alongside a fronted component to serve content to the end user in an elegant and responsive user interface. The approach chosen for the development of this project was Agile specifically Iterative and Incremental Development. This chapter has examined the proposed solution and how the developer aligned the project with the principles of agile. The next chapter will look at the requirements for the system, and how the requirements were elicited in line with the development model.

Chapter 3: Requirements analysis and specification

3.1 - Requirements Elicitation Process

In requirements engineering, requirements elicitation is the process of extracting the requirements of a system from stakeholders (Alfeche, 1997). As discussed in the previous chapter, this project was developed using an agile methodology in which working software is valued over comprehensive documentation (Agile Alliance, 2019).

For this reason, all requirements were elicited as user stories. The user stories were generated in the early stages of the project and were discussed and agreed upon by the developer and the stakeholder, Dr Gault.

For the purpose of this report and given the agile methodology used, the functional, non-functional requirements and error conditions for the system have been created retrospectively from the user stories and can later be used to test the success of the project. These requirements will now be outlined in the coming Sections. The requirements have been prioritised using the Numerical Assignment Technique which prioritises requirements as mandatory, desirable or inessential (Brackett, 1990) based on discussions had throughout the project.

3.2 - User Stories

It was important that the user stories were comprehensive for each type of user in the system - hence they were grouped together to ensure that no areas were missed.

Throughout this section of the chapter the user stories will be outlined and discussed.

Table 1 documents the user stories that are applicable to all types of user. Generally, the user stories focus around security measures for a user's account and their information. While these user stories are not entirely critical to the streamlining of the assessment process and auto marking functions, which is the principal focus of this project, they are still absolutely vital for the correct functioning of the system as a whole. In a system where Lecturers can create examinations for their students it is naturally vital that this information be kept private by securing users' accounts.

User	
1	As a user, I want to be able to change the password for my account, so that I can keep my account secure.
2	As a user, I want my information stored on the system to be kept secure, to allow me peace of mind.
3	As a user, I want to be able to log in to the system, so that I can view the relevant options within the system that are associated with my user-type.
4	As a user, I want to be able to log out of the system, so that I can protect my account while not using my device.

Table 1 - The user stories that are generic to all user type.

The user story outlined in Table 2 is one that applies to all users that are not admins, it would be desirable if these types of users would have the ability to contact admins in the event that issues arise.

Student, Lecturer and Teaching Assistant	
5	As a student/Lecturer/teaching assistant, I want to be able to contact the admins, so that I can raise requests or report issues about the system.

Table 2 - The user stories that are specific to non-admin users.

Moving on to user stories that are more specific to a user type. Table 3 contains the user stories specific to students. The principal areas of focus with the student user stories are: accessing their modules, taking tests, and receiving feedback on these tests.

Student	
6.	As a student, I want to be able to communicate with my Lecturer, so that I can ask questions or raise concerns.
7.	As a student, I want to be able to access all the modules I am associated with, so that I can perform necessary actions on them.
8.	As a student, I want to be able to take tests set by my Lecturer, so that I may be able to be assessed.
9.	As a student, I want to be able to take mock tests set by my Lecturer, so that I may be able to practice for upcoming tests.
10.	As a student, I want to be able to answer these questions on the system, so that they may contribute to my assessment score.
11.	As a student, I want to be able to receive automatic feedback on practice tests on the system, so that I do not have to wait for a response from my Lecturer.
12.	As a student, I want feedback on my examinations and practice examinations is kept private, so that other users cannot access my personal data.
13.	As a student, I wish to see statistics about my performance during a test (if this facility has been enabled by the Lecturer during the test construction), so that I may set my goals for future improvement accordingly.

Table 3 - The user stories that are specific to the student users.

In order to reduce students' waiting time for feedback, one feature that could be offered would be the inclusion of practice tests, which have the ability to provide the students with instant feedback through auto-marking. This would significantly decrease the Lecturers workload and allow them more time to focus on preparing lessons and other tasks that aid in their students' learning, in other words "less work for the teacher, more gain for the student" as stated in (Haswell, 1983). Instant feedback would allow the student to learn at a greater pace by eliminating this waiting time.

The user stories in Table 4 focus on module creation and access for the Lecturer user type. These user stories are the building blocks for all the important functionality within the system. Without modules there could be no tests and without tests there could be no auto marking or feedback. It is for this reason that these user stories are vital to the correct functioning of the system.

Lecturer	
14.	As a Lecturer, I want to be able to communicate with my students, so that I may answer any questions they have.
15.	As a Lecturer, I want to be able to communicate with my teaching assistants, to plan to assess with them.
16.	As a Lecturer, I want to be able to add new modules, to use the system to assist with assessing my students.
17.	As a Lecturer, I want to be able to associate students with my modules, so that they can begin to access any material I upload to the module area.
18.	As a Lecturer, I want to be able to associate teaching assistants to my module, so that they can begin to assist me with assessing the students.
19.	As a Lecturer, I want to be able to access all the modules I am lecturing, so that I can perform necessary actions on them.
20.	As a Lecturer, I want to be able to specify the permissions that teaching assistants have within my modules such as; test generating privileges or just marking privileges, so that I can keep control over my modules.

Table 4 - The user stories that are specific to the Lecturer user type that are centred on modules.

Table 5 highlights that the user stories for the Lecturer user type, are focused on test creation. User story 24 describes the various types of question that would desirably be included within the system. It can be seen that there is a rather large number of these which in the given time-frame for this project would be difficult to implement them all.

21.	As a Lecturer, I want to be able to create tests on the system, so that I can use it to assess my students.
22.	As a Lecturer, I want to be able to practice tests on the system, so that I can use it to further my students learning.
23.	As a Lecturer, I want to be able to specify a time-frame that any tests I provide will be available for, to ensure that students can only complete exams while under supervision etc.
24.	As a Lecturer, I want to be able to ask; text input-based, multiple choice, truth-table, flowchart-based, predicate-based, and questions based on solving equations of many different types such as; integrals (definite and indefinite), differentiation, differential equations, quadratic equations, so that I can have a diverse the range of question-types I can ask.

Table 5 - The user stories that are specific to the Lecturer user type that centres on testing.

Throughout the course of the project it was also requested by the stakeholder Dr Gault that insert-the-word type questions would be included within the system, due to the fact that they are very useful to have in terms of assessment. This was added to the functional requirements for the system which will be discussed later in Section 3.2.

The user stories in Table 6 are those concerned with the marking functionality for Lecturer user types within the system. Given that auto marking is the key focus of this project it is important to take a closer look at some of these stories.

25. z	As a Lecturer, I want to be able to choose whether the system automatically, semi-automatically or manually facilitates the marking of individual questions in the exam, so that I have full control over what I allow the system to do for me.
26.	As a Lecturer, I would like to give x marks to a student and have every other identical solution also receive x marks, so as to keep consistency in marking among all students.
27.	As a Lecturer, I want to be able to override the systems information that may be used to assess my students' answers, so that I can make the system more relevant to the field of study that I am teaching in.
28.	As a Lecturer, I want to ensure that another Lecturers rule change doesn't affect the rules used for my examinations, so that I can ensure the integrity of the marking done by the system.
29.	As a Lecturer, I want to easily be able to review marks allocated by the system while adding and subtracting marks to the total where appropriate, so that I can personally ensure the accuracy of the results.

Table 6 - The user stories that are specific to the Lecturer user type that are centred on marking.

User story 25 describes how the system should handle auto-marking on behalf of the Lecturer. It is important that the Lecturer has the ability to review any auto marking performed by the system manually if they so wish - so that they can ensure the correctness of the auto-marking performed. The Lecturer's ability to manually modify the marks is described in user story 29.

It was mentioned in Chapter 1 that a particular area of interest for this project is streamlining the marking process for medium to large class sizes. User story 26, if implemented correctly, will ensure that marking is consistent across medium to large class sizes and in fact can be used to ensure that marking is consistent across multiple year groups. Consistent marking is of the utmost importance to students, as the marks that a student receives can have a direct effect on the opportunities they have

in life (Anne Pinot de Moira et al, 2002). In a class with a larger number of students it can be said that the likelihood of students having the same answer is increased due to the fact that there are only a finite number of possible ways, within reason, that a student could answer a question. As such it can be deduced that the larger a class size is the more likely this system will have benefit.

The user stories in Table 7 focuses on the feedback aspect of the system for lecturers. In order to ensure the timely delivery of feedback to students it would be desirable if the system was able to assign some feedback to students' answers when certain elements appear within them.

30.	As a Lecturer, I want to be able to send auto-generated feedback from the official tests to my students, so that I can test, mark and issue feedback all from the same place.
31.	As a Lecturer, I want to be able to see auto-generated reports including metrics of my students' individual performance, so that I can monitor which students need assistance.
32.	As a Lecturer, I want to be able to see auto-generated reports including metrics of my students' performance as a class, so that I can assess whether certain areas need.

Table 7 - The user stories that are specific to the Lecturer user type that are centred on feedback.

It was desirable that the Lecturer be able to see metrics that report the performance of both individual students and the class as a whole, doing so would allow the Lecturer to assess the areas that they need to revise with students or indeed the students who are in need of increased assistance.

Moving on from the Lecturer user type, Table 8 outlines the user stories for the teaching assistant user type. The range of features available to this user type really isn't overly extensive, their main role within the system is to simply assist the Lecturer with the testing and marking process hence their user stories reflect this.

Teaching Assistant	
33.	As a teaching assistant, I want to be able to communicate with the Lecturer, so that I can plan to assess with them.
34.	As a teaching assistant, I want to be able to access all the modules I am assisting with, so that I can perform necessary actions on them.
35.	As a teaching assistant, I want to be able to perform the actions that the Lecturer has given me the privileges to do, so that I can correctly assist with the assessment of their students.

Table 8 - The user stories that are specific to teaching assistant type users.

Finally, Table 9 address the user stories that are required for an admin type user within the system. These user stories focus on the admins ability to approve the addition of new modules and respond to queries from other users. The main purpose of modules requiring admin approval is to ensure that no inappropriate content may be added that students may be exposed to.

Admin	
36.	As an admin, I want to be able to respond to messages from other users, so that I can take necessary action on them.
37.	As an admin, I want to be able to approve the addition of new modules, so that I can ensure suitability of added modules.

Table 9 - The user stories that are specific to admin type users.

3.2 - Functional Requirements

The functional requirements can be seen in Appendix A and have been generated from the user stories. Application of the Numerical Assessment Technique can be seen in the right-hand column of the table. Naturally the most important requirements in the system are the ones that focus on streamlining and automating marking within the assessment process. However, that is not to say that these are the only mandatory requirements for the system. In order for the system to function correctly various other functions such as logging in and out must be included. In the case of user story 24, it has been divided out in to a number of requirements to ensure that the requirements for the system remain atomic.

3.3- Non-functional Requirements

Given that user stories mainly focus on allowing users to perform certain functions it has not been possible to generate non-functional requirement retrospectively in the same way the functional requirements have been. However, given the proposed solution was a web-application it was possible to deduce some sensible non-functional requirements from this, they are outlined in Appendix B and are mainly focused on: the user's experience with the application and the applications accessibility.

3.4- Error-conditions

The error conditions that the system must handle are outlined in Appendix C. Successful handling of these conditions will ensure that the system always functions optimally even in the event of erroneous user input.

Throughout this chapter the requirements for the system have been defined and can be found in Appendices A-C. The main points of focus for the system are streamlining the process of testing, marking and providing feedback to students. The next chapter will discuss the design of the system that could potentially satisfy these requirements. Starting by taking a look at the architecture of the system in Section 4.1 the chapter will outline the various components of the system, for example: the spring-boot backend component or the angular frontend component, and identify the sub-components of these.

Chapter 4: Design

This chapter will discuss the design of the proposed system. Starting at a high-level and looking at the overall architecture of the system in Section 4.1 the various components of the system will be identified. Then through examination of each component in the following sections their sub-components will be identified and discussed in order to examine how they interact with each other and how they facilitate question creation, testing, marking and feedback.

4.1 - The Software System Design

The layered architecture for the designed system can be seen in Figure 3. Starting from the right-hand side of the diagram the Angular logo represents the frontend component of the system. The user interacts with this frontend component which sends to and receives data from the backend Spring-Boot component, which is represented in the centre of the diagram, through methods that the backend component exposes via the REpresentational State Transfer Application Programming Interface (REST API) (Spring, 2019). The backend component in turn performs the logic for the system.

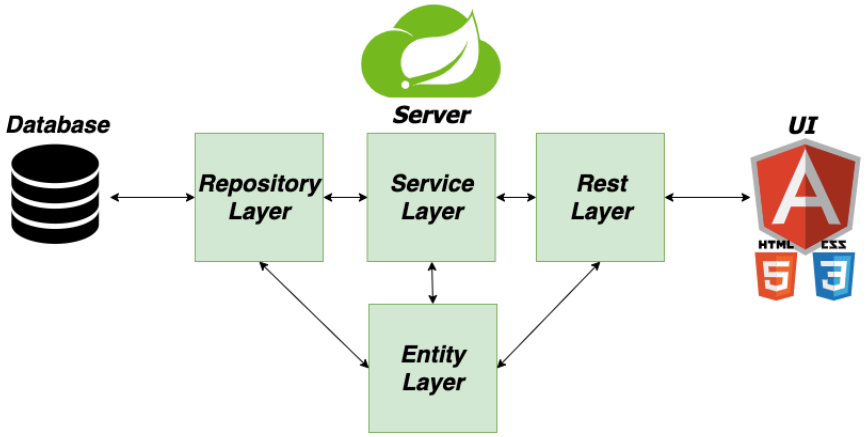


Figure 3 - The System Architecture comprising of the user interface, server and Database components.

All calls made to the REST API must pass through the REST Layer, from here they are passed to the Service Layer which is where all system logic is performed. Some examples of functions that are performed in the Service Layer include creating, submitting and marking tests.

Once the Service Layer needs to access or store persistent data it does so by passing it to the Repository Layer. Each table within the Database has a corresponding Repository class which provides Select, Insert, Update and Delete functions that can be used to perform actions on that table. Calls to the Database are made via Java Database Connectivity (JDBC) (Oracle, 2019).

As well as a corresponding Repository class, every table in the Database has a counterpart Entity class. The entity class makes it possible for attributes of a record in the Database table to be mapped to attributes within the class and in turn allows the system to create java objects that mirror Database records, for manipulation throughout the system.

This layered architecture was designed with the purpose ensuring that all components are decoupled from one another. Meaning that each can undergo minor changes without requiring the same for the other layers. The next section will outline the development of the Database in more detail.

4.2 - The Database

A MySQL Database was chosen for persisting data within this system. The Entity Relationship diagram for this Database can be seen below in Figure 4. The more detailed Relational Model could not be included in the body of the report however it can be found in Appendix D.

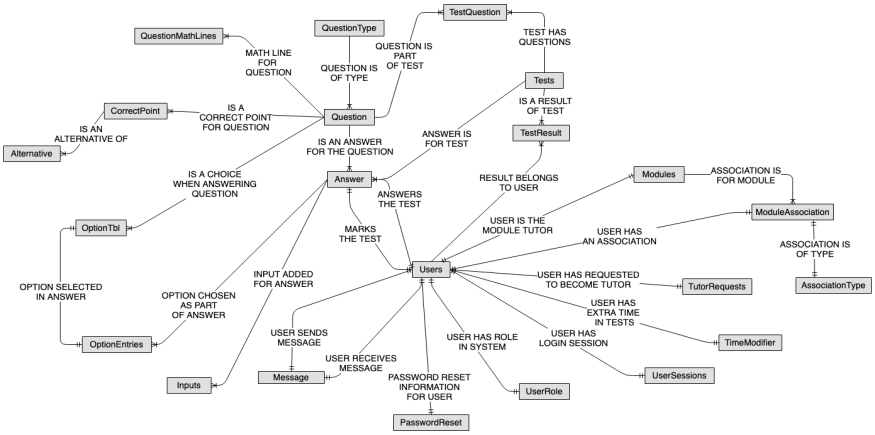


Figure 4 - The Entity Relationship Diagram for the Assessment Assistant system.

The Database has been designed to satisfy 1st, 2nd, and 3rd normal forms by; ensuring that each Database table has a primary key to allow rows to be uniquely identified, ensuring that there are no partial dependencies on these primary keys, and ensuring that no non-key fields are facts about other non-key fields (Anderson, 2017).

Due to the size of the Database this section will focus on the Database design of the more critical aspects for the system.

4.2.1 - Questions

In order to persist questions, a Question table seen in Figure 5 has been added to the Database with a many-to-one relationship with the QuestionType table. As such each question has a type which dictates which other tables, that are dependent on the question, will be populated. All questions may include any number of math lines and thus any that have been entered will be stored in the QuestionMathLines table, which has a many-to-one relationship with the question table, so that they can in future be called upon for display on the frontend.

Question	
FK	questionType int(11) NOT NULL
PK	questionID int(11) NOT NULL
	questionContent text NOT NULL
	questionFigure MEDIUMBLOB
	maxScore int(11) NOT NULL
	minScore int(11) NOT NULL
FK	creatorID int(11) NOT NULL
	allThatApply tinyint(1)

QuestionType	
PK	questionTypeID int(11) NOT NULL
	questionType varchar(255) NOT NULL

QuestionMathLines	
PK	questionMathLineID int(11) NOT NULL
FK	questionID int(11) NOT NULL
	content text NOT NULL
	indexedAt int(11) NOT NULL

OptionTbl	
PK	optionID int(11) NOT NULL
FK	questionID int(11) NOT NULL
	optionContent text NOT NULL
	worthMarks int(11) NOT NULL
	feedback text NOT NULL

CorrectPoint	
FK	questionID int(11) NOT NULL
PK	correctPointID int(11) NOT NULL
	phrase text NOT NULL
	marksWorth double NOT NULL
	feedback text NOT NULL
	indexedAt int(11)
	math tinyint(1) NOT NULL

Alternative	
PK	alternativeID int(11) NOT NULL
FK	correctPointID int(11) NOT NULL
	alternativePhrase text NOT NULL
	math tinyint(1) NOT NULL

Figure 5 – The tables relating to questions in the Assessment Assistant schema.

Options available within a multiple-choice question are persisted within OptionTbl, seen bottom centre of Figure 5, while all other question types have their marking data persisted in the CorrectPoint and Alternative tables shown on the right-hand side of Figure 5. Storing this data within these tables means that the marking information can be readily available for the auto-marking functions.

Questions are linked to tests via the associative table TestQuestion, here the primary key for a question is stored alongside the primary key for a test record from the Test table. This approach ensures that questions are not dependent on tests and can be used in many tests. This allows tutors to reuse questions across tests and as such means that auto marking can be performed against marked answers from previous years which ensures consistency in marking across multiple years.

4.2.2 – Answers

Answers will be persisted via the Answer table which can be seen on the left-hand side of Figure 6. When a student answers a question a record in the answer table is created for this. For text-based questions, the answer that the user supplies are stored in the content attribute of this record, for the auto-marking of these question types - this content attribute will be parsed. The user ID for the student who owns the answer is added to the answererID attribute and the Lecturer's user ID is automatically added to the markerID attribute until the Lecturer reallocates the answer to one of their teaching assistants if they so choose.

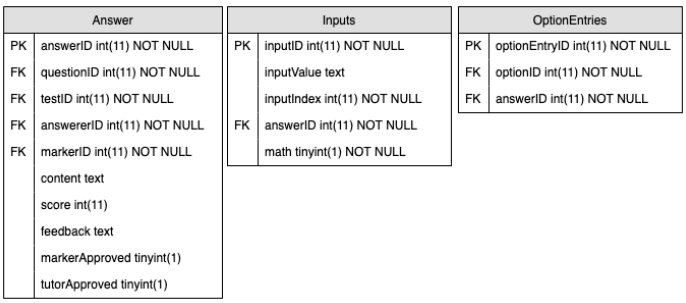


Figure 6 – The tables relating to answers within the Assessment Assistant schema.

In the case of an insert-the-word or math type question the user's response is stored as a series of records in the inputs table, each input belongs to a record in the answer table. For insert-the-word there is a set number of allowed inputs in line with the number of missing words specified at the creation of the question. For Math questions the number of inputs is at the discretion of the person answering the question. The user can add any number of lines of mathematical working or text. All inputs are indexed based on the order they appear in the users answer which is used when the inputs are parsed for the auto-marking of this question type to ensure that insert-the-word entries are correct for the order in which the missing words appear in the question. Options that are selected in multiple-choice questions are stored in the OptionEntries table.

When auto marking or manual marking is completed the score and feedback columns for the answer are populated with the necessary values. Once the time comes for feedback to be given to the students the data contained within these attributes is made available for the students to view.

The SQL scripts for the Database can be found at:
https://gitlab2.eeecs.qub.ac.uk/40126005/CSC7058/blob/master/backendAA/src/main/resources/schema.sql

4.3- The Repository Layer

The Repository layer consists of a series of classes, denoted by the @Repository annotation (Rod Johnson, 2006), that encapsulate Create, Read, Update and Delete (CRUD) operations from the application logic. They are also commonly referred to as Data Access Objects (DAO) defined in (Oracle, 2001), due to the fact that they allow the system to access the data that has been persisted in the Database.

A snippet of the class diagram for this layer can be seen in Figure 7. Each class within the layer corresponds to a table within the Database in order to keep the classes atomic. It is for this reason that none of the classes within the layer require any interaction with each other, no lateral movement between each class means that the flow of the code is less complicated to follow and debug for future developers of the system.

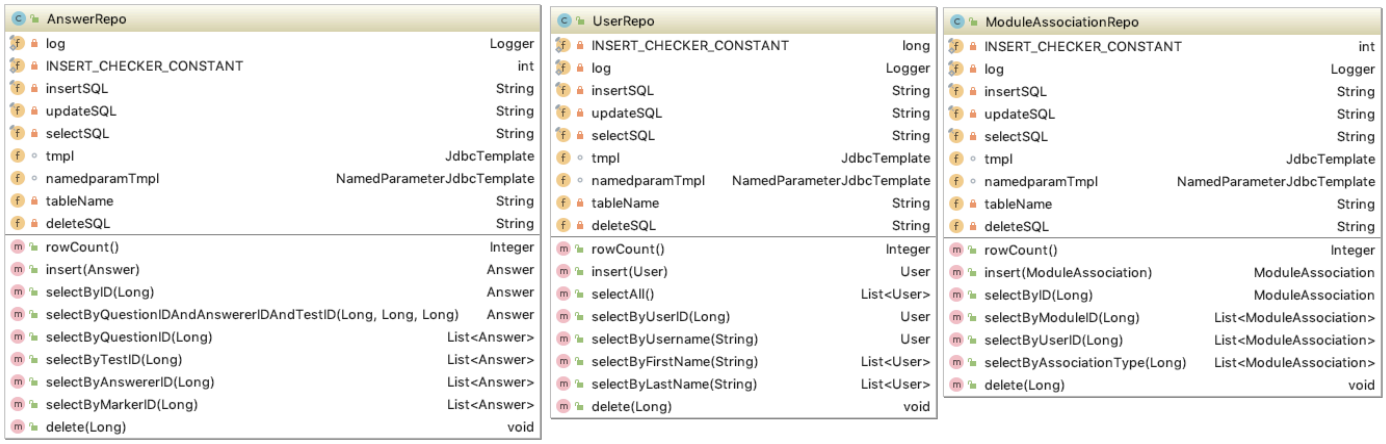


Figure 7 – A snippet of class diagram for the repository layer showing the AnswerRepo, UserRepo and ModuleAssociationRepo classes.

Classes within this layer follow the Singleton design pattern (Erich Gamma, 1994) meaning that there is only one instance of this class within the system. This instance is made available within classes by using the @Autowired annotation (Juergen Hoeller, 2007) to inject it in to each class that requires it. Only one instance was required to perform all the CRUD operations required by this system and the Singleton design pattern was useful in limiting the instances to this.

Each class contains an `insert()` method which takes care of insertions and updates made to the Database table. When an object is passed in to the method its Primary Key will be checked for its validity. If it is invalid, then the method will categories this as a new record and perform and insertion in to the Database table. If not, then the method updates the corresponding record in the Database table with the new attributes stored within the object. Each class also contains a `delete()` method and a method for selecting by the primary ID. Some classes will also have a number of more specified select methods on the basis that it is required in specific circumstances.

Each class uses a variable of type `NamedParameterJdbcTemplate` in order to execute the queries. This class maps attributes of a java object to the parameters specified within the SQL query allowing the object to be inserted in the Database. Similarly, when data is returned from a query this performs the mapping to the java object.

4.4 - The Entity Layer

The Entity layer is comprised of a series of classes that are as close as possibly mirrored to the Database tables. Attribute names within the class are an exact match for the names of columns within the corresponding table. A snippet of the class diagram for this layer can be seen in Figure 8.

