Senior Honours Project

Exam Typesetting Tool

University of St Andrews

Jamie Maclean

Supervisor: Dr. Graham Kirby

Date of Submission: 04/04/2016

Abstract

It is important for exam papers to have a rubric that is consistent between papers and that the exam itself has the correct number of marks according to that rubric. The research project discussed here, aimed to make an application to accomplish this and make the whole exam generating process easier. This paper looks at the design and implementation of this application and what it can deliver to the reader. Evaluation is also made on the application based on the ideas originally made at the start of the project and discuss ideas for future improvements to the system. A user of the system can choose to upload or, if they so wish, create an entirely new paper. The paper text can be edited, the paper's spelling can be checked and it is possible to see if the paper matches the rubrics set for the paper. These papers can be saved to a file to be edited further. It is also possible to set a secretary or an external examiner to an exam whom can then be emailed if desired.

Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project report is 11,272 words long, including project specification and plan.

In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the report to be made available on the Web, for this work to be used in research within the University of St Andrews, and for any software to be released on an open source basis. I retain the copyright in this work, and ownership of any resulting intellectual property.

Contents

1. Introduction

The creation of exam papers can be a tiring process and usually not a topic that brings excitement. Exams should be consistent between papers and contain the same number of questions and format as previously related ones have been. Sometimes, exams have been created that do not necessarily follow the previous papers, having a different rubric and style which in certain cases is not ideal.

One of the problems making exam preparation a lengthier process is the lack of automatic text generation. An exam has to be written from scratch meaning there is more time to be spent on planning the paper. With exams being very similar in style for their front page, it should be easier to automatically enter the text from this to lower the amount of content an exam writer has to create.

An exam needs to have the correct number of marks otherwise it would not be valid. The marks at the end of each question need to match the rubric that belongs to the corresponding course and, also, the exam needs to match the entire total number of marks for the exam rubric. This can be annoying for a reviewer where they have to manually count all the marks one by one to make sure they add up to the final total and the total per question.

The exam writer is not the only person involved with the preparation process. There are administrators, external examiners, and secretaries who help to review the exam and give input into the exam process. All of these positions need to be assigned and the person contacted for a smooth and legitimate delivery of an exam paper.

There has not been a lot of thought, if any, into developing a tool to try and combat these problems. However, the idea behind achieving this should not be expensive to create and not need a large team of developers to create. There has been a lot of work towards documentation creation in general and, even as this is being typed, the systems made for this are being heavily used. Although, these tools are not geared towards the exam paper process specifically, meaning less coherency between papers is possible.

The exam paper application created is an investigative research project looking into making the whole exam paper creation process easier. This paper describes the methods and design behind this application and looks into the implementation behind the system. Evaluation on the project as a whole is also conducted and reported here looking into future possibilities and improvements.

2. Objectives

These original objectives were decided at the beginning of the project to try and get an idea of what the system would attempt to achieve. Of course, there were many changes and additions to these throughout the year as they weren't necessarily the direction the final system should go. They would later help in analysis of the application and evaluate whether the objectives were met.

Primary Objectives
To have a tool which gives the ability to write and format an exam paper ensuring that there is a consistent page layout for all papers. The paper should be able to be saved as pdf or document so it can be integrated with different tools. The tool should store the rubrics of papers that can be used to automate part of writing the paper itself and have the possibility to change these if the rubrics change. Then, it should be possible to check whether the marks written in the paper match the stated rubrics.

Secondary Objectives
Spell checker to make sure everything in the paper that has been written is spelt correctly.

Tertiary Objectives
The ability to set external examiners to an exam paper and automating part of this section of the exam preparation stage such as sending out an email to the given external examiner. Also, the ability to set a secretary to a certain exam paper. The tool should make sure the transmission and storage of papers is secure.

3. Requirements

Requirement #1
Description: It should be possible to write an exam paper within the system.
Rationale: This is to allow the user to be able to create their own exam paper from scratch.

Requirement #1.1
Description: The papers should be able to be formatted within the program to make changes to the text.
Rationale: The reason for this is to let a user make changes to existing past papers or edit current papers.

Requirement #2
Description: Papers should be able to be saved as a document or pdf.
Rationale: This is to allow a user to save the paper so that it can be edited again later on. Another possibility would be for a user to start writing on the system, save and then use another program to edit.

Requirement #3
Description: The exam rubrics should be stored containing the number of questions, number of marks and any exam text
Rationale: This means that it would be possible to check if the paper has the correct number of marks and for automation possibilities.