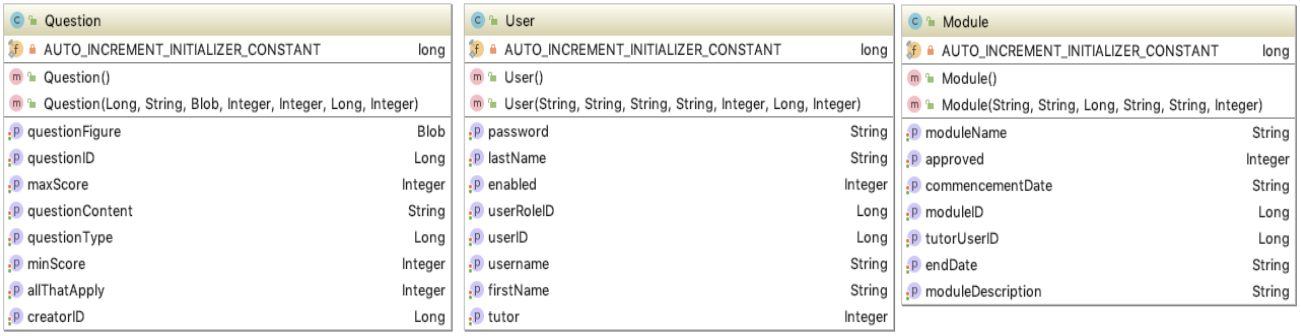


Figure 8 – A class diagram snippet for the Entity layer in the Assessment Assistant system.

The classes within this layer serve one purpose, to encapsulate the data from a table in to objects to be passed around and manipulated by the system. It is for this reason that entities classes are also commonly referred to as Data Transfer Objects (DTO).

The reason that these DTOs are created is simple, when calls are made to the Database there is a delay while the system awaits a response. A larger number of calls means more periods of delay which can have an adverse effect on the performance of the system. So naturally decreasing the number of calls made will reduce the number of these delay periods. The encapsulation of the table data in to the DTO helps decrease the number of calls made by allowing all the data needed to be sent and received over one call, hence reducing the number of calls required to one.

4.5 - The Service Layer

The Service layer consists of a series of classes, denoted by the @Service annotation (Holler, 2007), that encapsulate the application-specific logic. Each class groups together the functions for certain areas of interest within the application requirements. For example, one class will look after module creation and maintenance functionality while another will look after test creation and test sitting functions. A simplified class diagram can be seen in Figure 9, some of the classes within this layer will now be examined in more detail.

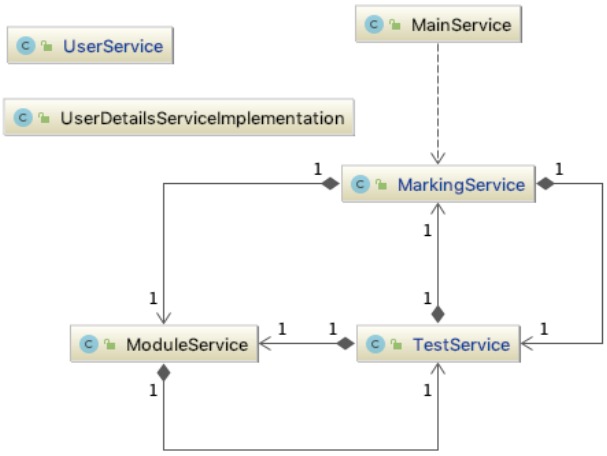


Figure 9 - A simplified version of the class diagram for the Service layer of the Assessment Assistant system.

4.5.1 - Main Service and User Details Service

The `MainService` and the `UserDetailsServiceImplementation` go hand-in-hand for facilitating the logging in and out of a User to the system. When a login request is sent to the system before it even arrives at the login REST method it is required that it passes through Spring Security (Pivotal Software, 2004).

Spring Security is available almost out of the box for the securing of an application. One action that must be performed by the developer is to provide a custom implementation of the `UserDetailsService` interface, which can be seen in Figure 10. The only requirement for this is that it implements the `loadUserByUsername(String)` method. This method will check that the username entered as part of the login function exists in the Database and then if it does returns a `UserDetails` object. This includes the User's username, password and roles within the system for the password to be compared against the one entered as part of the login. If the user does not exist in the system or the password is incorrect then an exception will be thrown, and the login rejected.

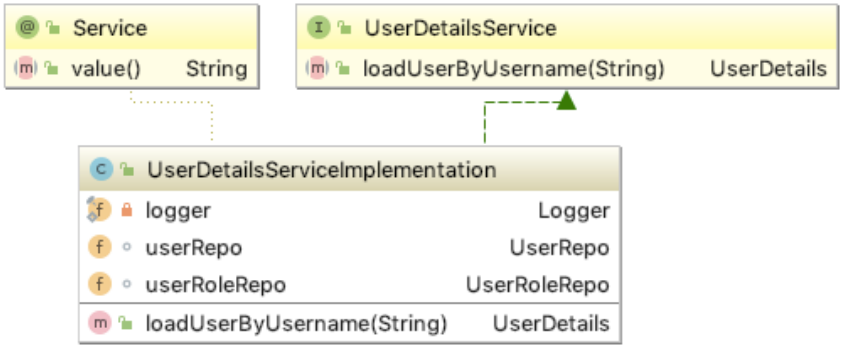


Figure 10 - The `UserDetailsServiceImplementation` class within the Service layer of the system.

Provided that this has been successful the Login rest method is then called which in turn called the `generateToken(String)` method within the `MainService` class, which can be seen in Figure 11. The String parameter here is the username of the logged in user. This method generates the users token for future interaction with the system. Similarly, when a user logs out of the system the `destroyToken(String)` method will be called to invalidate the token.

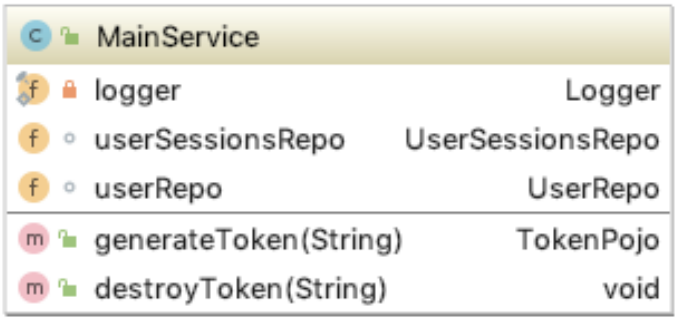


Figure 11 - The `MainService` class within the Service layer of the system.

4.5.2 - User Service

The `UserService` class, which can be seen in Figure 12, encapsulates all of the functions that relate to a user's account. Some notable examples of functions that are provided within this class are; creating or editing a profile, changing or resetting passwords, requesting to become a tutor, admin functions.

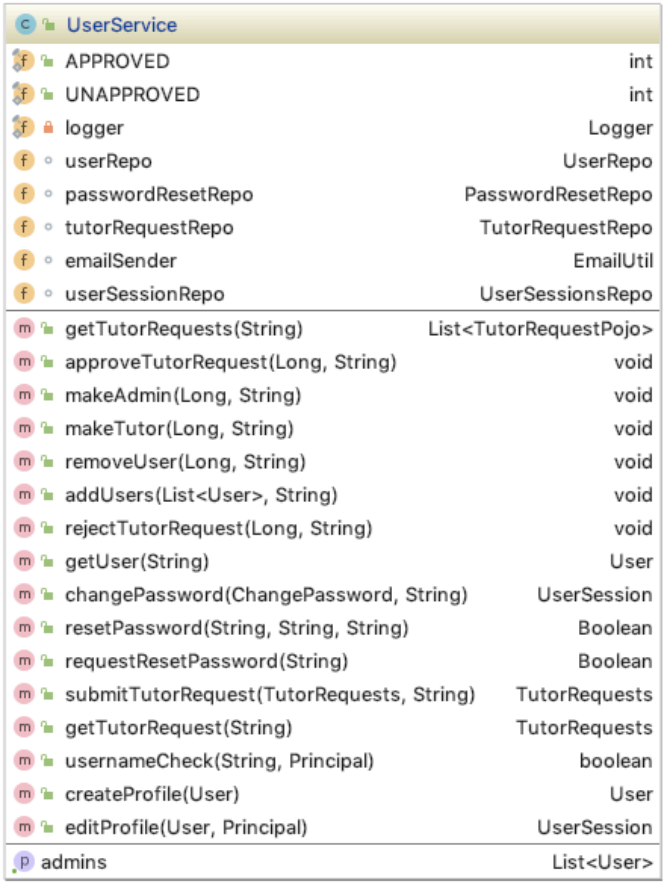


Figure 12 - The UserService class within the Assessment Assistant System.

4.5.3 - Module Service

The `ModuleService` class, as the name suggests, will encapsulate all module functions such as; module creation, adding associates to the modules and generating data for display on the module home page which can be seen later in the chapter in Section 4.7.5.

4.5.4 - Test Service

The `TestService` class encapsulates all of the functions that relate to tests. Some notable examples of functions that are provided within this class are; creating or editing a test, creating or editing questions, scheduling tests, preparing tests for the student, and submitting tests. Some auto marking is included in this class but only the auto marking methods that are executed once as a direct result of submitting a test. The majority of marking functionality is stored in the `MarkingService` class.

4.5.5 - Marking Service

The `MarkingService` class encapsulates all of the functions that relate to marking. Functions that are found within this class are; auto-marking, manual marking, assigning marking, generating chart data for marking pages, and approval of auto-marking.

4.6 - REST Layer

The user interface must be able to interact with the backend component. This will be done through a REST API which allows the User Interface to make http requests to access resources within the backend component by exposing some of its methods as URLs. Parameters for the function are sent as part of the http request in JavaScript Object Notation (JSON) format, as is the default with Angular, and JSON is again returned. The Angular component then converts this JSON to objects for use throughout the frontend system. A sample of some of the REST methods made available by the system can be seen in Table 10, the full list of REST methods can be found in Appendix E.

URL	Parameter(s)	Returns	Type
/marking (MarkingController)			
/getMarkersData	Principal, Long	MarkerWithChart	GET
/reassignAnswers	Principal, Long, List<MarkerAndReassigned>	Boolean	POST
/editAnswer	Principal, Answer	Answer	POST
/editScore	Principal, Answer	Answer	POST
/editFeedback	Principal, Answer	Answer	POST
/addCorrectPoint	Principal, CorrectPoint, Long	Boolean	POST
/addAlternative	Principal, Alternative, Long	Boolean	POST
/approve	Principal, Long	Answer	GET
/getScriptsMarker	Principal, Long	List<AnswerData>	GET
/getScriptsTutor	Principal, Long	List<AnswerData>	GET
/publishGrades	Principal, Long	Boolean	GET
/publishResults	Principal, Long	Boolean	GET
/getCorrectPoints	Principal, Long, Long	List<CorrectPoint>	GET
/removeCorrectPoint	Principal, Long, Long	Boolean	DELETE
/getResultChart	Principal, Long	ResultChartPojo	GET
/getQuestionsResultChart	Principal, Long	List<ResultChartPojo>	GET
/removeAlternative	Principal, Long, Long	Boolean	DELETE

Table 10 - The REST methods exposed as part of the MarkingController within the Assessment Assistant system

The REST API will be restricted to allow access only from the exact location of the frontend component to increase the security around the backend component. The layer itself encapsulates all of the function calls to the server away from the application logic stored in the Service layer and in the same format as the services layer encapsulates groups of functions in to classes.

The Principal parameter that can be seen in all of the REST endpoints listed in Table 10 represents the logged in user that has made the request. This parameter will ensure that the user only ever gains access to the data that is relevant to them, hence increasing the security surrounding the system.

It was important that the frontend code lined up with the backend code in this case in the sense that the REST URLs that the frontend code makes calls to must match the REST endpoints that are exposed by the API which in most projects could prove difficult. However, given the fact that there is one developer for the full stack and the Iterative and Incremental Development model this would be greatly simplified due to the frontend and backend code for each feature being written together.

This section has discussed the different layers of the backend component; Repository layer, Entity layer, Service layer and REST layer. The final component in the system to be examined is the user interface which will be discussed in the next section.

4.7 - User Interface

This section describes the most significant aspects of the user interface along with the choices made by the designer throughout the planning of the system. Starting with the entry point to the website, the login page, and moving through the system as a user would. The UI was designed with the aid of the Bootstrap (Bootstrap Team, 2019) and ngBootstrap (ngBootstrap Team, 2019) frameworks.

4.7.1 - The Login Page

Upon navigation to the website the user is greeted by the login page seen in Figure 13. This page was designed, as was every page within the website, with the intention that all components be clearly visible and distinguished.

Figure 13 - The Login Page for the Assessment Assistant system

The colour scheme remains consistent throughout the website with colours chosen to bolster the designer's attempt to distinguish the components and to give the website a professional feel. This professional feel is further upheld by attempting to provide each page with a minimalist approach to design this is performed through that ensuring there is no overloading of information on screen at any one time.

Throughout the entire website all pages, except for the "My Modules" page, are designed to be contained entirely within the viewpoint of the user. That is to say the content within the page never exceeds the width or height of the viewpoint and thus there is no option to use the browsers built in scroll.

Hence, all information available on the page is available to the user at the click of a button. Information is swapped in and out of the available space within the viewpoint. However, some components do have their own internal scrolling option which was a design decision made with the intent that this was cleaner and more professional than using the browser scroll.

As the system was to be developed using Angular, it would be single-page and dynamic. The means there is only one page (index.html) for the whole site, components are routed in and out of this page using JavaScript - giving the appearance that the page is changing. However, throughout this section each significant area of the website will be referred to as a 'page' for simplicity.

However, the browser itself is only refreshed in instances where the cache must be cleared hence the user can navigate through the system quicker than they would in conventionally designed websites.

4.7.2 - The My Modules Page

After a successful login the user will be navigated to the MyModules page in Figure 14, they have the ability to access to any modules that they are involved with, make changes to their account or add a new module.

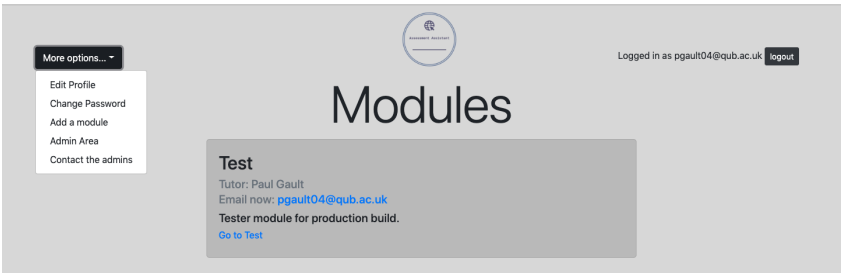


Figure 14 - My Modules (The Home Page), cropped for the purpose of this screenshot

4.7.3 - The Add a Module Page

In the module creation area, which can be seen in Figure 15, the user will be able to add all the necessary information required to create a new module alongside an area for the user to enter a CSV file detailing the users that should be associated to the module. While a new module is awaiting admin approval it will appear in the list of pending modules to the left of the form.

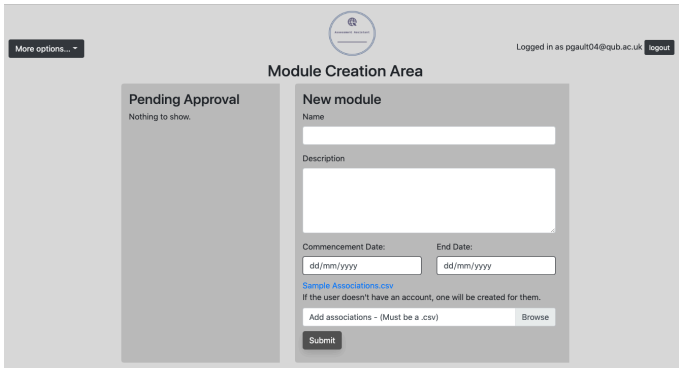


Figure 15 – The Module Creation Area for the Assessment Assistant system

4.7.4 - The Admin Area

The admin area, which can be seen in Figure 16, was designed to allow admins to perform necessary actions such as: approving tutors or modules, adding users, and changing users' permissions. Similar to the Module Creation Area the admins can add new users via a CSV file.

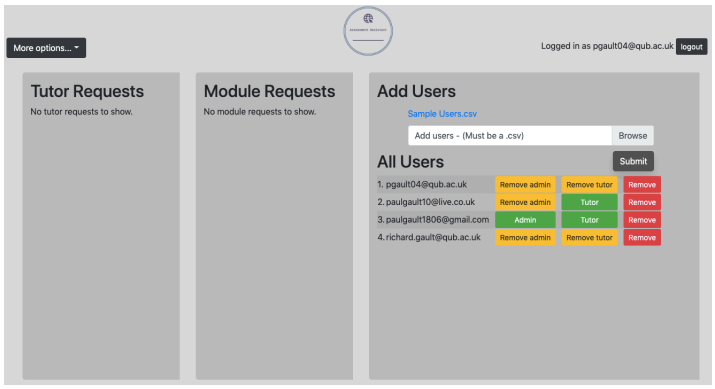


Figure 16 - The Admin Area for the Assessment Assistant system

4.7.5 - The Module Home Area

On the MyModules page each accessible module is displayed in a card which includes the name of the module, a *mailto* link to allow the user to contact the module tutor and a link allowing the user to navigate to the “Module Home” page for that module.

When the module’s tutor navigates to the module home page, they are greeted with something similar to what is seen in Figure 17. Navigation to this page by a student or teaching assistant yields a different set of areas due to different levels of access.

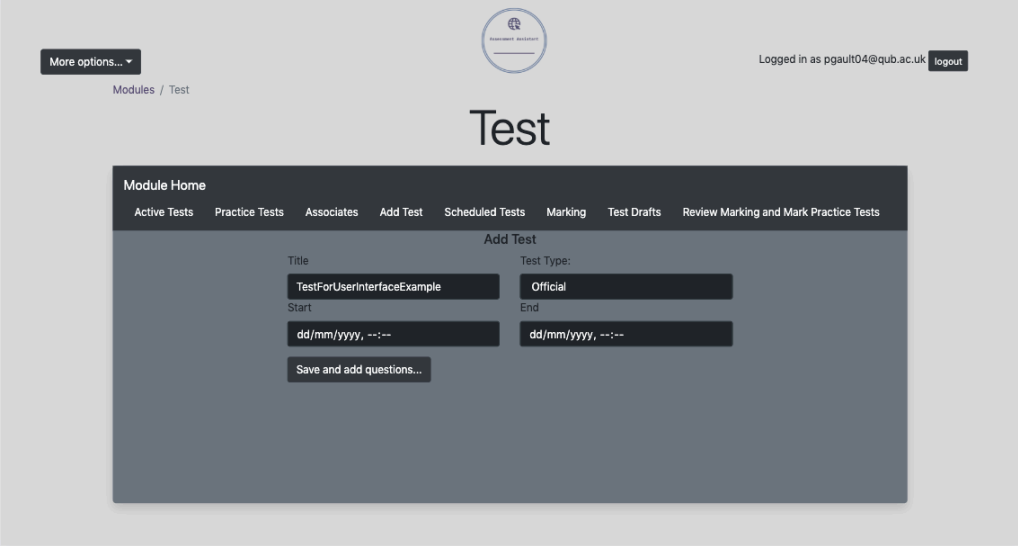


Figure 17 – The Module Home Tutor View with Add Test Selected

When a user clicks on a heading the dark grey area alternates the newly selected components and the component that was previously in view, this means that at any given time the user only sees the information that they want while being easily capable of viewing anything else with a single click.

Normally when a user lands on the page they are automatically directed to the Active Tests area. However, in order to explain the design of the user interface effectively it is necessary to follow the flow through the lifecycle of a new test.

To do so, the user must begin at the “Add Test” component which can be seen in Figure 17, here the user has the ability to add the title and timeframe for the test as well as specify whether it is an official test or practice test.

4.7.6 - Editing a Test

Upon submission of the “Add Test” form the user will be navigated to the Edit Test Page in Figure 18.

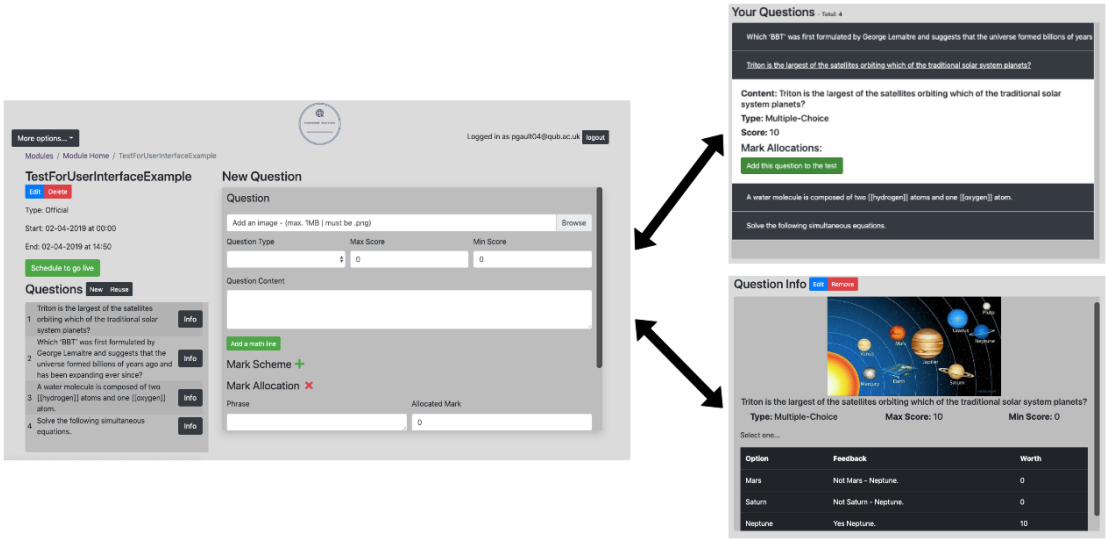


Figure 18 - The Edit Test Page with all components.

The Edit Test page is comprised of a number of components. Starting on the top left side the page is designed to include a breadcrumb trail, the purpose of this is to improve navigation around the site by allowing the user to easily access the two previous pages on their path to the current page.

Below the breadcrumb trail the information pertaining to the test itself is shown, in this area the user has 3 options available. The user can: edit the test which causes the test information to be replaced by a form with the current test values set, delete the test, or schedule the test to go live.

Scheduling the test will result in the user being redirected to the module home page where they will be able to see the test in the scheduled test area assuming the timeframe for the test has not yet been reached, otherwise it will be immediately in the active tests section until it’s time has elapsed.

The right-hand side of the page houses three major components. When a user reaches this page, the right-hand side is by default set to the ‘add question’ component. Here the user is able to choose the type of question to enter as well as all needed information for the question and mark scheme, including what the auto-marking on the backend should look for in an answer and what marks and feedback should be allocated in the event that it appears.

Once the user has entered the question it is added to the area on the bottom left-hand side of the page where a preview of each question is kept in a list. By clicking on the ‘info’ button for the question, the component on the right-hand side of the page switches to the question detail component which can be seen on the bottom right-hand area of Figure 18. In this area the user will also have the ability to remove the question from the test or edit it.

Upon selecting the edit option, the user is greeted with a popup window which asks if they want to edit the question itself or duplicate the question before editing. Duplicating the question creates an exact copy of the question for the user to edit without interfering with auto marking that has been performed on answers that may have been provided for the question in previous tests.

The final component of the edit test page is the reuse question component. By clicking the reuse button, the user is shown all the question they have ever created within the system, this component can be seen on the top right-hand area of Figure 18.

The questions appear as part of an accordion which users can expand to view more information, inside the accordion is a button that allows the user to add the question to the current test.

Once the test has been created and scheduled the next stage in its life cycle is for it to be taken by the students. In order to examine the user interface, the principal user has now been switched from the tutor for the module to one of the students for the module.

4.7.7 - Taking a Test

The take test page is outlined on the left-hand side of Figure 19. It follows a similar design to the edit test page in terms of how it is structured - with a breadcrumb trail and two main areas for components to appear. However, naturally in this case the user has a far more limited range of options on what they can do.

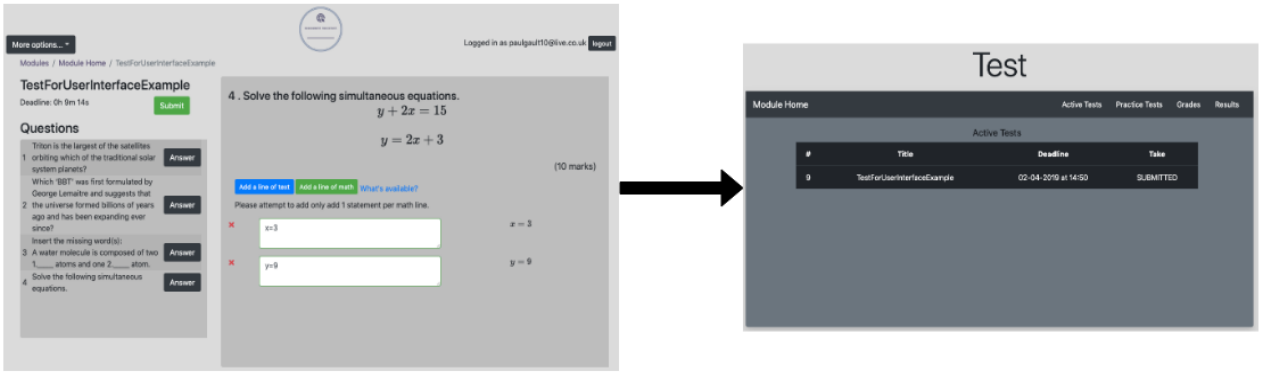


Figure 19 – The Take Test page and submission notification.

Once again starting with the upper left side of the page the user is able to clearly see the title of the test they are sitting, along with the test deadline and the submit button. The deadline for the test appears as a live countdown timer counting down until the time that the test expires, in which case the student's answers will automatically submit. The timer was design with the intention of allowing the student to always know how much time they have left in the test with a degree of accuracy in seconds.

As is the case with the edit test page, all the questions are listed on the left-hand side with a preview of the question content. When the user clicks on the answer button for the question it swaps in to the view on the right-hand side and the user can input their answer.

One such example of this can be seen in Figure 19 for a math type question. With this question type the user can add their answer and working out for a question line by line by clicking the buttons located below the content of the question. The user can add simple text or add mathematical text which is entered as TeX and rendered in to mathematical script to improve readability for both the student and the marker of the answer.

The other question types that the user may possibly be required to answer are; multiple choice which is answered using a radio input, text-based which is answered using a text-area, and insert the word which hides words from the content of the question and provides input boxes for students to respond.

Even though at any given time only one question can be seen, the answer inputs are all contained within the same form. Thus, the student only needs to click submit once and all their answers are collated and sent to the backend once they have confirmed that they are ready to submit.

Once the user has submitted their test, they will be redirected to the module home where they will see something similar to what is on the right-hand side of Figure 19. The option to take the test will be replaced with text informing the student that they have already submitted the test. This will remain there until the test deadline has been reached and the test is moved on to the marking phase of the test lifecycle.

4.7.8 - Marking a Test

Starting with the tutor navigating to the 'Marking' area of the module home page they will be greeted with something similar to what is shown at the top of Figure 20. This area shows the tutor how many answers they have marked out of the total answers they are responsible for, the same for their teaching assistants, and also the overall progress made by all markers. This was designed to provide the tutor with valuable information on marking progress at a glance.

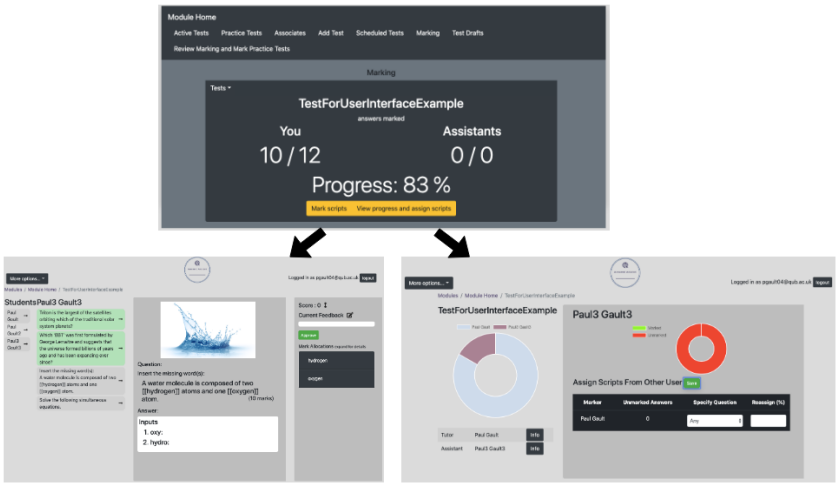


Figure 20 - The mark test page (bottom-left), assign marking page (bottom-right) and the marking area (top).

Initially all scripts are assigned to the tutor of the module by default. However, when they wish to assign some to their assistants, they can click the 'View progress and assign scripts' button which will bring them to the page shown on the bottom right-hand side of Figure 20.

In this page the tutor can see a doughnut chart detailing the total number of scripts that all markers have been assigned. These markers are listed beneath the chart and by selecting a marker they appear on the right-hand side of the page. A second doughnut chart details how many answers they have marked in green, versus how many they haven't marked in red.

Also included in this area is a form where the tutor can assign answers to this marker. The form is shown as a table with each marker listed alongside how many scripts they have left to mark. The two

input options are for the tutor to allocate only the answers for a specific question if they so wish, and the percentage of possible answers that should be allocated to the new marker.

The reason for this design is that when the tutor is performing their initial assignment of answers, they can decide to assign all answers for a specific question to one marker. Doing this will help ensure consistent marking across all answers to the question by eliminating the factor of differing opinions.

The marking page can be seen on the bottom left-hand side of Figure 20. Starting at the left-hand side the individual students are listed, clicking on a student brings up a list of their answers in the next column over and clicking on an answer brings it in to the detailed view in the third column from the right. The far-right column is the control centre for marking the answer.

From top to bottom the score, feedback, approval button and mark scheme are listed. The score and feedback can be manually edited by clicking the buttons at their sides. Once marking has been completed the marker can click the approve button, approved answers are highlighted in green so as to provide a checklist for the marker when working their way through the answers.

4.7.9 - Reviewing Marking and Releasing Results

Once all scripts have been marked the tutor will be able to review the marks allocated to all scripts. To do this the tutor must navigate to the review marking section of the module home page and follow the link to the Review Marking page for the test in question. The review marking page is comprised of three components: student performance, question statistics, and the review marking area.

The student performance chart can be seen on the left-hand side of Figure 21, the class average is displayed at the top and the students' scores appear as points on the chart with the x-axis set as the student's names and the y-axis set to a range of 0-100. In order to allow the tutor to gauge which students need increased support the grades are colour coordinated. If a student whose score is above the class average they appear in green, if their score is less than the class average and within one standard deviation below it the point appears as yellow and any more than one standard deviation below causes the point to appear as red.

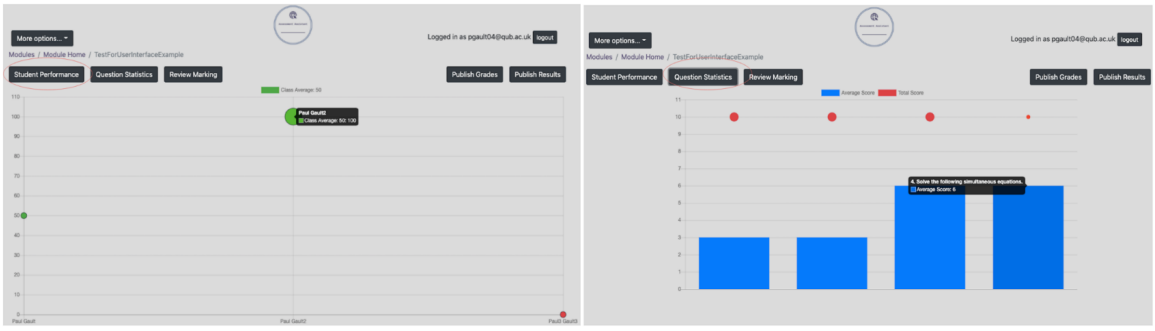


Figure 21 - Review Marking Charts: individual student performance (left) and class average per question (right)

The question statistics chart can be seen on the right-hand side of Figure 21. It is a bar chart that shows how the class scored on average in each question accompanied by a scatter chart that shows the maximum score for each question. The purpose of this chart is to allow the tutor to easily identify areas in which the class generally struggles and to revisit those areas in their teaching or take a different approach to how they deliver it.

The final component of the review marking page is the review marking component which can be seen in Figure 22. As far as the design goes this component is almost identical to the marking component examined previously, except for one crucial difference.

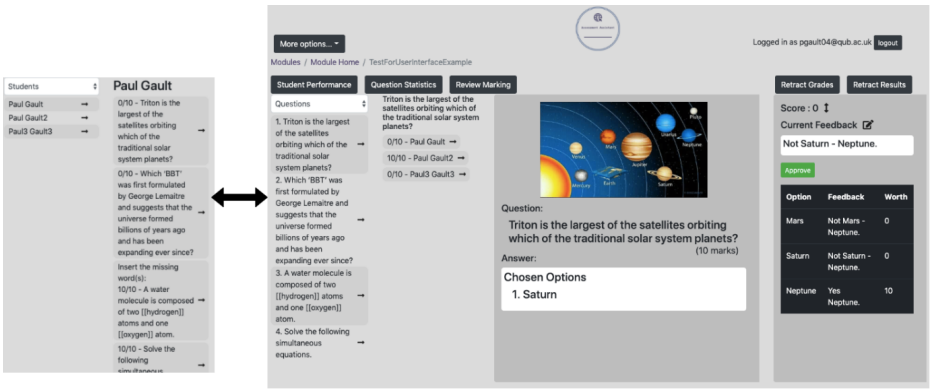


Figure 22 – The Review Marking page on the Assessment Assistant system.

Firstly, the tutor here has the ability to filter the answers by question which is shown on the right-hand side of Figure 22. The tutor can click on a question and the students appear in the column to the right along with the score they received in the question. If the tutor opts to filter by student, the questions that the selected student has answered appear in the column to the right along with the scores they received in the questions.

Once the tutor has completed reviewing marking, they can choose to publish the grades or publish the results by clicking the buttons to the right of the review marking page. Once either has been published the tutor can also retract them at their discretion by clicking the same buttons.

4.7.10 - Viewing Results and Feedback

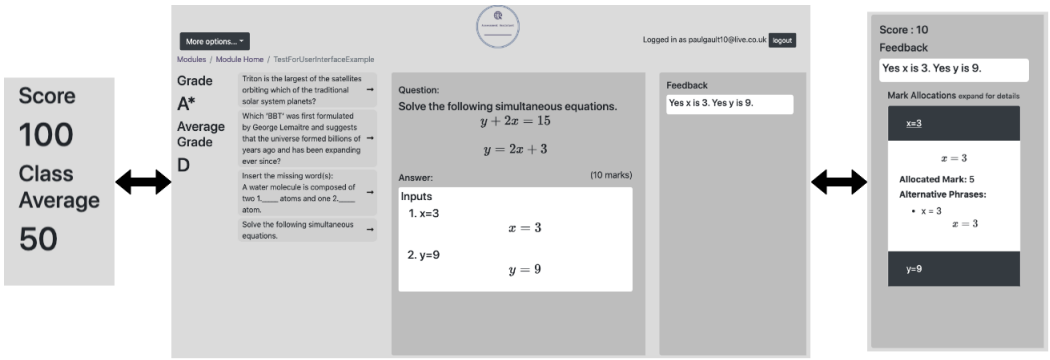


Figure 23 - The Feedback and View Performance Areas

If grades have been published the student may go to the module home page and access the ‘Grades’ area. By following the link for the test, they will be directed to the feedback page which can be seen in Figure 23. The page follows a similar theme to what is seen in the marking pages however all content displayed is read-only and there is no functionality for the student to edit anything on the page.

Starting at the left-hand side the grade achieved by the student is displayed alongside the average grade for the class, the next column list all of the questions within the test. By clicking on any of the questions, the question along with the student’s answer is displayed in detail in the third column. The fourth column contains the feedback that the student has received from the marker.

If the results have been published the student can access them from the 'Results' section of the module home. Following the link will lead the student to the performance page which is almost identical to the feedback page, the differences between them can be seen in Figure 23.

When accessing the performance page, the grades are replaced with exact results for both the students score and the class average. Aside from this everything remains the same except for the right-hand column. Instead of just showing feedback, also shows the score that the student received for the question and the mark scheme used for that question.

Throughout this Section the User Interface design has been examined in the form of a walkthrough of the lifecycle of a test. A complete hierarchical model for all important components in the User Interface can be seen in Appendix F.

This chapter has discussed the most crucial aspects of design of the system. The User Interface, Backend component and Database for the system have been examined along with the REST, Services, Repository and Entity sub-components of the backend component. The next chapter will discuss how these areas have been implemented within the Assessment Assistant system.

Chapter 5: Implementation

5.1 - The Development Process

Given the Iterative and Incremental (IID) development model chosen, the work was broken down in to a series of sprints with a total of 9 planned across the course of the project. The aim was that the system would be built up feature-by-feature based on deliverables that were agreed upon by the stakeholder, Dr Richard Gault, and the developer.

The initial sprint began on the 1/11/2018, before this the project was in the research and planning stage. During this time the developer performed the majority of the system design as has been discussed in Chapter 4. However, given the fact that this was finished slightly ahead of schedule it was possible to complete some development tasks before the beginning of the first sprint.

During this time the spring-boot project was created and built using maven. The *pom.xml* file for the project was configured to reference all the external dependencies that the project would need to pull in when building its jar for deployment, this can be seen in Appendix G. The only external jars that this project was dependent on were those that helped with the bootstrapping of the Database connections, and the bootstrapping of the system's security which will be discussed later in Section 5.1.1. Supplementary to this the package structure for the project was also generated and was in line with the layered architecture of the backend component described in Section 4.1, this package structure can be seen in Figure 24. The REST API is represented via the `controllers` package, named as such due to the fact that it contains all of the systems REST controllers. All other layers are matched to the name of their respective packages. Three other packages are included in the package structure these are: `configurations` which provides an area for annotation-based configuration classes to be added, `pojos` which allows for aggregations of entities in to one class, and the utilities class which provides some useful methods that are needed throughout various classes within the project.

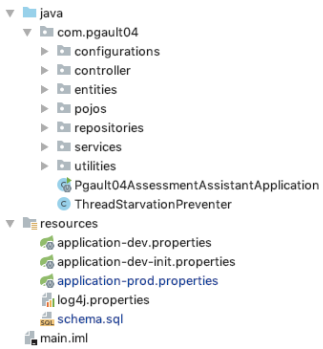


Figure 24 - The package structure for the Assessment Assistant system

During this period the code for the *Utilities layer* was written as it was quick and easy to implement. One class was created for each of the tables within the Database and each attribute of the Database table was added as a private instance variable. The purpose of this was to restrict the access to each variable to be done via a getter and a setter method. Due to the agile nature of the project this code was rigorously unit tested with 100% code coverage gained. The details of unit testing within the system will be discussed later in Section 5.2.1.

The various '.properties' files seen in Figure 24 are included hold the environment variables for the system. These include: the details of the data-source that the system is configured to connect to, the Simple Mail Transfer Protocol (SMTP) details for the system to be able to send and receive emails, and some URLs that that are required for the functioning of the system in each environment. The production properties file can be seen below in Figure 25.

```
server.port=8080

# THE PRODUCTION DATABASE
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/assessment_assistant?useSSL=false
spring.datasource.username=paulg
spring.datasource.password=password
spring.datasource.driver-class-name=com.mysql.jdbc.Driver

# THE MAIL HOSTING DETAILS
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=assessmentassistantqub@gmail.com
spring.mail.password=rzirkxzjcdeuztni
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true

# VARIABLES USED THROUGHOUT SYSTEM

# WHERE THE MAIL COMES FROM
app.email=assessmentassistantqub@gmail.com

# WHERE THE FRONTEND APP IS HELD
app.url=http://206.189.210.86

# RESET PASSWORD LINK (HELD HERE SO IT CAN BE EASILY CHANGED)
app.url.reset=/assessment-assistant/resetPassword/
```

Figure 25 - The application-prod.properties file for Assessment Assistant

At the beginning of the project it became apparent that an SSH connection to the developers QUB web-hosting database would not be possible. Instead the database would need to be hosted elsewhere when running the application in production. However, throughout development it was possible to use a H2 (H2, 2019) in-memory database. This database was created with an identical schema to the one for the production database and initialised by running the spring-boot application with the 'dev-init' properties file. Since the database was in-memory this meant that it could be wiped by simply shutting down the application, which was perfect for testing new features. A separate in-memory database was also created for the unit tests to provide them with their own area for testing functions without interfering with the production data hence making the tests atomic and easily repeatable.

5.1.1 - Sprint 1

As mentioned previously sprint 1 ran from the 1/11/2018 to the 12/11/2018. The sprint began with a meeting with Dr Richard Gault where the deliverables for the sprint were agreed upon. The deliverables were as follows: the login functionality, the home functionality, add a test functionality, add a question functionality and the Repository layer. Due to difficulties faced relating to time constraints, the 'add a test' and 'add a question' functionality were not able to be added in this sprint. This section will discuss the implementation of the deliverables that were successfully completed.

The Repository layer was implemented, in line with the design that was described in Section 4.3. Similar to the Entity layer, there is one class for each table in the database. Classes make connections to the database through the data source bean which is wired in to the class using the `@Autowired` (Juergen Hoeller, 2007) annotation. The common methods of each class are outlined below.

- `rowCount()` - This helps to test the correct execution of other methods. However future functionality could use this method, for example displaying the number of students in a module.
- `selectById(Long key)` - This method selects a unique row from the database by passing its primary key as a parameter. Other '*select'* methods are also available within the classes to select on any attribute that is significant to the class. Selections are performed via a `BeanPropertyRowMapper` object, which converts a row from a table in to an object of the Entity class it is mapped to (Risberg, 2007).
- `delete(Long key)` - Deletes the row which has a primary key equal to the method parameter.
- `insert(Object o)` - When a new object is created via a constructor in one of the Entity classes its primary key is always set equal to -1L. When that object is passed to this method a check is performed to see if the primary key is equal to -1L. If it is then a new row is inserted, and a key is generated on creation of the new row. Hence after the insertion the returned object has the correct primary key value. If an object is passed with a key that is not equal to -1L then an update is performed instead. The insert method can be seen in Figure 26.

```
public Answer insert(Answer answer) {
    BeanPropertySqlParameterSource namedParams = new BeanPropertySqlParameterSource(answer);
    if (answer.getAnswerID() < INSERT_CHECKER_CONSTANT) {
        // insert
        log.debug("Inserting new answer...");

        KeyHolder keyHolder = new GeneratedKeyHolder();
        namedparamTmpl.update(insertSQL, namedParams, keyHolder);
        answer.setAnswerID(Objects.requireNonNull(keyHolder.getKey()).longValue());

        // inserted
        log.debug( message: "New answer inserted: {}", answer.toString());
    } else {
        log.debug( message: "Updating answer: {}", answer.toString());
        namedparamTmpl.update(updateSQL, namedParams);
    }
    log.info( message: "AnswerRepo returning answer: {}", answer.toString());
    return answer;
}
```

Figure 26 - The insert method for the Answer table in the Assessment Assistant system

Given the agile methodology used, at the same time as the repository layer was implemented the unit tests for each class were also written. All tests needed to use the `@Transactional` (Sampaleanu, 2005) annotation so that each change the test made to the database would be rolled back once the test completed. This ensured that when the tests were running procedurally during the build process, the data entered from one test would not affect the outcome of the others.

One of the deliverables for the sprint was the login functionality. In order to provide the security for this, Spring Security (Pivotal Software, 2004) was used. To do this `org.springframework.security` was added as a dependency in the pom.xml which can be seen in Appendix G.

Successfully implementing this required a tweak to the database structure. The `UserRole` table was added to store roles such as '`ROLE_USER`' and '`ROLE_ADMIN`'. The `User` table needed to be changed to have attributes: `username`, `password`, `firstName`, `lastName`, `enabled`, `userRoleID`, `tutor`.

The `SecurityConfiguration` (SC) class was created with access to the data source bean and the `UserDetailsServiceImplementation` (UDSI) class which provides the logic to verify a user's login details. This class includes a password encoder in the form of `BcryptPasswordEncoder` (BCPE) which allows the UDSI to know how the passwords have been hashed in the database to successfully check if the entered password matches upon login.

The SC class includes two overloaded methods named `configure`. The first takes an argument of type `AuthenticationManagerBuilder` which configures the login system to use the USDI along with the BCPE. The second configure method takes argument of type `HttpSecurity`. This method specifies permissions for the `HttpSecurityService`, for example: disabling cross-site reference forgery and setting the session create policy. It also specifies which REST URLs are accessible only to users of a certain type, for example the admin functions are only available to '`ROLE_ADMIN`' users.

The USDI implements the `UserDetailsService` interface which has one method required: `loadByUsername(String)`. In the custom implementation the username is passed in. The method then makes a call to the user table for a user with this username, if the user is found the password entered to the login is then checked against what is stored for the user. If it matches, the user's role is found and a Principal User object, which is the currently logged in user, is created. If no user is found or the wrong password is entered an exception is thrown and the user making the request receives a `401 error`. The custom `loadByUsername` method can be seen in Figure 27.

```
@Override
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    // Necessary info gathered from database
    com.pgault04.entities.User user = this.userRepo.selectByUsername(username);
    // List initialised for authorities
    List<GrantedAuthority> grantList = new ArrayList<>();
    if (user != null) {
        // Gets the users role and adds it to a list if the user exists
        grantList.add(new SimpleGrantedAuthority(this.userRoleRepo.selectByUserRoleID(user.getUserRoleID()).getRole()));
        return new User(user.getUsername(), user.getPassword(), grantList);
    } else {
        // If user is not found an exception is thrown
        logger.debug( message: "{}: not found.", username);
        throw new UsernameNotFoundException(username);
    }
}
```

Figure 27 - The customer implementation of the loadByUsername method in the Assessment Assistant system.

The login page was synced up with this code to provide an area for the user to enter their username and password upon arrival at the website, the MyModules page was created to provide the home area for the website and the ModuleHome page was created to provide an area for users to perform actions on each module. All were implemented in line with their design in Section 4.7 using HTML, CSS and the templating engine Thymeleaf to build the pages based on the user data generated by the system.

5.1.2 - Sprint 2

Sprint 2 began on the 13/11/2018 after review of Sprint 1 with Dr Gault. Given that the add test and add question deliverables were not met within the last sprint it was necessary that they were carried forward. In addition, there was a number of other requests made by Dr Gault which are as follows: fix failing unit tests that have arisen from changes to the schema, make the home page (MyModules) the default page after logging in, create dummy data to populate the system, an area for students to take a test and an area for the student's answers to be marked. During this sprint all deliverables were met aside from the area for taking tests and the area for marking tests. Given that the most significant deliverable implemented was the functionality to add a question, the algorithms behind this will now be discussed.

The two approaches that a Lecturer can use to add a question to a test are; create a new question or reuse a question they have previously created. The two methods to facilitate this appear in Figure 28.

```
/**
 * Carries out actions need to enter a question in to the database
 *
 * @param questionData - collection of all question data available to tutor
 * @param username     - the principal user
 * @return the collection of all question data available to tutor after insertion
 * @throws SQLException image conversion
 */
public TutorQuestionPojo newQuestion(TutorQuestionPojo questionData, String username, Boolean update) throws SQLException {
    logger.info( message: "Request made to add new question in to the database by {}", username);
    Long check = modServ.checkValidAssociation(username, testRepo.selectByTestID(questionData.getTestID()).getModuleID());
    if (check != null && AssociationType.TUTOR == check) {
        List<CorrectPoint> correctPoints = prepareQuestionGeneral(questionData, username, update);
        // Insert the word and Text-based
        prepareNonMultipleChoice(questionData, update, correctPoints);
        // Multiple choice
        prepareMultipleChoice(questionData, update);
        questionData.setBase64(BlobUtil.blobToBase(questionData.getQuestion().getQuestionFigure()));
        questionData.getQuestion().setQuestionFigure(null);
        return questionData;
    }
    return null;
}
```

```
/**
 * Carries out actions needed to add an existing question in to a new test
 * The question must have been created by the requesting user and they
 * must be a tutor of the module
 *
 * @param questionID the id of the question
 * @param testID     the id of the test
 * @param username   the user
 * @return the test question record
 */
public TestQuestion addExistingQuestion(Long questionID, Long testID, String username) {

    logger.info( message: "Request made to add question #{} in to test #{} by {}", questionID, testID, username);

    Tests test = testRepo.selectByTestID(testID);
    Question question = questionRepo.selectByID(questionID);
    Long check = modServ.checkValidAssociation(username, test.getModuleID());
    if (check != null && check == AssociationType.TUTOR && question.getCreatorID().equals(userRepo.selectByUsername(username).getUserID())) {
        return testQuestionRepo.insert(new TestQuestion(testID, questionID));
    }

    return null;
}
```

Figure 28 - Methods to add a question to a test, addExistingQuestion (left) and newQuestion (right)

The method shown on the right-hand side of Figure 28 is the `newQuestion` method within the backend component. When a user generates a new question on the frontend, they send the data they have generated to the backend via the REST API which in turn calls this method within the `TestService` class. This data is packaged together in to in object of type `TutorQuestionPojo`.