Requirement #3.1
Description: The default rubrics for Honours and Sub-Honours modules should be stored and the system should have the ability to change them.
Rationale: The reason for this is so that the rules can be automated and the possibility to change them is allowed.

Requirement #3.2
Description: The default rubrics can be used to automatically add skeleton text set by the user to the paper based on the said rubrics.
Rationale: This is so that the user can save time having to write the text and questions for the exam paper.

Requirement #4
Description: The number of marks within the paper should be checked with the rubrics mark to make sure they match.
Rationale: This is to make sure that the written paper has the correct number of marks and does not contain more or less than the total marks.

Requirement #5
Description: The system should have a spell checker option to check the written papers spelling.
Rationale: This is for the user to see whether the words they have typed have been spelt correctly.

Requirement #6
Description: The system should have the ability to set an external examiner.
Rationale: This allows the user to set an external examiner to a paper so that they can be contacted at some point.

Requirement #6.1
Description: It should be possible to email this external examiner using a default email.
Rationale: This is to save the user having to type an email themselves to an external examiner.

Requirement #7
Description: The system should be able to set a secretary for a paper.
Rationale: This is to allow the user to give an exam paper a secretary of their choice that can then be contacted.

Requirement #7.1
Description: There should be a possibility to email the given secretary within the system.
Rationale: This is to let the secretary know that they are needed for a certain exam

Requirement #8
Description: A default email can be set for emails to an external examiner and a secretary.
Rationale: This is to allow the user to change the default email that is sent to the specified people.

Requirement #9
Description: Any transmission of papers should be kept secure.
Rationale: This is to make sure that when a paper is being sent across a network that it is not going to be compromised.

Requirement #10
Description: The storage of papers should be kept secure.
Rationale: When storing papers, they should be kept safe so that only the authorised people are allowed to use them.

4. Context Survey

When searching for relevant background information it was apparent that there had not been any recent work with similar aims. Nobody had looked into the exam paper process and thought about making an application to make it easier. There, of course, have been plenty of applications made for creating documents but none specifically for exam papers.

It is important to understand the current exam creation process to understand how it could possibly be improved. I spoke with the university’s exam administrator to get a better idea of the process. Firstly, an email is sent to remind an exam writer to submit the exam to the module coordinator by the deadline. An exam checker is the told that this has been done and they then check to see if the spelling is correct, the marks are correct, and the exam is fair.

The administrator then makes sure it is alright and has appropriate answers to the paper. Each Honours level has a different external examiner who talks to the administrator and also checks that the exam is appropriate and comments are made and sent back. The exam is then corrected if any mistakes were made and the administrator tidies it up for the module coordinator to check.

Looking at document creation more generally, there has been a lot of work done. There have been many applications made for this looking at research papers or even just publicly available software. G2 Crowd conducted some analysis into various software for this functionality and rates them to find the best. According to them, Adobe Acrobat is the best software for document creation followed by other popular applications, Microsoft Word and Google Docs [1]. These function as word processors, and PDF creator and editors allowing the user to create documents of many types.

Online collaboration of documents is also becoming increasingly popular. Google Docs is a big example of this with many people using the tool for group projects. A research project by Pilkington and Sanders [9] looked into an online collaborative document creation application to be used for lecturers in group study. This looked at allowing students to work on creating documents containing questionnaires and interview questions collaboratively online. This allowed the lecturers to be able to ensure students were still on track and challenge them to go deeper into the subject.

Tasks could be subdivided between students, and students could be divided into smaller groups. However, a problem with applications such as this, is the free rider effect where different people have different levels of work and effort.

Recent research has also been carried out into simplifying the documentation creation process. A recent project by Schultz *et al* [10] looked into this with the use of user made templates. A user creates a template using a first user interface and then can create content files based off this with the first or a second interface. Both these parts are then combined to create documents.

This second interface has user-defined limited functionality based on high level users allowing newer users to easily create their own documents without much supervision. It automates some of the more complex tasks such as linking documents meaning that it is easier for first time users to add a file to a set of documents.

This project looks to build upon these ideas of document creation by looking at building documentation for learning purposes, simplifying the creation process in general, and looking at the current software available. Also, the application built will try to construct upon the exam paper creation process described to try and help the users the system is designed for.