As is the case with most actions within the system it is important that the user has the right to perform them. For this reason, a method was written to check if a user has the correct association with a module before allowing them to perform actions. This method can be seen in Figure 29.

```
public Long checkValidAssociation(String username, Long moduleID) {
    logger.info( message: "Request made for {}'s association with module #{}", username, moduleID);
    List<ModuleAssociation> ma = modAssocRepo.selectByModuleID(moduleID);
    User u = userRepo.selectByUsername(username);
    return getTheAssociationTypeID(ma, u);
}
```

Figure 29 - The method written to ensure users have a valid association to a module

When the new question is passed through the `newQuestion` method it is initially passed in to the method `prepareQuestionGeneral` (Appendix I), this method prepares the attributes that all questions have for entry to the database. Once this method is executed the question will be added to the test, if the user wants to use this question again, they can do so through the `addExistingQuestion` method shown on the left of Figure 28. Here the chosen question is taken and inserted in to the `TestQuestion` table along with the ID of the new test it is being assigned to.

5.1.3 - Sprint 3

For sprint 3 it was decided that the main deliverable was to focus on the robustness of the application by improving the current functionality and navigation around the it. Until this stage the application had been a Spring boot application that had content stored the static folder and served using the templating engine Thymeleaf. This has been described in Section 2.1 as the solution that was deemed unsuitable and replaced with the final solution also outlined in Section 2.1. It was at this point in the project that the changeover was made. This was due to the fact that any changes to frontend files, even a small change to CSS, required that the spring-boot application was stopped and started which greatly increased the time taken for development. Upgrading the version of Angular to v6 in order to make use of Angular CLI solved this.

To do this it was necessary to move the frontend files out of the static folder of the spring-boot and use the CLI to turn them in to a standalone application so that changes to frontend code do not require a restart of the backend. This was done by using the `ng serve` command within the CLI, which made the frontend application available at '`localhost:4200/`'. Any time a change was made to the code within the editor and the browser was open at the application, it would automatically recompile the area that had been changed and trigger the browser to be automatically refreshed. This meant that all changes were available almost instantly and greatly decreased the time taken for development.

Any time a new page or component within the site was needed to be generated it was done so via the '`ng generate component compName`' command within the CLI. This was also another way in which the use of the CLI greatly sped up the development process, as running this command automatically created a HMTL, CSS and TS file which were then edited to implement the component.

Given that the frontend files were now stored at an entirely different location from the spring-boot component it was necessary to configure Cross Origin Reference Sharing (CORS) between the two. To do this a configuration method was added to the `SecurityConfiguration` class called `corsConfigurer` which can be seen in Figure 30, this configures the backend to allow requests from a given URL stored as an environment variable in the chosen properties file. For development it allows '`localhost:4200/`' and for production it allows '`206.189.210.86/assessment-assistant/`'.

```
/**
 * Method to ensure that backend server allows communication from the front end application
 *
 * @return a web mvc config
 */
@Bean
public WebMvcConfigurer corsConfigurer() {
    return (WebMvcConfigurerAdapter) addCorsMappings(registry) → {
            registry.addMapping( pathPattern: "/**").allowedOrigins(appUrl);
    };
}
```

Figure 30 - The corsConfigurer method for configuring Cross Origin Reference Sharing

Once the new approach to the frontend application had been correctly set up it was necessary to reimplement the features that had been generated thus far. The login page was recreated similarly to how it had been before. However, this time it was necessary that user credentials were converted to base64 and sent as part of the request header. Once successful the user's credentials are stored and used in every request header meaning the system follows basic authentication, this is an area were the system could be improved and will be discussed within Chapter 6.

The home page (MyModules) was implemented exactly same as before except with a slightly different method structure due to the fact that communication to the backend component needs to be done through Angular services. The services in the frontend component handle all of the HTTP requests to the backend component. The Module home page was also reimplemented for use with the Angular CLI with the add test area now integrated in to module home instead of being a standalone page.

The edit test area was implemented as it had been before and finished off to include the question detail area. This area allows a question to be viewed and edited once it had been added by the user. The `prepareQuestionGeneral` method in 'Appendix H' was repurposed to facilitate updates made to questions being passed through the `newQuestion` method.

5.1.4 - Sprint 4

Sprint 4 fell across the Christmas vacation and naturally, as the reviews needed to be performed at each stakeholder meeting, the duration of this sprint needed to be longer, running from 11/12/18 to 14/01/19. Among the deliverables were some bug fixes; an active username bug, renaming of variables to more sensible names and replacing navigation bars with dropdowns in cases where there may be a large number of values. It was also decided that it would be important to integrate components to allow a test to flow through its lifecycle by; allowing the Lecturer to create the test with text-input questions, students should then be able to answer it and have their answers auto-marked, the student should be able to see their results after they are released, the Lecturer should be able to see the classes performance before it goes live, and the Lecturer should be able to schedule and un-schedule tests.

The active username bug was due to the user's details being displayed to the next user after logout and was caused by the fact that the cache wasn't being cleared. As the website is single-page and dynamic the browser never refreshes due to the fact that components are 'routed' in and out of view, which will be discussed later in Section 5.1.9. Hence when next user logged in, all the cached data from the previous user was still being displayed. There needed to be a way to refresh the browser once when the home page component was loaded but no more than once so that the application does not get stuck in an infinite loop. The solution can be seen in Figure 31, any time a user hits the homepage it reloads once and adds '`?cacheCleared=1`' to the URL, the next time the method recognises that this variable has been added and knows not to reload again.

```
let parameters = [];
for (const s of window.location.search.replace( searchValue: "?",  replaceValue: "").split( separator: "&")) {
  let parameter: string[];
  parameter = s.split( separator: "=");
  parameters[parameter[0]] = parameter[1];
}
if (!parameters["cacheCleared"]) {
  (window as any).location.search = '?cacheCleared=1';
}
```

Figure 31 - The caches clearing method to clear previous users details on the home page

To allow for scheduling and un-scheduling tests a method `schedule(Long testID)` was added. With logic to check if the test is scheduled then it un-schedules it or if it is unscheduled it schedules it. Meaning that the method can be used for both actions. When a test is scheduled the students from the module are emailed with a notification from the system.

The edit test page was updated to allow for questions of three types to be added, as opposed to the one type which was agreed upon. The three types added were; text-based, multiple-choice and insert-the-word. Multiple-choice allows the lecturer to add options to the question, each option has an allocated mark and allocated feedback for when it is chosen by the student. Text-based has mark allocations which contain a phrase and a number of alternatives, if any of these appear in an answer the score for this mark allocation and its feedback are added to the student's answer. Insert-the-word works exactly the same as text-based - except each mark allocation must represent a word in double square brackets within the question content. When the student sees the question these words are replaced with blank spaces. If a student enters the correct word in the input space for that missing word the mark is automatically given.

Validation was added during this sprint for all form inputs, including ensuring mark allocations are not worth more marks than the question itself. Dates were validated using the Angular date pipe to ensure that the start date is never after the end. For insert-the-word questions the validation also ensures mark allocations can't be entered for phrases, that don't actually appear in the question.

Functionality to add images to questions was added. It was decided that instead of storing images in directories on the server it would be sensible to add them to the database instead to simplify the maintenance of the server. To do this the image is uploaded by the user and converted to a base64 string which allows it to be passed through HTTP to the backend component. Here it is again converted to an SQL Blob type and stored in the database as a sequence of bytes. This was necessary as it is not possible to send a Blob through HTTP. The images were limited in size to 1mb to ensure that an unnecessary amount of storage is not used.

The take test page was implemented during this sprint. As part of the submit process the '`checkForIdenticals`' method which can be seen in Figure 32 was added to the `TestService` class. This method scans the database for any stored answers to the question that may have been entered as part of this test or as part of a previous test and if one appears then the answer is allocated the same marks and feedback as its counterpart. This helps to ensure consistency in marking not only across the test but across all tests across all years that include this question. This is not required for multiple-choice as the auto-marking for this has 100% accuracy due to the controlled amount of input possibilities.

```
/*
 * Calls identical checks for questions of certain types
 */
private boolean checkForIdenticals(QuestionAndAnswer questionAndAnswer, boolean answerMatch) {
    List<Answer> answers = answerRepo.selectByQuestionID(questionAndAnswer.getQuestion().getQuestion().getQuestionID());
    if (questionAndAnswer.getQuestion().getQuestion().getQuestionType().equals(QuestionType.TEXT_BASED)) {
        answerMatch = checkForTextBasedIdenticals(questionAndAnswer, answerMatch: false, answers);
    } else if (questionAndAnswer.getQuestion().getQuestion().getQuestionType().equals(QuestionType.INSERT_THE_WORD)
                || questionAndAnswer.getQuestion().getQuestion().getQuestionType().equals(QuestionType.TEXT_MATH)) {
        answerMatch = checkForInputBasedIdenticals(questionAndAnswer, answerMatch: false, answers);
    }
    return answerMatch;
}
```

Figure 32 - The checkForIdenticals method in the TestService class used to ensure consistency in marking (format changed to fit screenshot)

The auto-marking methods for when an identical is not found were also implemented during this sprint, these methods can be found in Appendix I. Naturally a different auto-marking technique must be used for each different question type.

Starting with the simplest, the multiple-choice auto-marking is performed by checking which option(s) the user has selected. It allocates an accumulation of the scores and feedback that are included as part of these options during the question's creation.

For the auto-marking of the text-based questions, upon creation of the question a mark scheme is formed by adding the mark allocations (correct points) and their alternatives as something that the answer can be marked against. When the auto-marking is performed the method loops through each of the correct points for the question and checks if the answer includes each, if there is no match for a correct point the method checks all of its alternatives. If a match is found for a correct point or an alternative, then the score for the correct point and its feedback are added to the student's answer.

Auto-marking for insert-the-word is performed in the exact same way as for text-based. However, except for checking if the answer contains the phrase for the correct point or its alternatives it must instead be an exact match. Another extra factor considered by the insert-the-word auto-marking is that the input is indexed at the same point as the missing word in the system.

The assign marking area was also implemented in line with its design in 4.7.8 during this sprint. This meant the only deliverables that were not met during this period were: the student should be able to see their results after they are released and the lecturer should be able to see the class's performance before it goes live. As such these deliverables were planned for implementation at a later time in the project.

5.1.5 - Sprint 5

For sprint 5 it was decided that some bug fixes would be necessary for the features that were implemented over the previous sprint. Firstly, a bug had arisen causing the un-scheduling of tests to not work. Secondly, the auto-marking for insert-the-word questions had developed a bug which needed to be resolved. In addition to these bug fixes it was decided that the developer should add the mark test, review marking, create account and admin area functionality for the system.

The un-scheduling bug was caused by an issue with the system not updating on the frontend after the change, so the test was appearing as though it had never been un-scheduled. This was fixed by adding calls for getting the scheduled tests and test drafts after the action is performed. For the auto-marking bug the input-based questions (insert-the-word) are answered by populating the Inputs table in the database. As such this leaves the `answerContent` column in the Answer table empty. Thus, the system was allocating equal marks to every input-based question as it was seeing that the `answerContent` column was empty.

The mark test page was implemented during this sprint with the inclusion of two methods that helped to ensure consistency of marking among scripts within the test. The `editScore` method was added for when a marker manually adds a score to an answer, the system finds identical answers within the test submissions and gives the same score. The `editFeedback` method similar to the before method was added for when feedback is manually given to an answer and causes the system to find all identical answers within the test and gives those the same feedback. These methods behave in a similar

manner to the `checkForIdenticals` method outlined in the previous section, they can be seen in Appendix J.

When either of these actions are performed the flow is initially passed in to the `editAnswer` method. This method simply provides the check that the user has the correct permissions to perform the actions that they are attempting, via the `checkValidAssociation` method from Section 5.1.2. The answers that are relevant to the marker are then determined, if the user is the lecturer then all answers for the test are considered for this form of auto-marking, if the user is a teaching assistant then only the answers that have been assigned to them are eligible.

Using the `editFeedback` flow as an example, the `checkTypeAndUpdateFeedback` method is then entered. Here the system checks what question type the question is that the answer belongs to. It then checks if the answer is identical to the one that has been manually marked. For a text-based question, it simply checks for an exact match in the answer content. For multiple-choice, it checks if the options entered are identical and for insert-the-word questions it uses the `checkInputsForSimilarity` method which can be seen in Figure 33. This method compares the inputs for the answers and returns the number of them that are identical.

```
/*
 * Checks inputs for answers are identical to answer being checked against
 * If every element in the list matches the the size of list should be returned
 */
private int checkInputsForSimilarity(List<Inputs> inputs, Answer a) {
    List<Inputs> inputToCompare = inputsRepo.selectByAnswerID(a.getAnswerID());
    return (int) inputs.stream().
            filter(i -> inputToCompare.stream().anyMatch(iToCompare ->
                    i.getInputIndex().equals(iToCompare.getInputIndex())
                            && i.getInputValue().equalsIgnoreCase(iToCompare.getInputValue()))).count();
}
```

Figure 33 - The method within the MarkingService class which checks for input similarity for insert-the-word questions

If an answer is found to be identical then the `updateFeedback` method sets the checked answers feedback to be exactly same as the feedback for the answer that has just been manually marked. The flow is identical for `editScore`, except at the end the score is updated not the feedback. Hence these methods ensure that all answers which are identical are given the same score and feedback by the marker.

The deliverables for of this sprint that were not implemented were the review marking, create account and admin areas. The reason they were not completed was due to the unexpected time taken to implement the methods that have been described throughout this section that can be found in Appendix J. In addition, some extra features of mark test had yet to be implemented so they were passed along to the next sprint.

5.1.6 - Sprint 6

The deliverables for this sprint were to: finish off the mark test area, create the review marking area and to begin to research implementing the math type questions. Some areas for change had been identified in the mark test area upon review by Dr Gault.

As for the changes to the mark test area, when the marker performed an action on an answer and the action was completed, instead of the view remaining on the answer it would automatically swap back to the first answer within the list. This made it very difficult for the user to know whether the action they completed was successful. As such this was corrected to ensure that the view stayed on the current answer using the code snippet in Figure 34. It takes the index of the answer that has been selected and once the view refreshes it chooses the answer at this index again to be displayed. Another improvement that was added was highlighting any answers that have been approved by the marker in green in the view.

```
this.studentDetail = this.studentSet[this.studentCounter];
for (let x = 0; x < this.scripts.length; x++) {
  if (this.scripts[x].student.userID == this.studentDetail.userID) {
    if (this.answerCounter == -1) {
      this.answerDetail = this.scripts[x];
      break;
    } else {
      this.answerDetail = this.scripts[this.answerCounter];
      break;
    }
  }
}
```

Figure 34 - The student detail counter to ensure that the view remains on the chosen answer even after refresh

The review marking page also implemented during this sprint was almost identical to the mark test page. The key differences however are: firstly the fact that only the Lecturer has access to this page so all answers are displayed not just the ones that are assigned to the marker and secondly the marking view from the mark test page is swapped in and out with a chart that details how each student has performed in the test.

During this sprint the math question implementations were investigated and ng-katex library (Alpert, 2019) emerged as a front-runner. This library was perfect due to its ease of integration with angular. It allows the user to input TeX (Knuth, 1978) and have it quickly rendered as mathematical formulae. As such this was chosen to be the method of implementation in a later sprint.

5.1.7 - Sprint 7

Around the time of sprint 7 it was decided that the system should undergo some User Acceptance Testing, which will be discussed later in the Section 5.2.3. As such a major push was made in order to get the system ready for it in the coming months and there was a large number of deliverables for this sprint. The deliverables were broken down in to a number of areas based on the various user types: admin, teaching assistant, student and lecturer. The deliverables in the admin section were to create the admin area and the password reset for the system. The teaching assistant section includes two bug fixes to ensure that active tests are viewable by teaching assistants but not accessible and one to fix the fact that there are no marking stats shown on the module home page for teaching assistants. The student section includes a requirement for the performance area to be configured so that students can see results, a similar area for the student to see feedback when only grades are published not the exact result and functionality for students to query the marks given to them in practice tests. Finally, the lecturer section contained the following deliverables: an assign marking bug needed to be fixed, auto-marking needed to auto-approve multiple choice questions and input based questions, add module functionality needed to be added as well as add student's functionality.

Both deliverables outlined in the admin section were met within this sprint. The admin area was implemented in line with the design described in Section 4.7.4. With requests to become a tutor on the site shown in the left-hand column, requests to add a new module in the middle and a list of all the users in the system on the right with buttons to edit their privileges in the system. Supplementary to this a number of other admin related features were implemented during this sprint such as: become a tutor, create a profile, edit profile and change password. The most notable feature that was added as part of this however was the password reset functionality. When a user forgets their password, they can follow the link at the login page where they are prompted to enter their email address. After checking if the user exists and retrieving the user's password reset information this is passed to method shown in Figure 35.

```
/**
 * Sends an email to a user with the link to reset their password upon their request
 *
 * @param user          - the user who wants to reset
 * @param passwordReset - the object containing necessary password reset information
 */
@Async
public void sendPasswordResetMessageFromSystemToUser(User user, PasswordReset passwordReset) {
    SimpleMailMessage message = new SimpleMailMessage();
    message.setTo(user.getUsername());
    message.setSubject("Password Reset for: " + user.getUsername());
    message.setText("This is an autogenerated email from Assessment Assistant:\n" +
            "You have requested to reset your password, to do so please navigate to the following link: "
            + appUrl + resetUrl + passwordReset.getResetString());
    message.setFrom(SYSTEM_EMAIL_ADDRESS);
    emailSender.send(message);
}
```

Figure 35 - The method used to send the password reset email to a user

Every user, upon creation of their account, will be allocated a random reset string. When the email is sent out to the user it will have a link to the password reset page with the user's reset string appended on to it. The URL is added as an environment variable from the application.properties file to account for the fact that the frontend component will be held at different locations in different environments. The email is annotated with the `@Async` annotation, as is all others within the application, to allow the

compiler to execute it as a new thread. This is done so that the current thread can continue on with its execution without having to wait for a response from the email server which can take a considerable amount of time. When the user follows the link in the email and enters a new password, the system ensures that the reset string in the URL matches what is stored in the database before allowing the password to be changed.

All other deliverables were met within this sprint aside from the implementation of the practice tests. It was decided that given the amount of time it would take to implement this it would be better to save it for the next sprint and instead implement some features to improve upon the functionality that was already there.

In the add question area a minimum score field was added, this allowed the lecturer to set the minimum number of marks that are allowed for an answer to a question. This would be of use in the event that an assessment allows for negative marking, but the lecturer wants to place a cap on how low the mark can go for each question. As such the `validateScore` method that is used at the end of the auto-marking process was updated, it can be seen in Figure 36. The method checks if the allocated score is below the minimum score and if so, it alters it to match the minimum score. If this is not the case, then the method checks if the score is above the maximum score and if so it amends the score to be the maximum.

```
/*
 * Checks if the score given is above or below max or minimum scores and alters to match
 */
void validateScore(Answer answer, Question question) {
    if (answer.getScore() < question.getMinScore()) {
        answer.setScore(question.getMinScore());
    } else if (answer.getScore() > question.getMaxScore()) {
        answer.setScore(question.getMaxScore());
    }
    answerRepo.insert(answer);
}
```

Figure 36 - The validate score method to ensure auto-marking never goes beyond the boundaries

In order to identify which students needed increased assistance, Dr Gault suggested the points should be represented on the student performance graph displaying in different colours depending on how the student had performed in the test. It was agreed that all users that achieved the class average or above would be given a green point on the graph, user that were below the class average by less than one standard deviation would be given a yellow point on the graph and users below this boundary would be given a red point on the graph. The method used to calculate the standard deviation can be seen in Figure 37.

```
/*
 * Calculates standard deviation in the score from a set of Answers
 */
double calculateStandardDeviation(List<Answer> answers) {
    double sum = answers.stream().mapToDouble(answer -> answer.getScore().doubleValue()).sum();
    double mean = (sum) / (answers.size());
    double newSum = answers.stream().mapToDouble(answer ->
                    ((answer.getScore().doubleValue() - mean) * (answer.getScore().doubleValue() - mean))).sum();
    return (Math.sqrt((newSum) / (answers.size())));
}
```

Figure 37 - The method used to calculate standard deviation for customising points on the student results graph

5.1.8 - Sprint 8

Given the large number of deliverables implemented in sprint 7 it was necessary in this sprint that some of those features would be revisited and improved upon per some of the issues highlighted in the meeting with Dr Gault. To start, the was an issue with the way that the tokens were being destroyed upon logout of the system. In the previous sprint the functionality to add students was implemented by way of a user uploading a CSV file containing the details for the students, it was suggested by Dr Gault that some extra details be added to the CSV file and that the file entry be added to the admin page. In addition, the widget to view users from the admin area would also need to be added to the module home page. It was decided that the mathematical renderers should be delivered with this sprint and finally the practice tests which had been carried on from the last sprint would need to be implemented.

The add user csv that was implemented as part of sprint 7 in the `addUser` method and can be seen in Appendix K. This method is written in TypeScript and was designed to read in the users uploaded csv file and convert the records into objects of type User. The file is read in and converted to a string which is then split on every new line, resulting in each record being divided up in to an array. The first item in the array which corresponds to the headers of the file which is then split on each comma in order to determine what order the headers have been entered in to the file. This means that regardless of the order the file can still be read correctly. The remaining rows in the file are then iterated through in a for loop and split on commas. Each item of this new array is added to an attribute of the User object which corresponds to the header for the column it appeared in. If no errors are found the User object is pushed to an array of Users which is later sent to the backend component. Once this reaches the backend component each item in the list is checked to see if the user already exists. If not, the new User is created, and they are emailed their temporary password. The code for these backend actions can be seen in Appendix L. During this sprint the '`firstName`' and '`lastName`' columns were added to the CSV, initially it had been intended that users could add these themselves in the 'Edit Profile' area once their account had been created for them. However, it was pointed out by Dr Gault that this could be a problem if they do not add the details and their name appears as blank when they make submissions to tests.

The functionality for the lecturer to assess students with use of mathematical questions was added during this sprint using the ng-katex library (Alpert, 2019) which allows users to enter TeX and have it displayed as mathematical formulae. The library was added to the application as a dependency using the `npm install ng-katex` command on the Angular CLI.

In order to add mathematical content to a question a new table in the Database was created named QuestionMathLines where this could be stored. Any number of these could be entered by the lecturer creating the question on the front end. Some HTML was added to the 'Add Question' component to allow for this, it can be seen in Figure 38. An area is provided for the user to enter their TeX input which is bound to an ng-katex tag below which is where the rendered mathematical text will be output. An example of this output can be seen in Appendix M.

```
<div class="row">
  <div class="col-sm-6">
<textarea class="form-control" rows="2" type="text" name="equation_{{rowIndex}}" id="equation_{{rowIndex}}"
          [(ngModel)]="mathLine.content"></textarea>
  </div>
  <div class="col-sm-6">
    <ng-katex [equation]="mathLine.content" [options]="options"></ng-katex>
  </div>
</div>
</div>
```

Figure 38 - The HTML to allow a user to enter TeX and have it displayed as Mathematical Formulae

In addition, a new question type was added named ‘Text-Math’. While all questions have the ability to contain QuestionMathLines added by the lecturer, only questions of this new type will allow for the students to enter a mathematical response. As such there needed to be some auto-marking rules added to consider this. The `autoMarkCorrectPointsForTextMath` method shown in Appendix I, was added for this and is marked in a very similar nature to that of insert-the-word questions. When a ‘Text-Math’ question was chosen by the lecturer the mark allocations section would give them the opportunity to choose whether the phrase was of type text or math and likewise for the alternatives of the mark allocation. This was done to allow the lecturer to see how these would be rendered if they were in fact intended to be mathematical.

When a student is taking a test the QuestionMathLines are shown in rendered form beneath the question. For ‘Text-Math’ questions the student will see ‘Math+ Text+’ beneath the questions content, clicking these creates either a math input or text input - the student can create as many as they like.

When a lecturer creates a test, a new option was added to choose official or practice tests. Practice tests are shown in a separate section on the module home. Lecturers can ask and create questions just as they can with normal tests however the key difference is that practice tests are amendable even during the phase where they are live. Students can submit multiple times unlike official tests and feedback is available as soon as student submits. In addition, the student is given an option to contact their lecturer to query feedback that they have received.

Once all of the deliverables for this sprint were completed the developer was able to turn focus to the deployment of the application. Starting by deploying the database, an account for digital ocean was obtained and an Ubuntu server was used to store the system. A MySQL server was downloaded on to the machine and the schema for this system was added.

The application-prod.properties file was included to point the system towards this production database. The .jar file for the backend component was then also added to this server and ran using the following command:

```
nohup java -Dspring.config.location=application-prod.properties -jar backendAA-0.0.1-SNAPSHOT.jar > assessment-assistant-backend.log 2>&1&
```

The nohup command ensures that the application would not be stopped when the user’s SSH in to the machine timed out after two hours. The production properties file was used and the logs for the system were outputted to a file for when problems arise.

5.1.9 - Sprint 9

At this stage the coding phase of the project was completed, and the deployment had begun. The deliverables for this sprint were simply to deploy the frontend component to the server and prepare the application for user acceptance testing.

When deploying to an angular project to production the user can run `ng build --prod` which compiles all the components created during development in to a handful of files. All HTML files collated in to the index.html file, while all CSS all goes to main.css and all TypeScript files go to the main.js file. As an angular project is single-page and dynamic it allows the user to move around the application by use of routing. Routes are added to the app.routing.ts module. When a route is referenced in a link in the application, clicking on that link swaps the view to the destination component for the route.

The frontend component makes a request to the backend component so needs to know where it is. Hence, different environment variables are needed for production and development through .env files which are similar to the property files in the backend component. These files on the frontend component simply contain the location of the backend component to send requests to.

When a user makes a request for a certain URL the server naturally looks for the exact file within the server's directory listing. However, given that the process of routing is used and there is only one index.html file, when the server searches for any other file it will not be able to find it. To solve this problem a .htaccess file was created to redirect all requests to the index page.

The next section of the report will discuss the various testing strategies used throughout the systems implementation and the goals which they sought to achieve.

5.2 - Testing

Given the nature of the application it was important that there was adequate testing of the frontend component and the backend component. It was decided that white-box testing for the backend, black-box testing for the frontend, and User Acceptance Testing for the overall system would be performed. It was necessary to use all of these testing strategies in order to ensure the quality of the end product.

5.2.1 - White-box testing

White box testing is defined as 'testing that takes into account the internal mechanism of a system or component' (ISO, 2017). In this project the white-box testing was executed in the form of unit tests. It was important this form of testing was performed so as to spot any bugs in the code that were introduced during development that might not be easily spotted by general use of the application. Given the agile approach taken to development in this project it was important that unit testing was performed little and often so that bugs could be identified and removed early before they have a chance to snowball and create even bigger problems down the line.

In order to measure the success of this testing the built-in code coverage measuring tool from intelliJ IDEA (Jet Brains, 2018) was used in line with TeamCity build agent. Every time a build is performed the entire test suite is executed by the agent in order to verify whether the code is fit for purpose. If

there are failing unit tests within the build; then the build fails, the production files are not created and hence cannot be deployed. When the build is successful the code coverage statistics are made available on the build agent. The coverage statistics for the final build of the project can be seen in Figure 39. As can be seen 99% of all lines within the backend code were tested with a total of 398 tests written across the course of the project. Achieving 100% code coverage was not feasible due to the fact that the applications `run` method cannot be tested without creating a new application context. This would greatly slow down the build as it is expensive in both memory and time.

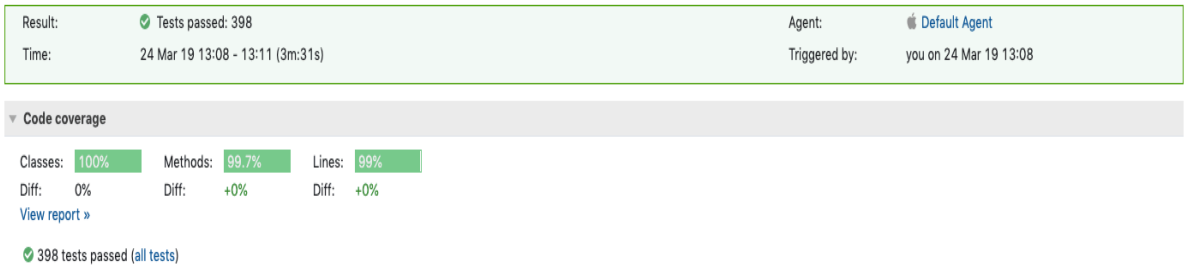


Figure 39 - The code coverage statistics of the backend component show on the TeamCity build agent

The full details of the unit test results can be found on the project's GitLab space at:

https://gitlab2.eeecs.qub.ac.uk/40126005/CSC7058/blob/master/Test%20Documentation/Unit_Test_Results.csv

5.2.2 – Black-box Testing

Black-box testing is defined as 'testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions' (ISO, 2017). This testing was performed at the end of the projects execution in order to determine the percentage of requirements that were successfully implemented and gauge the success of the project itself. This success level of the project will be discussed in more detail in Chapter 6 during the evaluation of the project.

Black-box testing was used to test both the functional and non-functional requirements for the project, the test cases and results can be found in Appendix N and Appendix O. However, some of the non-functional requirements were not possible to test in this way without being independently evaluated by a separate tester. Given the fact that some of these requirements may be subjective in how successfully they were implemented it did not make sense for the developer to test them due to bias. Once such example for this was the requirement:

'The application must be easy to use for new users.'

The most sensible way to test this would be through allowing as many users as possible to use the system and allow them to report back any issues that they face with the system. For this reason, the testing for this requirement was passed on to the User Acceptance Testing phase of the test plan which will now be discussed.

5.2.3 - User Acceptance Testing

User Acceptance Testing is defined as “testing conducted to determine whether a system satisfies its acceptance criteria and to enable the customer to determine whether to accept the system” (ISO, 2017). As part of this project it was decided that a select number of students would undergo this type of testing of the system in order measure its level of acceptability.

Dr Gault gathered the students for the testing while the developer prepared the system and drew up a form for the testers to use. The form was made using google forms can be found at the following link: https://docs.google.com/forms/d/1r58rpF649uCN7nsJ9tt_7uWSnHZHOI2IQkd1hhOGoYU/edit#responses

The testing was carried out over the course of a week and the students were asked to perform certain actions on the system and rate these actions from 1-10 based on how intuitive they are. In addition, at the end of the test the user would be asked the respond to the statement seen in Figure 40. Of the 7 users who responded to the User Acceptance Testing the average response for this was 8.4/10, indicating clear preference of the Assessment Assistance system among the sample. Across all questions asked as part the testing the average score across the sample was never lower than 7.5.

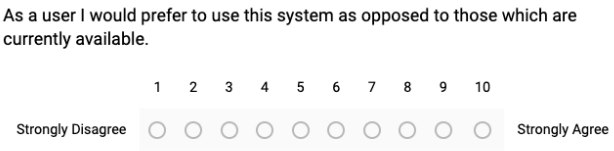


Figure 40 - Sample question from the User Acceptance Testing form.

The full set of testing results can be found at the project’s GitLab space at the following location: https://gitlab2.eeecs.qub.ac.uk/40126005/CSC7058/blob/master/Test%20Documentation/Assessment_Assistant_User_Testing_Results.xlsx

The document also allowed the users to provide answers to some open response questions in relation to any bugs, issues or suggestions that they identify during the user testing. A clear pattern emerged in the fact that a number of users had criticism for the lack of confirmation method in the reset password flow. Some user responses were as follows:

- “Confirmation messages when you complete a task, combined with redirection to a different page- if I reset my password and I'm left on the password reset page I might think it didn't work properly unless I'm told it did. Also maybe tell people the password requirements before they enter them.”
- “Validation message to show that their password has been reset successfully.”
- “Some submissions don't have confirmation which makes it a little confusing as to whether you've successfully submitted or not. Marked those ones as 8 intuitiveness.”

In an attempt to improve the system as per the user’s desires, a confirmation message was added to the reset password page. This change was built and deployed and is now available.

Throughout this chapter the implementation of the project’s solution has been discussed in line with the agile methodology used. The project was executed as a series of sprints with rigorous testing performed via the following testing strategies: white-box testing, black-box testing and user acceptance testing. The next chapter will provide a critical evaluation of the solution alongside some of the difficulties that were faced throughout, some ideas for future improvement and metrics to measure the success of the project.

Chapter 6: Evaluation and Conclusion

This chapter will provide an evaluation in to the success of the project in comparison to the criteria stated in the Chapter 3 of the report. Starting with a general evaluation of the success, the chapter will then move on to provide a critical evaluation of the technology used to implement the final software and discuss any alternatives, where appropriate, that would have been used given the opportunity to start again. After this the discussion will then turn to what suggestions there may be for future work, including: the weaker areas of the system which may be improved upon and ideas for new features that may be implemented in future iterations of the system. Finally, a conclusion will be drawn on the overall success of the project.

6.1 – Evaluation of the Project's Success

In order to effectively evaluate the project, one must be able to measure its success by applying metrics to the evidence generated throughout the testing of the system. Four key factors will be considered throughout this section in order to evaluate the success of the project: completeness, acceptability, productivity and stability. This section of the report will, where possible, define and use some metrics to find the extent to which each success criterion has been met.

The completeness of the project can be measured based on the results obtained while testing the functional and non-functional requirements through both the black-box testing and the User Acceptance Testing of the system as discussed in Sections 5.2.2 and 5.2.3, respectively. One such way to determine the success of this criterion is through the Requirements Completion Ratio shown in Equation 2 (Ninoslav Slavek, 2013).

$$RCR = \frac{Satisfied\_reqs}{Planned\_reqs} * 100\%$$

Equation 2 - Equation to measure the Requirements Completion Ratio as a percentage

The test cases for all functional and non-functional requirements can be found in Appendices N and O. The total number of satisfied requirements for the project was 43 out of the planned total of 47, giving the project a value of *RCR = 91.5%*. This is a considerably high ratio and as such can be considered as a positive indicator of success for this project. The requirements that were not met were: 17, 23, 24, 25. By referring to Appendices A and B one can see that none of these 4 requirements were considered as mandatory, hence the failure to implement these should not be considered as a critical defect for the system. Most importantly, the key objectives of the project as outlined in the beginning were met. These key objectives included: improving consistency in marking across large samples, providing an efficient way to ask mathematical questions in an e-Assessment setting and providing students with consistent and instantaneous feedback to help improve their learning experience.

Secondly, the acceptability of the system is a natural indicator of the successfulness of the project. As outlined in Section 5.2.3 the system was User Acceptance Tested. The average score received for the likelihood that the user would choose this system over those that are currently available was *8.4/10* with no user from the sample scoring the system at less than an 8. This is a clear testament to the acceptability of the system and as such allows acceptability to be considered as one of the measures of success of this project.

The developer's productivity throughout the project could also be considered a factor in the success of this project, especially when considering the developer's performance throughout. One such way to measure productivity can be seen in Equation 3 (Ninoslav Slavek, 2013), the project size is measured as the number of lines of code within the system while the project effort is the number of hours dedicated to the development.

$$Productivity = \frac{Project\_Size}{Project\_Effort}$$

Equation 3 - Equation measuring the developer productivity throughout the course of the project.

The number of lines of code for each of the 5 significant programming languages used can be seen in Figure 41. The main languages used throughout this system were: Java, TypeScript, SQL, HTML and CSS. The significant measurement throughout Figure 41 is the number of 'Source Code Lines' which includes the actual written code, while the 'Total Lines' column also includes both comments and blank lines.

Figure 41 - Lines of code per significant file type: .java, .css, .html and .ts

Through summing up the source code lines for the 5 languages the total number of lines of code written for the system is equal to 22,083. As this was a part-time project is can be estimated that an average of 4 hours per day were spent working on development. From the beginning of the project to the end of the final sprint 02/10/2018 - 25/03/2019 was 174 days which gives a total of 696 hours spent on development. Applying this and the number of lines of code to in the system to equation 3 gives a *Productivity = 31.74loc/h*. This is a high productivity rate with the developer averaging one line of code every 2 minutes throughout the entire course of the development.

The final success criterion to discuss is the stability of the code. Stability can be generated in code by ensuring a high code coverage throughout development. If a code-base is well covered then it is easy to determine whether the code still works after a change has been made, by simply running the tests again. If the tests pass then nothing has been broken - if anything fails then something has. This helps to ensure stability as it makes it easier to identify regression in the code. As identified in Section 5.2.1 the code coverage at the end of this project and indeed throughout most of the project was *99%.* As a result for future iterations of the project it will be easy for the developer to identify whether their new changes have regressed the system in any way.

6.2 – Evaluation of the technology used

This section of the report will provide a critical evaluation of the technology used in the development of the Assessment Assistant system. As outlined in Section 2.1 the system is a web-application with a backend component to perform the system's logic and communicate to the database, alongside a fronted component to serve content to the end user in an elegant and responsive user interface.

The backend component was written in Java and makes use of the Spring-Boot framework for configuration purposes. Data was persisted in a MySQL (Oracle, 2019) database and the frontend component was developed with Angular using the Angular CLI.

As a whole the technology used throughout the development was very suitable to the needs of the system. Using java and spring-boot for the backend application provided many useful ways to bootstrap certain functionality of the system, such as security and database connections which allowed the developer more time to focus on the functionality that was specific to the project's requirements. In addition, the use of GitLab for Version Control and TeamCity for Continuous Integration has proved extremely useful on the project management side.

There was however one language used throughout development that could have been improved upon, which is the use of MySQL for creating the database. So far the system has been populated and tested using a small sample of data, and even with this small sample it is evident that the calls to the database in some cases takes a significantly long time to respond. Hence if this application was ever to become regularly used across multiple universities the increased amount of data within the database may lead to extremely long waiting periods when making calls. Instead the use of MonogDB (MongoDB, Inc, 2019) could provide a solution to this as MongoDB is known to be significantly faster than MySQL. Table 11 outlines the speeds with which both MongoDB and MySQL can perform the same large queries (Shah, 2017).

Query	MongoDB	MySQL
	Time Taken (s)	
Select	3.8	34.47
Update	0.9	49.64
Insert	633	1914

Table 11 - The performance of MongoDB vs. MySQL when querying 1,000,000 rows of data.

The Angular CLI was undoubtedly the most instrumental technology in developer productivity throughout the development of this system. The use of development mode to instantly update changes to the frontend code significantly reduced the amount of time taken to implement and test them.

The majority of the technologies used throughout the development of the project proved to be extremely suitable in implementing the proposed solution. While MySQL has been identified as a weakness of the system it must also be said that its ease of implementation and maintenance were major advantages of using it.

6.3 – Suggestions for further work

This section will describe some of the opportunities that there are within the system for further work. Undoubtedly the most important functionality that could be added to the system is that of the requirements that were not met within the development of the system, in particular the addition of flow-chart type questions. The following examples are ways in which the system could be additionally improved: tailored grades, improved security, browser compatibility and migrating the database to MongoDB. While the benefits of migrating to MongoDB have been discussed in the previous section, the others will now be discussed.

Flow-chart type questions would be a significant addition to the system and provide assessors with a much greater range of question possibilities than is currently available. One possible implementation of this would be through the use of the ngx-graph (Georgiev, 2017) plugin for Angular. The demo for this can be seen at https://swimlane.github.io/ngx-graph. Given the use of Angular throughout the development of this project this would tie in naturally to the systems design. The charts built could be stored as map, converted in to a sequential stream of bytes and then stored in the database.

Another option for further work is the ability for a lecturer to tailor the grades that they want to give out with respect to their module. Some lecturers may prefer the use of alphabetic grading while others may prefer to number their grades. To implement this a form on the frontend could be used, where lecturers could enter the grades they want to be allowed within the module and the boundaries for each.

Currently the application only offers basic security throughout. When a user logs in their credentials are stored in base64 and passed around the angular application as a token to be used as a header in all requests made to the backend. This is not entirely safe as this ‘token’ could in theory be used to log in at any time by any user if intercepted in some way. The use of JSON Web Tokens (JWT) (Auth0, 2019) would be one way to solve this issue. JWT does not store the user’s credentials and instead stored data that the system has agreed will allow the user access for a period of time. Given the fact that JWT is commonly used with spring security and as such has many libraries in place to help with the bootstrapping of this.

The final and perhaps most important area in which further work could be done is to improve the browser compatibility of the system. Due to the fact that Angular 6 was used throughout the development of the system the JavaScript code is extremely new and up to date. One would assume that this is beneficial to an application in that the developer can avail of all the new features of JavaScript. However, there is one major drawback to this. The compilation of new JavaScript requires an up-to-date JavaScript engine which a lot of browsers simply don’t have. As browser compatibility was not in the scope of this project it is currently optimised for Google Chrome v72 and above. In order to make this application perfectly functional on all browsers one could use Babel.js (Babel, 2019). Babel.js is a JavaScript compiler that takes new JavaScript code and makes it backwards compatible with old JavaScript engines. The developer can create pipelines to send their code through for every specific version of JavaScript that they need their code to be backwards compatible with and build versions to deploy for use with the browsers which require them.

6.4 - Conclusion

The system's thorough planning and design has contributed to a solution that has adequately achieved its goals. The use of iterative and incremental development throughout has allowed for a system to be built out that satisfies the requirement for a system that reduces the time spent by lecturers on the more time-consuming aspects of marking and has allowed lecturers to deliver fast and effective feedback to aid in student's learning.

Through the large percentage of implemented requirements, high level of productivity throughout, stability of the system and its demonstrated acceptance by the end user it can reasonably be concluded that this project has been a success.

Bibliography

Agile Alliance, 2019. *Agile 101.* [Online]
Available at: https://www.agilealliance.org/agile101/
[Accessed 27 March 2019].

al, A. P. d. M. e., 2002. Marking consistency over time. *Assessment and Qualifications Alliance,* 67(1), pp. 79-87.

Alfeche, R. R. a. K., 1997. *Requirements Engineering A good practice guide.* s.l.:John Wiley and Sons.

Alpert, E. E. a. S., 2019. *Katex - The fastest math typesetting library for the web..* [Online]
Available at: The fastest math typesetting library for the web
[Accessed 21 April 2019].

Anderson, N., 2017. *CSC7052 Databases Lecture Notes,* Belfast: Queen's University Belfast .

Angular, 2010-2019. *Angular.* [Online]
Available at: https://angular.io/
[Accessed 27 March 2019].

Anne Pinot de Moira et al, 2002. Marking Consistency over Time. *Assessment and Qualifications Alliance,* 67(1), pp. 79-87.

Auth0, 2019. *JWT.* [Online]
Available at: https://jwt.io/
[Accessed 25 April 2019].

Azevedo, J. M., 2015. *e-Assessment in Mathematics Courses with Multiple-choice Questions Tests.* Porto, Portugal, ISCAP, Polytechnic Institute of Porto.

Babel, 2019. *Babel.js.* [Online]
Available at: https://babeljs.io/
[Accessed 25 April 2019].

Bootstrap Team, 2019. *Bootstrap.* [Online]
Available at: https://ng-bootstrap.github.io/#/home
[Accessed 2 April 2019].

Bootstrap, 2019. *Bootstrap.* [Online]
Available at: https://getbootstrap.com/docs/4.1/getting-started/introduction/
[Accessed 2 April 2019].

Brackett, J., 1990. *Software Requirements Technical Report,* Software Engineering Institute, Camegie Mellon University, USA: SEI-CM-19-1.2.

Campbell, A., 2005. Application of ICT and rubrics to the assessment process where professional judgement is involved: the features of an e-marking tool. *Assessment & Evaluation in Higher Education,* 30(5), pp. 529-537.

Cook, J. &. J., 2010. *Getting Started with e-Assessment,* Bath: University of Bath.

Erich Gamma, R. H. R. J. J. V., 1994. *Design Patterns: Elements of Reusable Object-Oriented Software.* 1 ed. s.l.:Addison Wesley.

Euler, L., 1748. Chapter 8: On transcending quantities arising from the circle. In: I. Bruce, ed. *Introduction to the Analysis of the Infinite.* s.l.:s.n., p. 214.

Georgiev, M., 2017. *ngx-graph,* Berlin: s.n.

Gradescope, 2019. *Gradescope Help Center Assignment Workflow.* [Online]
Available at: https://www.gradescope.com/help#help-center-item-assignment-types
[Accessed 26 March 2019].

H2, 2019. *H2 Database.* [Online]
Available at: https://www.h2database.com/html/main.html
[Accessed 26 April 2019].

Haswell, R. H., 1983. Minimal Marking. *College English,* 45(6), pp. 600-604.

Hokstad, V., 2008. *Why coupling is always bad / Cohesion vs. coupling.* [Online]
Available at: https://hokstad.com/why-coupling-is-always-bad-cohesion-vs-coupling
[Accessed 9 April 2019].

Holler, J., 2007. *org.springframework.stereotype Annotation Type Service.* [Online]
Available at: https://docs.spring.io/spring/docs/3.0.x/javadoc-api/org/springframework/stereotype/Service.html
[Accessed 13 April 2019].

ISO, 2017. *Systems and software engineering — Vocabulary.* [Online]
Available at: https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:24765:ed-2:v1:en
[Accessed 22 April 2019].

Jet Brains, 2018. *Terms of Use.* [Online]
Available at: https://www.jetbrains.com/company/useterms.html
[Accessed 22 April 2019].

Juergen Hoeller, M. F., 2007. *Annotation Type Autowired.* [Online]
Available at: https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/beans/factory/annotation/Autowired.html
[Accessed 12 April 2019].

Kent Beck et al, 2001. *Agile Manifesto.* [Online]
Available at: http://agilemanifesto.org/principles.html
[Accessed 28 March 2019].

Knuth, D., 1978. *TeX.* s.l.:s.n.

Kumar, M. K. a. N., 2013. COMPARISON OF SIX PRIORITIZATION TECHNIQUES FOR SOFTWARE REQUIREMENTS. *Journal of Global Research in Computer Science,* 4(1), pp. 38-43.

Larman, C., June 2003. *Iterative and Incremental Development: A Brief History,* s.l.: s.n.

MongoDB, Inc, 2019. *MongoDB.* [Online]
Available at: https://www.mongodb.com/

ngBootstrap Team, 2019. *ngBootstrap.* [Online]
Available at: https://ng-bootstrap.github.io/#/home
[Accessed 2 April 2019].

ngBootstrap Team, 2019. *ngBootstrap.* [Online]
Available at: https://ng-bootstrap.github.io/#/home
[Accessed 2 April 2019].

Ninoslav Slavek, D. B. K. N., 2013. Critical Measures of Success For a Software Project. *Tehnicki vjesnik,* 20(6), pp. 1119-1127.

Oracle, 2001. *Core J2EE Patterns - Data Access Object.* [Online]
Available at: https://www.oracle.com/technetwork/java/dataaccessobject-138824.html
[Accessed 12 April 2019].

Oracle, 2019. *Java SE Technologies - Database.* [Online]
Available at: https://www.oracle.com/technetwork/java/javase/jdbc/index.html
[Accessed 10 April 2019].

Oracle, 2019. *MySQL.* [Online]
Available at: https://www.mysql.com/
[Accessed 27 March 2019].

Pivotal Software, 2004. *Spring Security.* [Online]
Available at: https://spring.io/projects/spring-security
[Accessed 13 April 2019].

Pivotal, 2019. *Spring-Boot.* [Online]
Available at: https://spring.io/projects/spring-boot
[Accessed 27 March 2019].

Questionmark, 2019. *Getting started with Perception.* [Online]
Available at: https://www.questionmark.com/content/questionmark-perception-5-getting-started-guide
[Accessed 26 March 2019].

Questionmark, 2019. *Getting started with Perception.* [Online]
Available at: https://www.questionmark.com/content/questionmark-perception-5-getting-started-guide

Risberg, T., 2007. *Class BeanPropertyRowMapper<T>.* [Online]
Available at: https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/jdbc/core/BeanPropertyRowMapper.html
[Accessed 17 April 2019].

Rod Johnson, J. H., 2006. *Annotation Type Repository.* [Online]
Available at: https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/stereotype/Repository.html
[Accessed 12 April 2019].

RubyGarage, 2018. *What's the Difference Between Single-Page and Multi-Page Apps.* [Online]
Available at: https://rubygarage.org/blog/single-page-app-vs-multi-page-app
[Accessed 9 April 2019].

Sampaleanu, C., 2005. *Transactional.* [Online]
Available at: https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/transaction/annotation/Transactional.html
[Accessed 17 April 2019].

Scrum.org Team, 2019. *Scrum.org.* [Online]
Available at: https://www.scrum.org/
[Accessed 7 April 2019].

Shah, M., 2017. *MongoDB vs. MySQL.* [Online]
Available at: https://dzone.com/articles/comparing-mongodb-amp-mysql

Spring, 2019. *Spring Data REST Changelog.* [Online]
Available at: https://docs.spring.io/spring-data/rest/docs/current/changelog.txt
[Accessed 10 April 2019].

Technopedia, 2019. *Iterative and Incremental Development.* [Online]
Available at: https://www.techopedia.com/definition/25895/iterative-and-incremental-development
[Accessed 27 March 2019].