5. Methodology

This application was a project based on any relevant research. As a result of this, with changes to requirements and project objectives, it would have been very difficult to create a detailed plan for the software engineering process originally. However, there were meetings assigned once a week with my supervisor to discuss the project. It was possible, within these meetings, to talk about what work had been done in the last week and look at how this can be improved. We could discuss any questions I had for the future of the project, what I should plan on doing for the next week, and any possible obstacles that would stop progress.

Before these meetings, I was able to have some time to think for myself any questions prior to them and think about what was left to do in the project. This and the meetings helped to break the entire process into more manageable chunks, which is important for keeping the flow of the project going and understanding where things need to be implemented or improved.

The application itself was firstly started by creating a structure to the interface which would be built upon. New components could then be added to this to make sure that it did not break the current system. Once one component had been fully tidied up, the next one could be created so that eventually a working system would be made.

6. Ethics

As discussed at the start of the project, there were no ethical issues raised by the process. Self-audit was conducted to make sure that this was the case, as opinions were gained from test users. However, thought was put into making sure nobody would be put under any stress or harm, and would not create any ethical issues.

7. Design

Once the main aims were thought through and the requirements were made for the project, the design could be conducted. It was important to firstly think of how the application would look so that it could be built upon and then be implemented. Care was taken to make sure the requirements were met by the design and that the objectives would be met.

7.1. Project Overview

The first point to look at is the flow process of how an exam paper is created and managed. This helped to see how different parts of the system would need to interact and which steps were more important over others. Although not everything was fully developed, each part of the application was designed to work with one another.

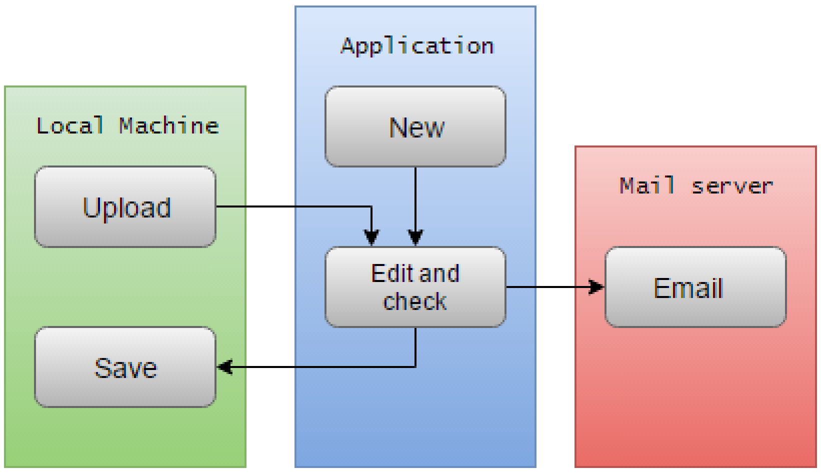

Figure 1: Exam paper flow

As seen in figure 1, an exam could be either created from scratch or uploaded by the user into the system into text format. Text, PDF, Document, and Latex files could be search for and uploaded into the system. These exams could then be edited within the application itself to check the marks, makes changes, or check the spelling within the system. Then once that was sorted, the exam could be saved to a file for another time or emailed to a certain email address.

7.2. Application Interface Overview

Firstly, the exam paper application interface was needed to be designed. The interface was designed to be simple and easy to use, and more focus was put to making it user friendly than cutting edge style.