Thymeleaf, 2018. *Thymeleaf.* [Online]
Available at: https://www.thymeleaf.org/
[Accessed 27 March 2019].

Wiggins, G., 2012. Seven Keys to Effective Feedback. *Eductional Leadership,* 70(1), pp. 10-16.

Appendices

Appendix A - Functional Requirements

	Requirement	Priority
User		
1.	The system must allow users to change the password for their account.	Desirable
2.	The system must allow users to log in.	Mandatory
3.	The system must allow users to logout.	Mandatory
Student, Lecturer and Teaching Assistant		
4.	The system must allow users to contact the admins.	Inessential
Student		
5.	The system must allow students to communicate with their tutors.	Inessential
6.	The system must allow students to access their modules.	Mandatory
7.	The system must allow students to take tests.	Mandatory
8.	The system must allow students to take practice tests.	Desirable
9.	The system must allow students to answer questions set in tests.	Mandatory
10.	The system must ensure feedback on practice tests to be automatic.	Desirable
11.	The system must ensure a student's feedback is only available to them.	Mandatory
12.	The system must allow tutors to provide students information on their performance per question.	Inessential
Lecturer		
13.	The system must allow lecturers to communicate with users associated to their modules.	Desirable
14.	The system must allow lecturers to create new modules.	Mandatory
15.	The system must allow lecturers to associate students and teaching assistants to their modules.	Inessential
16.	The system must allow lecturers to access their modules.	Mandatory
17.	The system must allow lecturers to specify whether their assistants can create tests or just mark.	Inessential
18.	The system must allow lecturers to create tests.	Mandatory
19.	The system must allow lecturers to create practice tests.	Desirable
20.	The system must allow lecturers to specify how long their tests are live for.	Mandatory
21.	The system must allow a lecturer to ask text input-based questions.	Mandatory
22.	The system must allow a lecturer to ask multiple choice questions.	Mandatory
23.	The system must allow a lecturer to ask truth-table questions.	Desirable
24.	The system must allow a lecturer to ask flowchart-based questions.	Desirable
25.	The system must allow a lecturer to ask predicate-based questions.	Desirable
26.	The system must allow a lecturer to ask mathematical questions.	Desirable
27.	The system must allow a lecturer to ask insert-the-word questions.	Mandatory
28.	The system must allow lecturers to choose whether the system automatically, semi-automatically or manually facilitates the marking of individual questions in the exam.	Inessential
29.	The system should give x marks to all identical solutions to what has been given x marks by a lecturer.	Desirable
30.	The system should allow lecturers to override the systems information that may be used to assess students' answers.	Desirable
31.	The system must separate questions among lecturers to ensure no clash if rules are changed.	Desirable
32.	The system must allow the lecturers to easily review and amend marks that have been allocated by the system.	Mandatory
33.	The system must allow lecturers to issue test feedback to their students.	Desirable
34.	The system must allow the lecturer to see statistics on how students performed in tests.	Desirable

35.	The system must allow the lecturer to see statistics on how the class performed in tests.	Desirable
Teaching Assistant		
36.	The system must allow teaching assistants to communicate with the lecturer.	Inessential
37.	The system must allow teaching assistants to access modules they are assisting with.	Mandatory
38.	The system must allow teaching assistants to perform actions the lecturers has permitted them to.	Mandatory
Admin		
39.	The system must allow admins to respond to messages from other users.	Inessential
40.	The system must allow admins to approve the addition of new modules.	Desirable

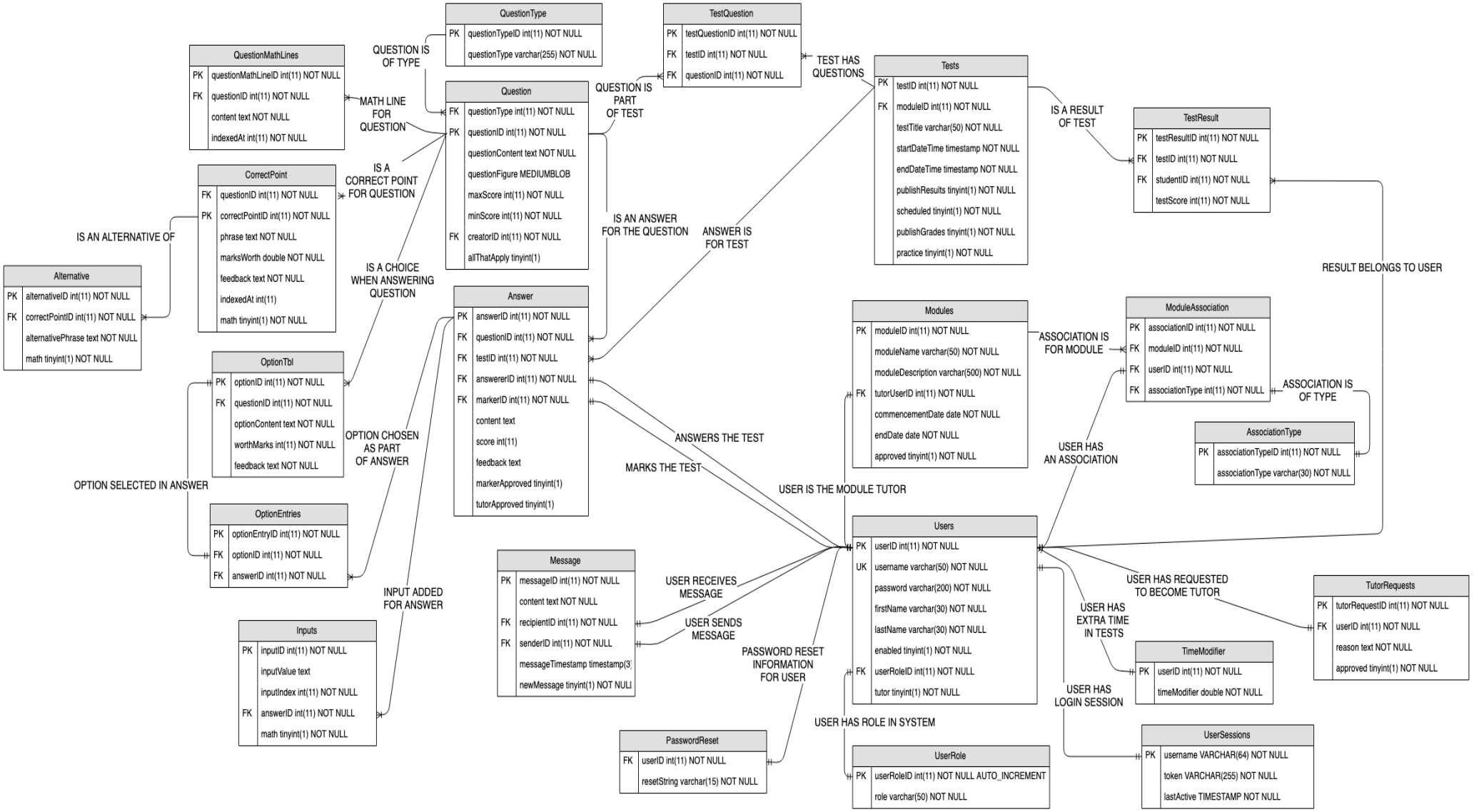
Appendix B - Non-functional Requirements

	Requirement	Priority
1.	The system must store information securely.	Mandatory
2.	The front-end must have a consistent user-friendly UI.	Desirable
3.	The application must be accessible publicly on the world wide web.	Mandatory
4.	The application must be easy to use for new users.	Desirable
5.	The application should make appropriate use of tooltips.	Desirable
6.	The user should be able to move around the application with ease.	Desirable
7.	Execution of processes within the system must take no longer than 10 seconds.	Desirable

Appendix C - Error Conditions

	Condition
1.	The system must validate all front-end user input and provide feedback.
2.	The system must elegantly handle all exceptions in the backend.
3.	The system must ensure no unauthorized user can gain access to the system.

Appendix D – Database Relational Model

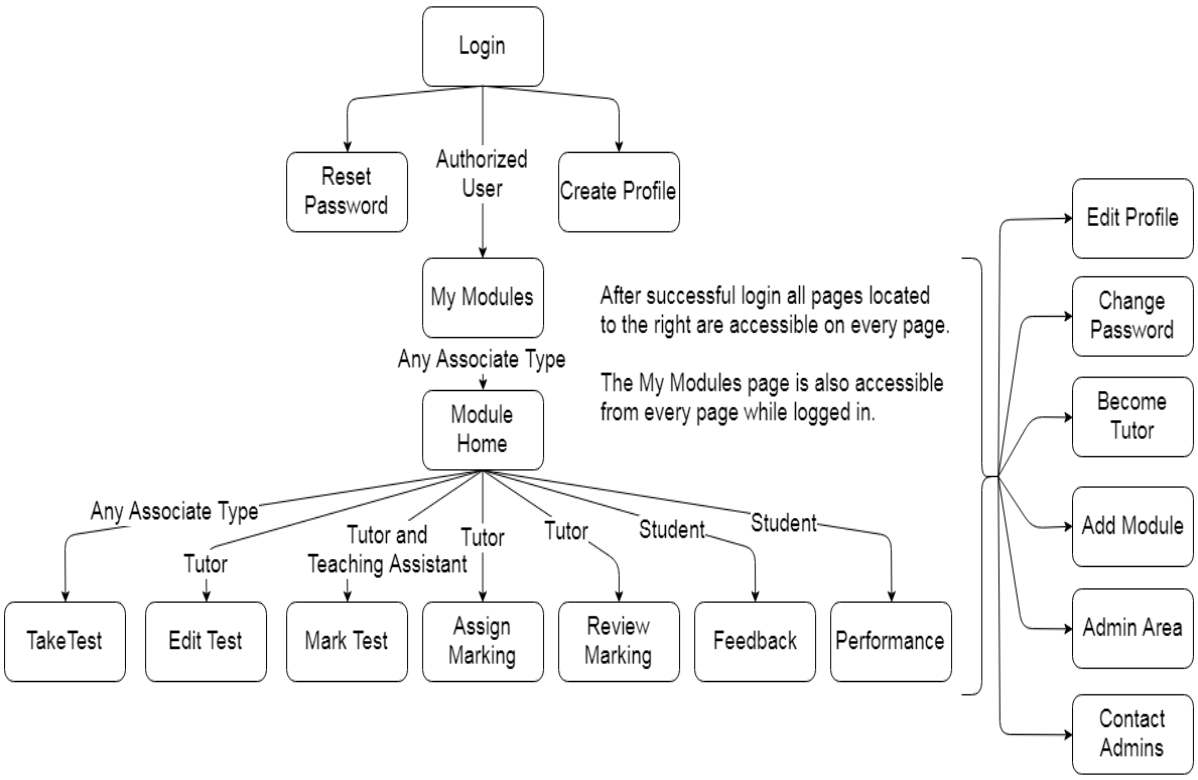


Appendix E – Rest Endpoints

Rest URL	Parameters	Return	Type
/main (MainController)			
/getUser	Principal	User	n/a
/login	Principal	TokenPojo	n/a
/logout	Principal	void	n/a
/tests (TestController)			
/addTest	Principal, Tests	Tests	POST
/submitTest	Principal, List<QuestionAndAnswer>	Boolean	POST
/editTest	Principal, Tests	Tests	POST
/getByTestID	Principal, Long	Tests	GET
/getQuestionsStudent	Principal, Long	List<QuestionAndAnswer>	GET
/getPerformance	Principal, Long	Performance	GET
/getFeedback	Principal, Long	Performance	GET
/getAnsweredTests	Principal	Set<Integer>	GET
/getQuestionsByTestIDTutorView	Principal, Long	List<TutorQuestionPojo>	GET
/getOldQuestions	Principal, Long	List<TutorQuestionPojo>	GET
/getQuestionTypes	n/a	List<QuestionType>	GET
/editQuestion	TutorQuestionPojo, Principal	TutorQuestionPojo	POST
/addQuestion	TutorQuestionPojo, Principal	TutorQuestionPojo	POST
/addExistingQuestion	Long, Long, Prinicpal	TestQuestion	GET
/duplicateQuestion	Long, Principal	Boolean	GET
/removeQuestionFromTest	Long, Long, Principal	Boolean	DELETE
/removeCorrectPoint	Long, Principal	Boolean	DELETE
/removeQuestionMathLine	Long, Principal	Boolean	DELETE
/removeOption	Long, Principal	Boolean	DELETE
/removeAlternative	Long, Principal	Boolean	DELETE
/deleteTest	Long, Principal	Boolean	DELETE
/scheduleTest	Long, Principal	Boolean	GET
/marking (MarkingController)			
/getMarkersData	Principal, Long	MarkerWithChart	GET
/reassignAnswers	Principal, Long, List<MarkerAndReassigned>	Boolean	POST
/editAnswer	Principal, Answer	Answer	POST
/editScore	Principal, Answer	Answer	POST
/editFeedback	Principal, Answer	Answer	POST
/addCorrectPoint	Principal, CorrectPoint, Long	Boolean	POST
/addAlternative	Principal, Alternative, Long	Boolean	POST
/approve	Principal, Long	Answer	GET
/getScriptsMarker	Principal, Long	List<AnswerData>	GET
/getScriptsTutor	Principal, Long	List<AnswerData>	GET
/publishGrades	Principal, Long	Boolean	GET
/publishResults	Principal, Long	Boolean	GET
/getCorrectPoints	Principal, Long, Long	List<CorrectPoint>	GET
/removeCorrectPoint	Principal, Long, Long	Boolean	DELETE
/getResultChart	Principal, Long	ResultChartPojo	GET
/getQuestionsResultChart	Principal, Long	List<ResultChartPojo>	GET
/removeAlternative	Principal, Long, Long	Boolean	DELETE
/modules (ModuleController)			
/getByModuleID	Long	Module	GET
/getModuleAndTutor	Long	ModuleWithTutor	GET
/getMyModulesWithTutors	Principal	List<ModuleWithTutor>	GET
/addModule	ModulePojo, Principal	void	POST
/addAssociations	Long, List<Associate>, Principal	void	POST
/getModulesPendingApproval	Principal	List<Module>	GET
/getActiveTests	Principal, Long	List<Tests>	GET
/getPracticeTests	Principal, Long	List<Tests>	GET

/getActiveResults	Principal, Long	List<TestAndGrade>	GET
/getScheduledTests	Principal, Long	List<Tests>	GET
/getTestDrafts	Principal, Long	List<Tests>	GET
/getReviewMarking	Principal, Long	List<TestMarking>	GET
/getMarking	Principal, Long	List<TestMarking>	GET
/getPerformance	Principal, Long	List<Performance>	GET
/getModuleAssociation	Principal, Long	Long	GET
/getModuleRequests	Principal	List<ModuleRequestPojo>	GET
/approveModuleRequest	Long, Principal	void	GET
/rejectModuleRequest	Long, Principal	void	GET
/getAssociates	Long, Principal	List<Associate>	GET
/removeAssociate	String, Long, Principal	void	GET
/user (UserController)			
/changePassword	ChangePassword, Principal	UserSession	POST
/resetPassword	String, String, String	Boolean	GET
/requestResetPassword	String	Boolean	GET
/submitTutorRequest	TutorRequests, Principal	TutorRequests	POST
/getTutorRequest	Principal	TutorRequests	GET
/getUsernames	Principal, String	Boolean	GET
/createProfile	User	User	POST
/editProfile	User, Principal	User	POST
/findAll	n/a	List<User>	GET
/getTutorRequests	Principal	List<TutorQuestionPojo>	GET
/approveTutorRequest	Long, Principal	void	GET
/makeAdmin	Long, Principal	void	GET
/makeTutor	Long, Principal	void	GET
/removeUser	Long, Principal	void	GET
/rejectTutorRequest	Long, Principal	void	GET
/getAdmins	n/a	List<User>	GET
/addUsers	List<User>, Principal	void	POST

Appendix F – Frontend Hierarchy



Appendix G - The pom.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://maven.apache.org/POM/4.0.0"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <artifactId>backendAA</artifactId>

    <name>backend</name>
    <description>The backend for Assessment Assistant</description>

    <parent>
        <groupId>com.pgault04</groupId>
        <artifactId>pgault04_AssessmentAssistant</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-mail</artifactId>
            <version>2.0.1.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jdbc</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <scope>runtime</scope>
        </dependency>

        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-entitymanager</artifactId>
        </dependency>
```

```
<dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-validator</artifactId>
            <version>4.1.0.Final</version>
        </dependency>

        <dependency>
            <groupId>com.microsoft.sqlserver</groupId>
            <artifactId>mssql-jdbc</artifactId>
            <scope>runtime</scope>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
            <scope>test</scope>
        </dependency>


        <dependency>
            <groupId>com.pgault04</groupId>
            <artifactId>frontendAA</artifactId>
            <version>${project.version}</version>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger-ui</artifactId>
            <version>2.9.2</version>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

Appendix H - *prepareQuestionGeneral method in TestService class*

```
/*
 * Generic preparation for all question types to be output to tutor
 */
private List<CorrectPoint> prepareQuestionGeneral(TutorQuestionPojo questionData, String username, Boolean update) throws SQLException {
    Question question = questionData.getQuestion();
    if (question.getQuestionID() == null || question.getQuestionID() < 1) {
        question.setQuestionID(-1L);
    }
    if (questionData.getBase64() != null) {
        question.setQuestionFigure(BlobUtil.baseToBlob(questionData.getBase64()));
    }
    List<CorrectPoint> correctPoints = questionData.getCorrectPoints();
    User user = userRepo.selectByUsername(username);
    question.setCreatorID(user.getUserID());
    questionData.setQuestion(questionRepo.insert(question));
    questionData.setMathLines(addMathLines(questionData.getQuestion().getQuestionID(), questionData.getMathLines()));
    if (!update) {
        testQuestionRepo.insert(new TestQuestion(questionData.getTestID(), questionRepo.insert(question).getQuestionID()));
    }
    return correctPoints;
}
```

Appendix I - The Auto-marking methods in TestService class

```
/*
 * Auto-marks multiples choice questions based on score and feedback given to options in marking phase
 */
private void autoMarkMultipleChoice(QuestionAndAnswer questionAndAnswer) {
   Question question = questionRepo.selectByID(questionAndAnswer.getAnswer().getQuestionID());
   List<Option> options = findOptions(questionAndAnswer.getAnswer().getQuestionID());
   options.sort(Collections.reverseOrder(Comparator.comparingDouble(Option::getWorthMarks)));
   prepareAnswerForAutoMarking(questionAndAnswer);
   if (options.size() > 0) {
      for (Option o : options) {
         for (OptionEntries oe : questionAndAnswer.getOptionEntries()) {
            if (o.getOptionID().equals(oe.getOptionID())) {
               addMarkAndFeedbackForCorrectOption(o, questionAndAnswer, oe);
            }
         }
      }
      validateScore(questionAndAnswer.getAnswer(), question);
   }
}

/*
 * When an option is found in one of the users entries then the mark and feedback for the option is added to their answer
 */
private void addMarkAndFeedbackForCorrectOption(Option o, QuestionAndAnswer questionAndAnswer,
OptionEntries oe) {
   oe.setAnswerID(questionAndAnswer.getAnswer().getAnswerID());
   optionEntriesRepo.insert(oe);
   questionAndAnswer.getAnswer().setScore(questionAndAnswer.getAnswer().getScore() + o.getWorthMarks());
   questionAndAnswer.getAnswer().setFeedback(questionAndAnswer.getAnswer().getFeedback() + o.getFeedback()
+ "\n");
   questionAndAnswer.getAnswer().setMarkerApproved(1);
}
```

```
/ * Auto-marking method for all questions that use correct points
 * Calls relevant methods for their specific question types
 */
void autoMarkCorrectPoints(QuestionAndAnswer questionAndAnswer) {
   Question question = questionRepo.selectByID(questionAndAnswer.getAnswer().getQuestionID());
   List<CorrectPoint> correctPoints = findCorrectPoints(questionAndAnswer.getAnswer().getQuestionID());
   correctPoints.sort(Collections.reverseOrder(Comparator.comparingDouble(CorrectPoint::getMarksWorth)));
   prepareAnswerForAutoMarking(questionAndAnswer);
   if (question.getQuestionType().equals(QuestionType.INSERT_THE_WORD)
                                                    || question.getQuestionType().equals(QuestionType.TEXT_MATH)) {
      insertInputs(questionAndAnswer);
      automarkCorrectPointsForInputs(questionAndAnswer, question, correctPoints);
   } else {
      autoMarkCorrectPointsForTextBased(questionAndAnswer, correctPoints);
   }
   validateScore(questionAndAnswer.getAnswer(), question);
}

/*
 * Auto-marking method for questions that use inputs
 * Calls submethods for automarking insert the word and math type inputs
 */
private void automarkCorrectPointsForInputs(QuestionAndAnswer questionAndAnswer, Question question,
List<CorrectPoint> correctPoints) {
   for (CorrectPoint c : correctPoints) {
      for (Inputs i : questionAndAnswer.getInputs()) {
         if (question.getQuestionType().equals(QuestionType.INSERT_THE_WORD)) {
            autoMarkCorrectPointsForInsertTheWord(questionAndAnswer, c, i);
         } else {
            if (autoMarkCorrectPointsForTextMath(questionAndAnswer, c, i)) break;
         }
      }
   }
}

/*
 * Gets answer ready for auto-marking - i.e. removes prior feedback, score and trims the content to ensure random spaces
to affect the content check
 */
private void prepareAnswerForAutoMarking(QuestionAndAnswer questionAndAnswer) {
   questionAndAnswer.getAnswer().setScore(0);
   questionAndAnswer.getAnswer().setFeedback("");
   if (questionAndAnswer.getAnswer().getContent() != null) {
      questionAndAnswer.getAnswer().setContent(questionAndAnswer.getAnswer().getContent().trim());
   }
}

/*
 * Performs auto-marking for text based questions
 * Checks if answer contains a correct point and then if not check the correct points alternatives
 */
private void autoMarkCorrectPointsForTextBased(QuestionAndAnswer questionAndAnswer, List<CorrectPoint>
correctPoints) {
   for (CorrectPoint cp : correctPoints) {
      if (questionAndAnswer.getAnswer().getContent().toLowerCase().contains(cp.getPhrase().toLowerCase())) {
         setScoreAndFeedbackForCorrectPoint(questionAndAnswer, cp.getMarksWorth(), cp.getFeedback());
      } else {
         for (Alternative alt : alternativeRepo.selectByCorrectPointID(cp.getCorrectPointID())) {
            if (questionAndAnswer.getAnswer().getContent().toLowerCase()
                                                            .contains(alt.getAlternativePhrase().toLowerCase())) {
               setScoreAndFeedbackForCorrectPoint(questionAndAnswer, cp.getMarksWorth(), cp.getFeedback());
               break;
            }
         }
      }
   }
}
```

```
/*
 * Auto-marks text-math type questions
 * If the correct point is exactly contained the marks are given if not the alternatives are checked
 */
private boolean autoMarkCorrectPointsForTextMath(QuestionAndAnswer questionAndAnswer, CorrectPoint c,
Inputs i) {
   if (i.getInputValue().trim().equalsIgnoreCase(c.getPhrase())) {
      setScoreAndFeedbackForCorrectPoint(questionAndAnswer, c.getMarksWorth(), c.getFeedback());
      return true;
   } else {
      boolean altBroken = false;
      for (Alternative alt : alternativeRepo.selectByCorrectPointID(c.getCorrectPointID())) {
         if (i.getInputValue().contains(alt.getAlternativePhrase())) {
            setScoreAndFeedbackForCorrectPoint(questionAndAnswer, c.getMarksWorth(), c.getFeedback());
            altBroken = true;
            break;
         }
      }
      return altBroken;
   }
}

/*
 * Auto-marks insert the word type questions
 * If the correct point or one of it's alternatives are contained in the correctly indexed input then mark is given
 */
private void autoMarkCorrectPointsForInsertTheWord(QuestionAndAnswer questionAndAnswer, CorrectPoint c,
Inputs i) {
   if (i.getInputValue().equalsIgnoreCase(c.getPhrase()) && i.getInputIndex().equals(c.getIndexedAt())) {
      setScoreAndFeedbackForCorrectPoint(questionAndAnswer, c.getMarksWorth(), c.getFeedback());
   } else {
      for (Alternative alt : alternativeRepo.selectByCorrectPointID(c.getCorrectPointID())) {
         if (i.getInputValue().equalsIgnoreCase(alt.getAlternativePhrase())
                                                      && i.getInputIndex().equals(c.getIndexedAt())) {
            setScoreAndFeedbackForCorrectPoint(questionAndAnswer, c.getMarksWorth(), c.getFeedback());
            break;
         }
      }
   }
}
```

Appendix J - Manual marking methods in MarkingService

```
/**
 * Method to allow for editing score/feedback for given answer by marker/tutor
 * Performs tasks that overlap between editScore and editFeedback methods
 *
 * @param username the user
 * @param answer   the answer
 * @return the answer
 * @throws IllegalArgumentException when unauthorized user attempts
 */
public Answer editAnswer(String username, Answer answer) throws IllegalArgumentException {
   User user = userRepo.selectByUsername(username);
   Tests test = testsRepo.selectByTestID(answer.getTestID());
   Long check = modService.checkValidAssociation(username, test.getModuleID());
   if (check != null && check == AssociationType.TUTOR || user != null &&
user.getUserID().equals(answer.getMarkerID())) {
      answer = answerRepo.insert(answer);
      return answer;
   } else {
      throw new IllegalArgumentException("Must be marker or tutor to complete this action.");
   }
}
```

```
/**
 * Allows user to edit feedback provided to an answer
 * Also triggers search for identical answers to match their feedback accordingly
 *
 * @param username the user performing the manual editing
 * @param answer   the answer being manually changed
 * @return the answer after manual change
 * @throws IllegalArgumentException when user is not allowed to perform this action
 */
public Answer editFeedback(String username, Answer answer) throws IllegalArgumentException {
   answer = editAnswer(username, answer);
   List<Answer> answers = answerRepo.selectByTestID(answer.getTestID());
   List<Inputs> inputs = inputsRepo.selectByAnswerID(answer.getAnswerID());
   List<OptionEntries> optionEntries = optionEntriesRepo.selectByAnswerID(answer.getAnswerID());
   List<Answer> answersForNonTutor = new ArrayList<>();
   answers = reviseAnswerListForTeachingAssistant(username, answer, answers, answersForNonTutor);
   for (Answer a : answers) {
      checkTypeAndUpdateFeedback(answer, inputs, optionEntries, a);
   }
   return answer;
}

/*
 * Different approaches must be made to check similarity of different question types
 * This check the type and then performs the update
 */
private void checkTypeAndUpdateFeedback(Answer answer, List<Inputs> inputs, List<OptionEntries>
optionEntries, Answer a) {
   if (a.getQuestionID().equals(answer.getQuestionID())) {
      Question questionToCompare = questionRepo.selectByID(a.getQuestionID());
      if (questionToCompare.getQuestionType() == QuestionType.TEXT_BASED &&
a.getContent().equalsIgnoreCase(answer.getContent())) {
         updateFeedback(answer, a);
      } else if (questionToCompare.getQuestionType() == QuestionType.INSERT_THE_WORD ||
questionToCompare.getQuestionType() == QuestionType.TEXT_MATH) {
         int inputCounter = checkInputsForSimilarity(inputs, a);
         if (inputCounter == inputs.size()) {
            updateFeedback(answer, a);
         }
      } else if (questionToCompare.getQuestionType() == QuestionType.MULTIPLE_CHOICE) {
         updateFeedbackForMultipleChoice(answer, optionEntries, a);
      }
   }
}

/*
 * Checks if identical options selected by other users before updating feedback
 */
private void updateFeedbackForMultipleChoice(Answer answer, List<OptionEntries> optionEntries, Answer a) {
   List<Long> optionEntriesIDs = new ArrayList<>();
   List<OptionEntries> optionEntriesToCompare = optionEntriesRepo.selectByAnswerID(a.getAnswerID());
   List<Long> optionEntriesIDsToCompare = new ArrayList<>();
   for (OptionEntries o : optionEntriesToCompare) {
      optionEntriesIDsToCompare.add(o.getOptionID());
   }
   for (OptionEntries o : optionEntries) {
      optionEntriesIDs.add(o.getOptionID());
   }
   if (optionEntriesIDs.containsAll(optionEntriesIDsToCompare)) {
      updateFeedback(answer, a);
   }
}
```

```
/*
 * Performs the actual feedback update (common among all question types)
 */
private void updateFeedback(Answer answer, Answer a) {
   a.setFeedback(answer.getFeedback());
   answerRepo.insert(a);
}

/*
 * When a teaching assistant performs the edit feedback or score methods
 * the answer list affected must only include answers they are markers of
 * For tutor this is not necessary as their decision is allowed to affect all scripts
 */
private List<Answer> reviseAnswerListForTeachingAssistant(String username, Answer answer, List<Answer>
answers, List<Answer> answersForNonTutor) {
   Long check = modService.checkValidAssociation(username,
testsRepo.selectByTestID(answer.getTestID()).getModuleID());
   if (check != null && check == AssociationType.TEACHING_ASSISTANT) {
      User user = userRepo.selectByUsername(username);
      for (Answer a : answers) {
         if (a.getMarkerID().equals(user.getUserID())) {
            answersForNonTutor.add(a);
         }
      }
      answers = answersForNonTutor;
   }
   return answers;
}

/**
 * Allows marker to manually edit score for an answer they are marking
 * Check all identical submissions and updates their scores accordingly
 *
 * @param username the username of user performing action
 * @param answer   the answer having its score updated
 * @return the updated answer
 * @throws IllegalArgumentException when user is not permitted to perform such an action
 */
public Answer editScore(String username, Answer answer) throws IllegalArgumentException {
   answer = editAnswer(username, answer);
   List<OptionEntries> optionEntries = optionEntriesRepo.selectByAnswerID(answer.getAnswerID());
   List<Answer> answers = answerRepo.selectByTestID(answer.getTestID());
   List<Answer> answersForNonTutor = new ArrayList<>();
   answers = reviseAnswerListForTeachingAssistant(username, answer, answers, answersForNonTutor);
   List<Inputs> inputs = inputsRepo.selectByAnswerID(answer.getAnswerID());
   for (Answer a : answers) {
      checkTypeAndUpdateScore(answer, optionEntries, inputs, a);
   }
   return answer;
}
```

```
/*
 * Similar to update feedback
 * different question types require different checks
 * before knowing whether to update score for identicals
 */
private void checkTypeAndUpdateScore(Answer answer, List<OptionEntries> optionEntries, List<Inputs> inputs,
Answer a) {
   if (a.getQuestionID().equals(answer.getQuestionID())) {
      Question questionToCompare = questionRepo.selectByID(a.getQuestionID());
      if (questionToCompare.getQuestionType() == QuestionType.TEXT_BASED
                                                   && a.getContent().equalsIgnoreCase(answer.getContent())) {
         updateScore(answer, a);
      } else if (questionToCompare.getQuestionType() == QuestionType.INSERT_THE_WORD
                                          || questionToCompare.getQuestionType() == QuestionType.TEXT_MATH) {
         int inputCounter = checkInputsForSimilarity(inputs, a);
         if (inputCounter == inputs.size()) {
            updateScore(answer, a);
         }
      } else if (questionToCompare.getQuestionType() == QuestionType.MULTIPLE_CHOICE) {
         updateScoreForMultipleChoice(answer, optionEntries, a);
      }
   }
}

/*
 * Checks if identical options selected by other users before updating score
 */
private void updateScoreForMultipleChoice(Answer answer, List<OptionEntries> optionEntries, Answer a) {
   List<Long> optionEntriesIDs = new LinkedList<>();
   for (OptionEntries o : optionEntries) {
      optionEntriesIDs.add(o.getOptionID());
   }
   List<OptionEntries> optionEntriesToCompare = optionEntriesRepo.selectByAnswerID(a.getAnswerID());
   List<Long> optionEntriesIDsToCompare = new LinkedList<>();
   for (OptionEntries o : optionEntriesToCompare) {
      optionEntriesIDsToCompare.add(o.getOptionID());
   }
   if (optionEntriesIDs.containsAll(optionEntriesIDsToCompare)) {
      updateScore(answer, a);
   }
}

/*
 * Performs the actual score update (common among all question types)
 */
private void updateScore(Answer answer, Answer a) {
   a.setScore(answer.getScore());
   answerRepo.insert(a);
}

/*
 * Checks inputs for answers are identical to answer being checked against
 * If every element in the list matches the the size of list should be returned
 */
private int checkInputsForSimilarity(List<Inputs> inputs, Answer a) {
   List<Inputs> inputToCompare = inputsRepo.selectByAnswerID(a.getAnswerID());
   return (int) inputs.stream().
         filter(i -> inputToCompare.stream().anyMatch(iToCompare ->
               i.getInputIndex().equals(iToCompare.getInputIndex())
                     && i.getInputValue().equalsIgnoreCase(iToCompare.getInputValue()))).count();
}
```

Appendix K - The method in adminArea.component.ts to convert csv to array of Users

```
/**
 * Processes users from CSV and converts them to user objects @param csv
 */
readUsers(csv: any) {
  this.userFile = csv.target.files[0];
  let fileReader = new FileReader();
  let stopPush;
  fileReader.readAsText(this.userFile);

  fileReader.onload = () => {
    this.fileSuccess = false;
    this.fileError = false;
    this.fileErrorMessage = "";
    // Split on rows
    let data = fileReader.result.toString().split(/[\r\n]/);

    // Split header on commas
    let headers = data[0].split(',');
    let spliceArray = [];
    for (let i = 0; i < data.length; i++) {
      if (data[i] == "") {
        spliceArray.push([i]);
      }
    }
    for (let i = 0; i < spliceArray.length; i++) {
      data.splice(spliceArray[i] - i, 1);
    }
    for (let i = 1; i < data.length; i++) {
      stopPush = false;
      // split content based on comma
      let subData = data[i].split(',');
      let userToAdd = new User();
      if (subData.length === headers.length) {
        for (let j = 0; j < headers.length; j++) {
          if (headers[j] == "Email") {
            userToAdd.username = subData[j];
          } else if (headers[j] == "Tutor") {
            userToAdd.tutor = subData[j] == 'Y' ? 1 : 0;
          } else if (headers[j] == "FirstName") {
            userToAdd.firstName = subData[j];
          } else if (headers[j] == "LastName") {
            userToAdd.lastName = subData[j];
          }
        }
        if (!userToAdd.tutor && !userToAdd.username && !userToAdd.firstName && !userToAdd.lastName) {
          stopPush = true;
        } else if (userToAdd.tutor > 1 || userToAdd.tutor < 0 || !userToAdd.firstName || !userToAdd.lastName
                                                                                    || !userToAdd.username) {
          this.fileError = true;
          this.fileErrorMessage = "One or more cells have been left empty.";
          this.userInput.nativeElement.value = null;
          return;
        }
      } else {
        this.fileError = true;
        this.fileErrorMessage = "Column count does not match headings.";
        this.userInput.nativeElement.value = null;
        return;
      }
      if (!stopPush) {
        this.usersToAdd.push(userToAdd);
      }
    }
    this.fileSuccess = true;
  };
}
```

Appendix L - Add users method in UserService to add the newly added users to system

```
/**
 * Allows admins to add users to the system
 * Added on the front end via CSV this is converted to a list of users to be inserted in to the database
 * Asynchronously notifies each user that they have been added
 *
 * @param users    the users to add
 * @param username the user name who is adding them
 */
public void addUsers(List<User> users, String username) {
   User admin = userRepo.selectByUsername(username);
   if (admin.getUserRoleID().equals(UserRole.ROLE_ADMIN)) {
      for (User u : users) {
         User user = userRepo.selectByUsername(u.getUsername());
         if (user == null) {
            String password = PasswordUtil.generateRandomString();
            user = userRepo.insert(new User(u.getUsername(),
                                    PasswordUtil.encrypt(password),
                                    u.getFirstName(), u.getLastName(), 0, UserRole.ROLE_USER, 0));

            passwordResetRepo.insert(new PasswordReset(user.getUserID(), PasswordUtil.generateRandomString()));
            userSessionRepo.insert(new UserSession(user.getUsername(),
                                       new String(Base64.getEncoder().encode((user.getUsername() + ":"
                                                                              + password).getBytes())),
                                       new Timestamp(System.currentTimeMillis())));

            emailSender.sendNewAccountMessageFromSystemToUser(user, password, username);
         }
      }
   }
}
```

Appendix M - The output on front end of rendering math input for a Question Math Line

Appendix N - Functional Requirements Test Cases

	Requirement	Action	Expected Result	Result
1.	The system must allow users to change the password for their account.	While logged in navigate to the change password area and change password.	The user's password should update.	Pass
2.	The system must allow users to log in.	Log in as a user with correct credentials.	The user will be navigated to the MyModules page.	Pass
3.	The system must allow users to logout.	While logged in click the logout button from any page.	The user should be returned to the login page.	Pass
4.	The system must allow users to contact the admins.	Navigate to the contact admins area from the more options dropdown. Click on any admin's email.	The users email client should be launched with an email addressed to the selected admin.	Pass
5.	The system must allow students to communicate with their tutors.	Navigate to the MyModules page. Click on any tutor's email.	The users email client should be launched with an email addressed to the selected tutor.	Pass
6.	The system must allow students to access their modules.	Click on the link to go to module home for any modules displayed on the MyModules page.	User should be navigated to module home for the selected module.	Pass
7.	The system must allow students to take tests.	When a test is live click the link to go to the take test area.	The user should be navigated to the take test area for the chosen test.	Pass
8.	The system must allow students to take practice tests.	When a practice test is live click the link to go to the take test area.	The user should be navigated to the take test area for the chosen practice test.	Pass
9.	The system must allow students to answer questions set in tests.	When inside the take test area edit the response area for any question and click submit.	The user should be navigated to module home page and be able to see they have submitted the test.	Pass
10.	The system must ensure feedback on practice tests to be automatic.	Submit a practice test and immediately click the feedback link after being redirected.	The feedback should be visible on the new page.	Pass
11.	The system must ensure a student's feedback is only available to them.	Once feedback becomes available log on as a student who took the test and go to the grades Section.	The feedback shown should only be for the user.	Pass
12.	The system must allow tutors to provide students information on their performance per question.	Once results become available log on as a student who took the test and go to the results Section and navigate to the performance page for the test.	The performance per question should be shown to the user.	Pass
13.	The system must allow lecturers to communicate with users associated to their modules.	As the lecturer for the module go to the associates' Section and click on the button to contact a student.	The lecturer's email client should open with an email addressed to the student.	Pass

14.	The system must allow lecturers to create new modules.	As a user who has become a tutor/lecturer, go to the add module Section and add the data for a module and click submit.	The entered module should appear in the modules pending approval Section.	Pass
15.	The system must allow lecturers to associate students and teaching assistants to their modules.	In the associates Section of the module home page, download the sample csv and fill in some user data and submit.	The user should now appear in the associates Section.	Pass
16.	The system must allow lecturers to access their modules.	Click on the link to go to module home for any modules displayed on the MyModules page.	User should be navigated to module home for the selected module.	Pass
17.	The system must allow lecturers to specify whether their assistants can create tests or just mark.	n/a	n/a	Fail (Not complete)
18.	The system must allow lecturers to create tests.	As the lecturer/tutor for the module go to the add test Section and fill in the form to create an official test.	The user should be navigated to the edit test page and have the ability to add questions.	Pass
19.	The system must allow lecturers to create practice tests.	As the lecturer/tutor for the module go to the add test Section and fill in the form to create a practice test.	The user should be navigated to the edit test page and have the ability to add questions.	Pass
20.	The system must allow lecturers to specify how long their tests are live for.	As the lecturer/tutor for the module go to the add test Section and fill in the form to create any test and fill in the start and date time.	The user should be navigated to the edit test page and have the ability to add questions.	Pass
21.	The system must allow a lecturer to ask text input-based questions.	In the edit test Section add a new question and select the “text-based’ question type, fill in the data and submit.	The question should appear in the left column in the questions for the test	Pass
22.	The system must allow a lecturer to ask multiple choice questions.	In the edit test Section add a new question and select the “multiple-choice” question type, fill in the data and submit.	The question should appear in the left column in the questions for the test	Pass
23.	The system must allow a lecturer to ask truth-table questions.	n/a	n/a	Fail (Not complete)
24.	The system must allow a lecturer to ask flowchart-based questions.	n/a	n/a	Fail (Not complete)
25.	The system must allow a lecturer to ask predicate-based questions.	n/a	n/a	Fail (Not complete)

26.	The system must allow a lecturer to ask mathematical questions.	In the edit test Section add a new question and select the “test-math” question type, fill in the data and submit.	The question should appear in the left column in the questions for the test	Pass
27.	The system must allow a lecturer to ask insert-the-word questions.	In the edit test Section add a new question and select the “insert-the-word” question type, fill in the data and submit.	The question should appear in the left column in the questions for the test	Pass
28.	The system must allow lecturers to choose whether the system automatically, semi-automatically or manually facilitates the marking of individual questions in the exam.	Make a test and add some questions, schedule the test to go live, log back in as a student and sit the test. Once the test has ended log back in as a tutor and go to the marking Section.	All questions should be auto-marked but the lecturer will have total control over which ones to manually override if any.	Pass
29.	The system should give x marks to all identical solutions to what has been given x marks by a lecturer.	Make a test and add some questions, schedule the test to go live, log back in as a student and sit the test repeat for another student and answer identically. Once the test has ended log back in as a tutor and go to the marking Section. Manually mark one of the student’s questions.	The answer from the other student should automatically be given the same marks.	Pass
30.	The system should allow lecturers to override the systems information that may be used to assess students’ answers.	Make a test and add some questions, schedule the test to go live, log back in as a student and sit the test. Once the test has ended log back in as a tutor and go to the marking Section. Manually mark one of the student’s questions.	The auto-marking should be overridden.	Pass
31.	The system must separate questions among lecturers to ensure no clash if rules are changed.	On the edit test page go to reuse questions.	Only questions created by the user (if any have been entered) should be seen, with none from other tutors.	Pass
32.	The system must allow the lecturers to easily review and amend marks that have been allocated by the system.	Make a test and add some questions, schedule the test to go live, log back in as a student and sit the test. Once the test has ended log back in as a tutor and go to the marking Section. Manually mark all of the student’s questions. Navigate to the review marking page for this test and edit the score of an answer.	The score for the answer should be updated.	Pass

33.	The system must allow lecturers to issue test feedback to their students.	Make a test and add some questions, schedule the test to go live, log back in as a student and sit the test. Once the test has ended log back in as a tutor and go to the marking Section. Manually mark all of the student's questions. Navigate to the review marking page for this test and click publish results or publish feedback.	The student who answered this test should now be able to see the feedback.	Pass
34.	The system must allow the lecturer to see statistics on how students performed in tests.	Make a test and add some questions, schedule the test to go live, log back in as a student and sit the test. Once the test has ended log back in as a tutor and go to the marking Section. Manually mark all of the student's questions. Navigate to the review marking page.	The student result chart should be visible.	Pass
35.	The system must allow the lecturer to see statistics on how the class performed in tests.	Make a test and add some questions, schedule the test to go live, log back in as a student and sit the test. Once the test has ended log back in as a tutor and go to the marking Section. Manually mark all of the student's questions. Navigate to the review marking page.	The question result chart should be visible.	Pass
36.	The system must allow teaching assistants to communicate with the lecturer.	Navigate to the MyModules page. Click on any tutor's email.	The users email client should be launched with an email addressed to the selected tutor.	Pass
37.	The system must allow teaching assistants to access modules they are assisting with.	Click on the link to go to module home for any modules displayed on the MyModules page.	User should be navigated to module home for the selected module.	Pass
38.	The system must allow teaching assistants to perform actions the lecturers has permitted them to.	Make a test and add some questions, schedule the test to go live, log back in as a student and sit the test. Once the test has ended log back in as a tutor and go to the marking Section. Manually mark all of the student's questions. Navigate to the review marking page.	The question result chart should be visible.	Pass

39.	The system must allow admins to respond to messages from other users.	Navigate to the contact admins area from the more options dropdown. Click on any admin's email. Send an email. As an admin check the email associate with your account.	A new email should be in inbox from the user, and it should be possible to reply to it.	Pass
40.	The system must allow admins to approve the addition of new modules.	As a tutor add a new module. Login as an admin and approve it and then log back in as the tutor.	The module should be available in the MyModules Section.	Pass

Appendix O - Non-functional Requirements Test Cases

	Requirement	Action	Expected Result	Result
1.	The system must store information securely.	Attempt to log in with bad credentials	The system should not let the user access the site	Pass
2.	The front-end must have a consistent user-friendly UI.	n/a	The color scheme and design should be consistent across the site and no issues with UI should be reported during user testing.	Pass
3.	The application must be accessible publicly on the world wide web.	Navigate to http://206.189.210.86/assessment-assistant/	The website should appear.	Pass
4.	The application must be easy to use for new users.	n/a	Users should report no issues with how to use the application during user testing.	Pass
5.	The application should make appropriate use of tooltips.	Go to any page and hover on the logo.	A tooltip should appear indicating this action will bring the user to the home page.	Pass
6.	The user should be able to move around the application with ease.	n/a	Users should report no issues with navigation around the application during user testing.	Pass
7.	Execution of processes within the system must take no longer than 10 seconds.	The largest transaction is uploading a new question that includes an image, do this.	The action should not take longer that 10 seconds.	Pass

Appendix P - Accessing the Application or Running locally

- The application can be found at http://206.189.210.86/assessment-assistant/
- As outlined in the report the application has been optimised for Google Chrome version 72+, so please access it through this browser.
- Some accounts have been set up for your use:

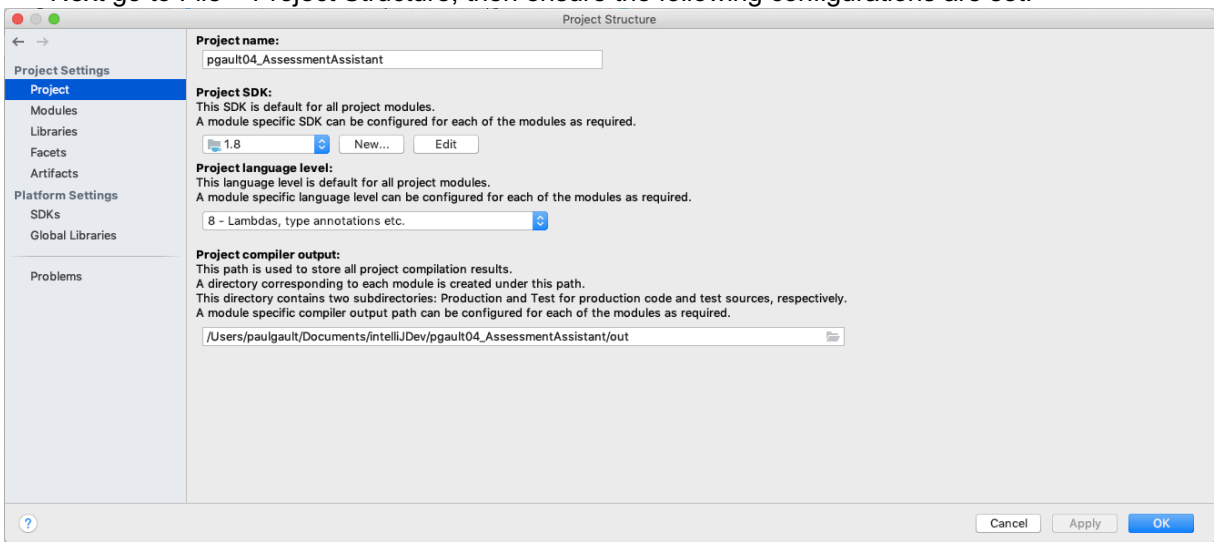
Email / Username	Password
tutor@qub.ac.uk	tutorUser1
student@qub.ac.uk	studentUser1
t_a@qub.ac.uk	assistantUser1

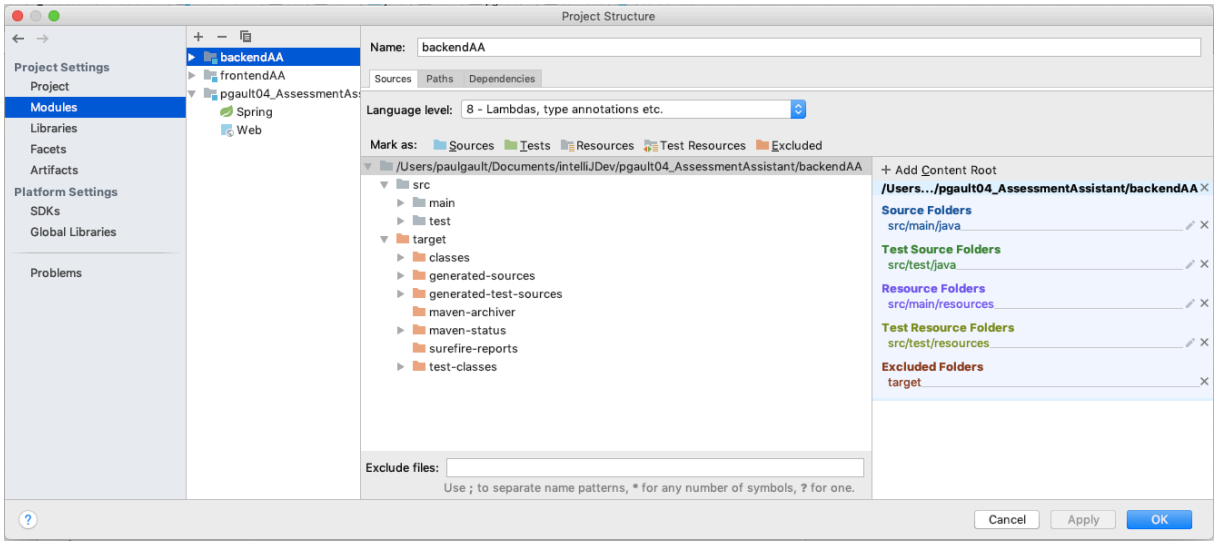
- All of these accounts are registered as Admins.
- The system is populated with some mock data for you to view, edit or test.

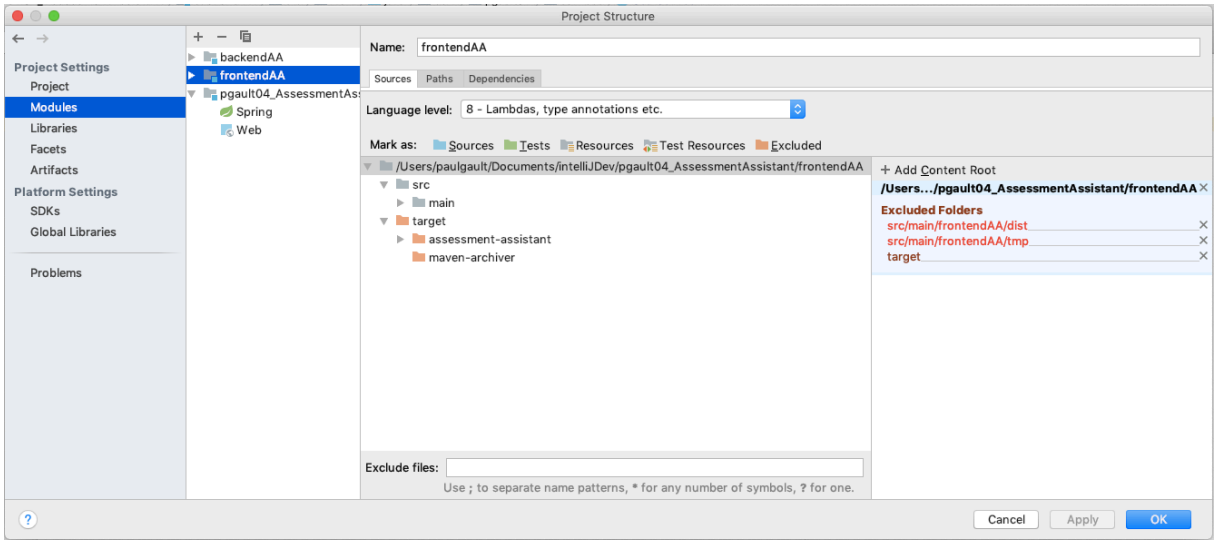
Local Alternative:

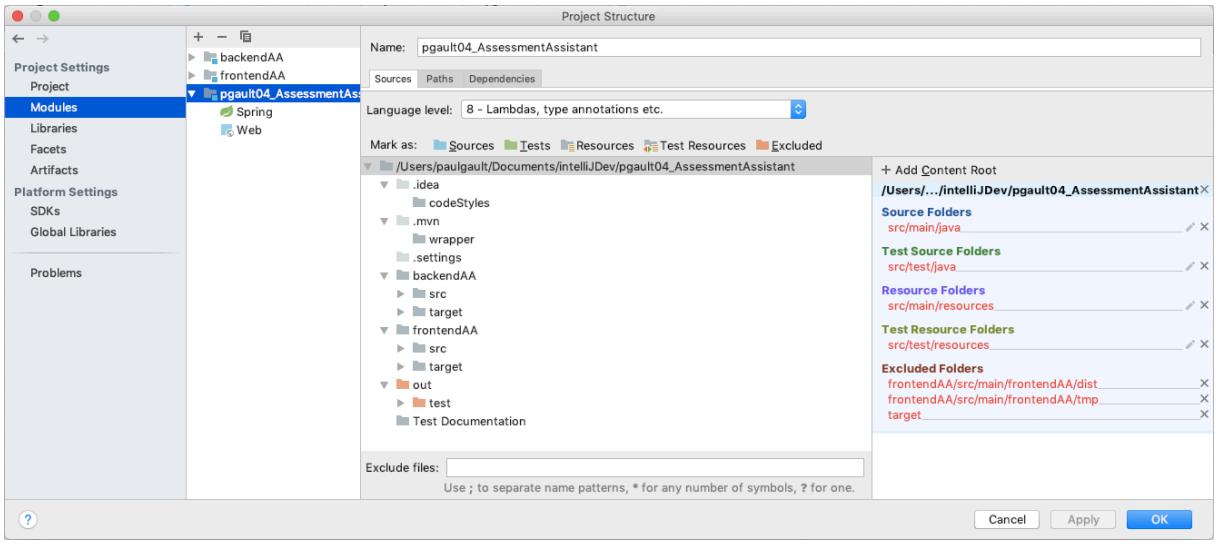
If for whatever reason the application is not available in production, you can run the system locally.

- I would recommend doing this through intelliJ IDEA. If you need to, you can download the community edition for the machine you are working off here: https://www.jetbrains.com/idea/download/#section=mac
- In order to configure the project, you have two choices:
  1. From USB submitted as part of project
     a. Open intelliJ and go to File > New Project From Existing Sources
     b. Choose the folder on the USB named pgault04_AssessmentAssistant
  2. GITLAB
     a. Clone the Repo at https://gitlab2.eeecs.qub.ac.uk/40126005/CSC7058
     b. Inside intelliJ go to File > New > Project from Version Control > Git and paste the link retrieved from cloning the repo. (You may need to remove the “git clone” from the beginning).
- Choosing USB may mean you won’t require to configure the following steps - if so, go straight to the maven clean and install step.
- Next go to File > Project Structure, then ensure the following configurations are set.

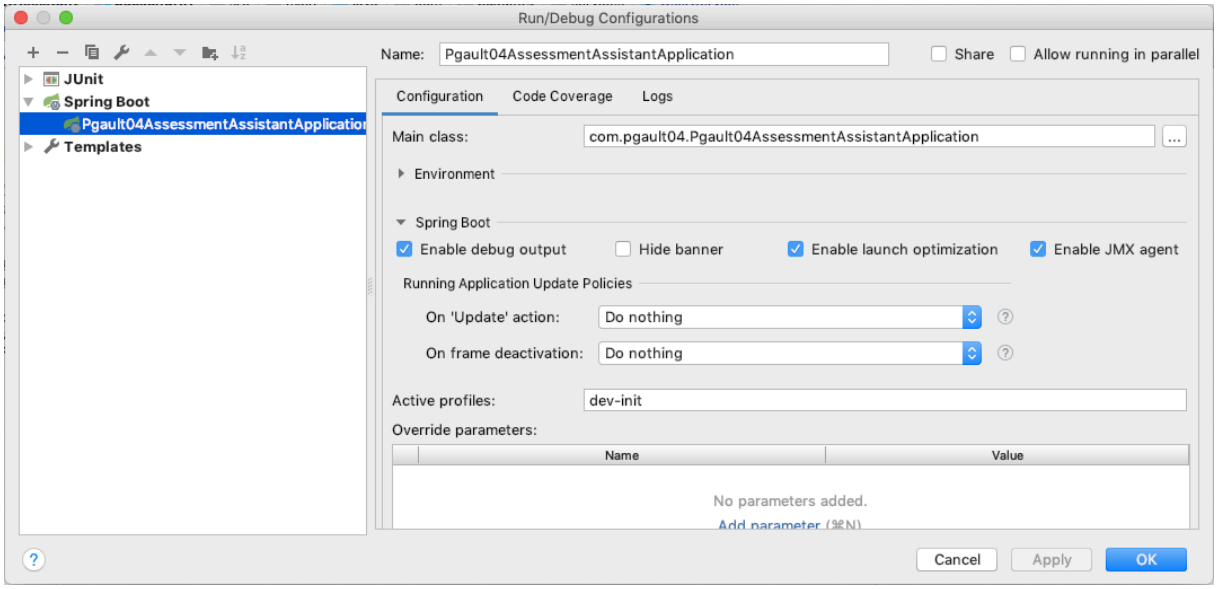



Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
Problems
backendAA
frontendAA
pgault04_AssessmentAs
Spring
Web
Name: backendAA
Sources
Paths
Dependencies
Language level: 8 - Lambdas, type annotations etc.
Mark as: Sources Tests Resources Test Resources Excluded
/Users/paulgault/Documents/intelliJDev/pgault04_AssessmentAssistant/backendAA
src
main
test
target
classes
generated-sources
generated-test-sources
maven-archiver
maven-status
surefire-reports
test-classes
+ Add Content Root
/Users.../pgault04_AssessmentAssistant/backendAA
Source Folders
src/main/java
Test Source Folders
src/test/java
Resource Folders
src/main/resources
Test Resource Folders
src/test/resources
Excluded Folders
target
Exclude files:
Use ; to separate name patterns, * for any number of symbols, ? for one.
Cancel
Apply
OK

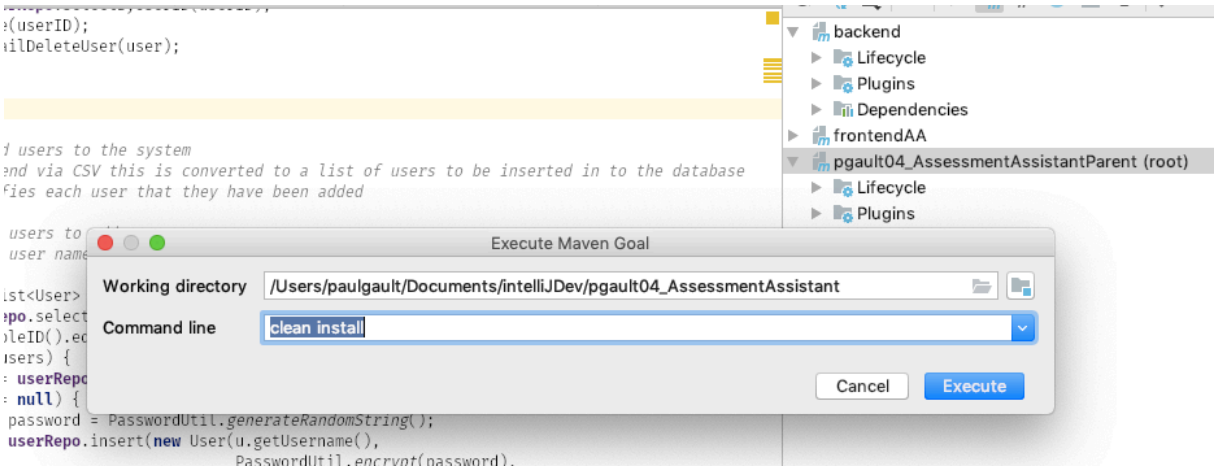

Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
Problems
backendAA
frontendAA
pgault04_AssessmentAs
Spring
Web
Name: frontendAA
Sources
Paths
Dependencies
Language level: 8 - Lambdas, type annotations etc.
Mark as: Sources Tests Resources Test Resources Excluded
/Users/paulgault/Documents/intelliJDev/pgault04_AssessmentAssistant/frontendAA
src
main
target
assessment-assistant
maven-archiver
+ Add Content Root
/Users.../pgault04_AssessmentAssistant/frontendAA
Excluded Folders
src/main/frontendAA/dist
src/main/frontendAA/tmp
target
Exclude files:
Use ; to separate name patterns, * for any number of symbols, ? for one.
Cancel
Apply
OK


Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
Problems
backendAA
frontendAA
pgault04_AssessmentAs
Spring
Web
Name: pgault04_AssessmentAssistant
Sources
Paths
Dependencies
Language level: 8 - Lambdas, type annotations etc.
Mark as: Sources Tests Resources Test Resources Excluded
/Users/paulgault/Documents/intelliJDev/pgault04_AssessmentAssistant
.idea
codeStyles
.mvn
wrapper
.settings
backendAA
src
target
frontendAA
src
target
out
test
Test Documentation
+ Add Content Root
/Users/.../intelliJDev/pgault04_AssessmentAssistant
Source Folders
src/main/java
Test Source Folders
src/test/java
Resource Folders
src/main/resources
Test Resource Folders
src/test/resources
Excluded Folders
frontendAA/src/main/frontendAA/dist
frontendAA/src/main/frontendAA/tmp
target
Exclude files:
Use ; to separate name patterns, * for any number of symbols, ? for one.
Cancel
Apply
OK

- Next go to Run > Edit Configuration, ensure the following configuration.
- The dev-init file will have an exact copy of the production schema at the time of submission.



- Perform a maven clean and install.



- Run the backend application by clicking the play icon.

Frontend

- In order to run the frontend application, you will require node.js
- Navigate to https://nodejs.org/en/download/
- Download the LTS version for the machine you are working on and install it
- Inside intelliJ navigate to the terminal
- cd "PathToProjectOnYourMachine"/pgault04_AssessmentAssistant/frontendAA/src/main/frontendAA
- From here type the command npm start and execute it.
- Go to your Google Chrome browser and the application will be available at localhost:4200/

Appendix Q - Meeting Minutes

CSC7058 Minutes of Meeting Template

Student Name: Paul Joseph Gault **Student Number:** 40126005

Date of Meeting: 03/10/2018 **Time of Meeting:** 17:30

Other Attendees at Meeting: Richard Gault

Agenda:

#1: *Potential Projects for Paul's CSC7058 Module*

Summary of Discussion:

- A run through of some of the potential projects that Richard had to offer.
- There were three main projects proposed.
- At the end the one Paul was most interested in was a system that facilitates the automatic marking of examination papers for a wider range of question types and structures than is already offered by current systems such as QuestionMark.
- The system would be one that facilitated many different user types. For example, student, tutor, teaching assistant and administrator.
- The system would provide features that allow for tutors to choose from a variety of marking options on a per question basis. For example, manually, semi-automatic or fully-automatic.
- The tutor could specify within what timeframe tests that they uploaded are available to students.
- The tutor could issue practice tests which the students could complete and get automatic feedback generated by the system without need for direct consultation with the tutor.
- Users would be able to sign on to the system and see all modules related to them and be able to click in to perform actions or answer tests.

Action Points:

- *Paul would go home and perform some research in to the topic.*
  - What types of question could the system allow for automatic or semi-automatic marking on, and how might this perhaps be implemented?
  - What might be the best approach to this system, what technologies might be used?
  - Begin thinking about the official Project Proposal.
  - Begin drafting some user stories.
- Richard would issue Paul with some required reading.

CSC7058 Minutes of Meeting Template

Student Name: Paul Joseph Gault **Student Number:** 40126005

Date of Meeting: 15/10/2018 **Time of Meeting:** 17:30

Other Attendees at Meeting: Richard Gault

Agenda:

#2: *Initial Progress Report from Paul*
Summary of Discussion: - Discussion of use of Overture in the project. - Review of Database structure. - Discussion of finishing off project proposal and possible use of a Gantt chart. - Deployment - Syncing MySQL server with Local Spring project.
Action Points: - Complete project proposal including creation of Gantt chart. - Amend Database for confirmation emails and user permissions. - Consider how to include truth tables in questions.

CSC7058 Minutes of Meeting Template

Student Name: Paul Joseph Gault **Student Number:** 40126005

Date of Meeting: 29/10/2018 **Time of Meeting:** 17:30

Other Attendees at Meeting: Richard Gault

Agenda:

#3: *Initial Progress Report from Paul*
Summary of Discussion: - Review of progress so far. - Paul had completed Entities layer with full code coverage and some mock-ups of the UI along with Class and Architecture Diagrams. - It was discussed that Paul had been taking a too general approach to the project creation and should instead focus on the key deliverables outlined in the project spec.
Action Points: - Paul would go and focus on creating a functioning: Login Page, Home Page, Add a Test and Add a Question page in time for the next meeting.

CSC7058 Minutes of Meeting Template

Student Name: Paul Joseph Gault **Student Number:** 40126005

Date of Meeting: 12/11/2018 **Time of Meeting:** 17:30

Other Attendees at Meeting: Richard Gault

Agenda:

#4: *Initial Progress Report from Paul*
Summary of Discussion: - Reviewed progress made on coding for project. - The repository layer was completed along with the security configurations, login/logout functionality on front end and back end, home page and module home page.
Action Points: - Paul will fix up all unit tests, make the home page the default page after logging in, create dummy tutors, TAs, and students to populate various modules. Create an Add a test page and allow for basic question types to be added. Add an area for students to take this test and allow for it to be marked.

CSC7058 Minutes of Meeting Template

Student Name: Paul Joseph Gault **Student Number:** 40126005

Date of Meeting: 26/11/2018 **Time of Meeting:** 17:30

Other Attendees at Meeting: Richard Gault

Agenda:

#5: *Initial Progress Report from Paul*
Summary of Discussion: - Review of question and add test forms created by Paul.
Action Points: - Paul would go and focus on the robustness of the application, ensuring that the system accommodates all users use cases to some basic degree.

CSC7058 Minutes of Meeting Template

Student Name: Paul Joseph Gault **Student Number:** 40126005

Date of Meeting: 10/12/2018 **Time of Meeting:** 17:30

Other Attendees at Meeting: Richard Gault

Agenda:

#6: *Initial Progress Report from Paul*

Summary of Discussion:

- Discussion of progress made by Paul over the course of the last few weeks.
- Paul had added features to the module home page and made the website more robust and easier to navigate.
- Paul had also switched and began developing his front end with use of Angular 6.

Action Points:

- Paul would go and make a handful of corrections to content that has thus far been created incl.
    - Fix active username bug, rename some features to more sensible and intuitive names, and replace some navigation bars with dropdowns to account for when many options (i.e. tests) are available.
- Paul would also complete a handful of user stories for after the Christmas break incl.
    - Tutor can create a test and set the most basic types of question e.g. text input.
    - 3 students can take this test and receive a range of auto generated marks.
    - Each student can see how they did after the test is marked and score released.
    - Tutor can see the class performance before it goes public (to students).
    - Tutor can edit test drafts and move them back and forth between being ready for release and in process.

CSC7058 Minutes of Meeting Template

Student Name: Paul Joseph Gault **Student Number:** 40126005

Date of Meeting: 14/01/2019 **Time of Meeting:** 17:30

Other Attendees at Meeting: Richard Gault

Agenda:

#7: *Progress Report from Paul*

Summary of Discussion:

- Paul updated Richard on progress that had been made over the Winter Vacation.
- This include completion of the Edit Test page, for three question types (Text, Insert the word, and Multiple Choice).
- The ability for these tests to be scheduled for release was added.
- Once the release date arrives the functionality for the students to take the tests was added.
- Auto-marking was made added for submissions of all three question types.
- Functionality for Lecturers / Tutors to assign scripts and view metrics on how their markers are performing was added.
- Validation added for all form inputs including insuring sensible dates (end date can't be before start date etc).
- Fixed bug that led to firsts user's username still being displayed when a user logs out and another logs in.
- 100+ unit tests written bringing the total to 300+.
- We looked at GradeScore and how it will be a useful thing to compare against in the report.

Action Points:

- Paul will go and fix some bugs - the bug that stops the test from being returned to drafts on click and the bug that prevents the auto-marking of insert the word questions.
- Paul will attempt to have the system ready for some user acceptance testing for the end of February.
- For the next meeting Paul will attempt to have all bugs fixed, the marking user interface completed, some tweaks in the creation of insert the word questions to make it more intuitive, review marking will be completed to allow tutors to preview scores and grades before they are released, create account / add user functionality, add module functionality and lastly if there is time an admin area for accepting requests to add modules.

CSC7058 Minutes of Meeting Template

Student Name: Paul Joseph Gault **Student Number:** 40126005

Date of Meeting: 29/01/2019 **Time of Meeting:** 17:30

Other Attendees at Meeting: Richard Gault

Agenda:

#8 *Progress Report from Paul*
Summary of Discussion: - Paul had begun work on the Mark Test functionality. - All auto-marking now fully functional for the current question types. - Some bug fixes made to scheduling tests and insert the word auto-marking.
Action Points: o Paul will finish the Mark-Test functionality. o Paul will add functionality for adding and approving users and modules. o Paul will add basic functionality for mathematical questions. o Paul will if times permits add the review marking functionality.

CSC7058 Minutes of Meeting Template

Student Name: Paul Joseph Gault **Student Number:** 40126005

Date of Meeting: 11/02/2019 **Time of Meeting:** 17:30

Other Attendees at Meeting: Richard Gault

Agenda:

#9 *Progress Report from Paul*

Summary of Discussion:

- Paul demoed the completed mark-test and review marking functionality.
- The next steps for the project where discussed

Action Points:

Admin:

- Admin Area for approving modules and users.
- Password reset.

Teaching Assistant:

- Active Tests should be viewable from module home but not accessible.
- Fix bug which shows no marking stats.

Student:

- Performance needs to be configured so students can see their results.
- Exam feedback needs to be made available.
- Functionality needs to be added where students can query marks given in practice tests.

Tutor:

- Auto-marking needs to be fixed to consider question type.
- Assign marking bug needs to be fixed.
- When marking the student last marked needs to be displayed not the first student.
- Auto-marking needs to auto-approve Multiple choice questions and Input based questions.
- Add module functionality needs to be added.
- Add student's functionality needs to be added.
- Add teaching assistant's needs to be added.
- Practice tests need to be added.

CSC7058 Minutes of Meeting Template

Student Name: Paul Joseph Gault **Student Number:** 40126005

Date of Meeting: 25/02/2019 **Time of Meeting:** 17:30

Other Attendees at Meeting: Richard Gault

Agenda:

#10: *Progress Report from Paul*
Summary of Discussion: - Paul demoed the completed add module, add students / teaching assistants and tutor area. - The next steps for the project where discussed.
Action Points: - Fix bug that ruins token store in Database when user logs out. - Make add user csv include all user fields i.e. first name and last name need to be added. - Add the add user csv to the admin page. - Add the view all users widget to the module home page that views all associates of the module. - Finish mathematical renderers throughout system. - Add practice tests.

CSC7058 Minutes of Meeting Template

Student Name: Paul Joseph Gault **Student Number:** 40126005

Date of Meeting: 11/03/2019 **Time of Meeting:** 17:30

Other Attendees at Meeting: Richard Gault

Agenda:

#11: *Progress Report from Paul*

Summary of Discussion:

- Fixed bug that ruins token store in Database when user logs out.
- Add user csv now includes all user fields i.e. first name and last name need to be added.
- The add user csv is now also on the admin page.
- The view all users widget is now on the module home page that views all associates of the module.
- Finished mathematical renderers throughout system (question creation → answering → marking).
- Practice tests now available, tutors can always edit them even when live, students receive feedback instantly, can answer as many times as they want and can query results for tutor to amend in marking area or in test draft.
- Database deployed to production server along with server-side system.

Action Points:

- Paul will finish deployment by deploying the front end.
- Paul will verify everything looks and behaves as it should in the production build and create a Google Doc for User Acceptance Testers to follow.
- Paul will be on hand for all issues during the user acceptance testing phase.
- Paul will begin to concentrate on documentation and report for the project.

CSC7058 Minutes of Meeting Template

Student Name: Paul Joseph Gault **Student Number:** 40126005

Date of Meeting: 25/03/2019 **Time of Meeting:** 17:30

Other Attendees at Meeting: Richard Gault

Agenda:

#12: *Progress Report from Paul*

Summary of Discussion:

- Paul had completed deployment of entire application and deployed it to a server held on digital ocean.
- Paul also completed a document for user acceptance test in the form of a google doc for users to access online.
- Paul completed the finalised ER diagram.
- Paul also generated some class diagrams for use in the documentation phase.
- Paul and Richard discussed some best practices for writing the report.

Action Points:

- Paul will attempt to have some progress made on the report for Richard to review come the end of the week.
- Whenever student access to the Docker container becomes available Paul will attempt to move the system over to that server.

CSC7058 Minutes of Meeting Template

Student Name: Paul Joseph Gault **Student Number:** 40126005

Date of Meeting: 8/04/2019 **Time of Meeting:** 17:30

Other Attendees at Meeting: Richard Gault

Agenda:

#13: *Progress Report from Paul*
Summary of Discussion: - Paul had completed the first 2 Sections of the report. - Richard and Paul discussed the need for more references throughout the report and some changes that needed to be made. - Richard advised Paul on how to optimise the writing style for the report.
Action Points: - Paul would attempt to get Chapters 3 and 4 completed by the end of the week and send them to Richard for review. - Paul would attempt to get Chapters 5 and 6 completed by the end of the following week.