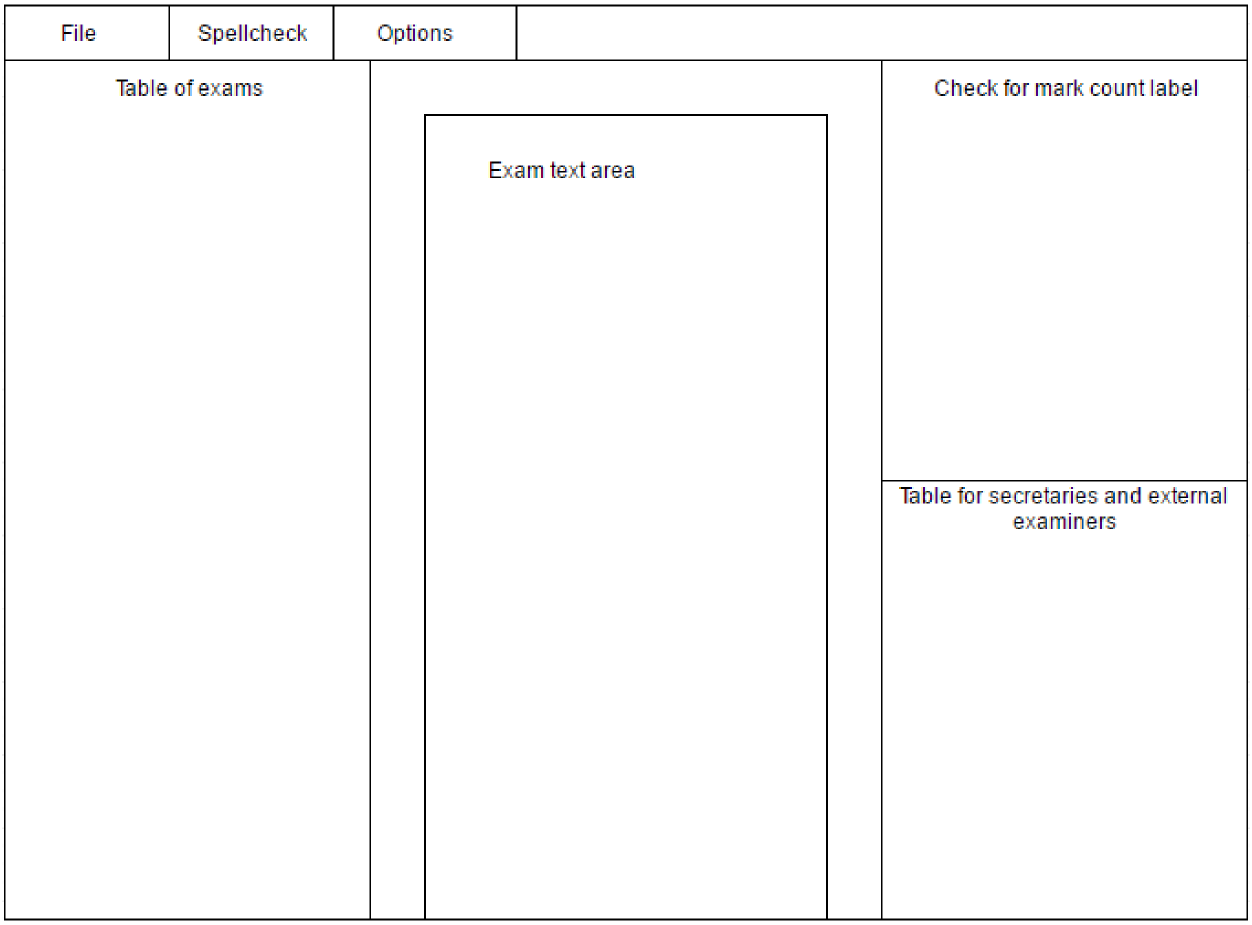
File
Spellcheck
Options
Table of exams
Exam text area
Check for mark count label
Table for secretaries and external examiners

Figure 2: Original application design

The implementation was based off this original application design of the interface, shown in figure 2, and only minor changes were made to this in the menu bar. The left hand side was designed to be a list of exams that were created and saved in the system. It would be possible to select certain exams that were stored, so that they could be edited and possibly saved to the computer locally. Selecting an exam would bring up the exams main text and show the secretaries or external examiners that are assigned to the exam.

In the middle is the text area in which the exam could be created or viewed in. Changes to the exam could be made here that could then be saved to a file and the system. This was designed to show the main text of the selected exam in the exam list column.

The top right of the application is a label containing the details of the current papers marks to the set rubric for that paper. Once the marks checked was clicked in the menu, the current marks per question would be listed here, showing how many marks for each question there currently was.

Underneath this, there is a list for external examiners and secretaries. When a secretary or examiner is assigned to an exam their details, such as email and name, will appear here. It will also let the user know if they have emailed said person using the application, letting them know that they have been assigned to it.

The menu bar was, at first, designed to be split into file, options, and spell check options. Spellcheck was later planned to be in a tools menu item and a help option was added as well. The file option was for the user to be able to create papers and save them. The tool option was for checking the exams and the edit tab was for changing any default options of the application.

7.3. Menu Overview

As discussed before the menu was split into file, edit, tools and help tab.

File Overview

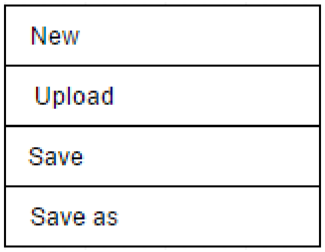

Figure 3: File menu options

The file menu tab was for options such as uploading an exam paper, creating a new exam, and for saving an exam.

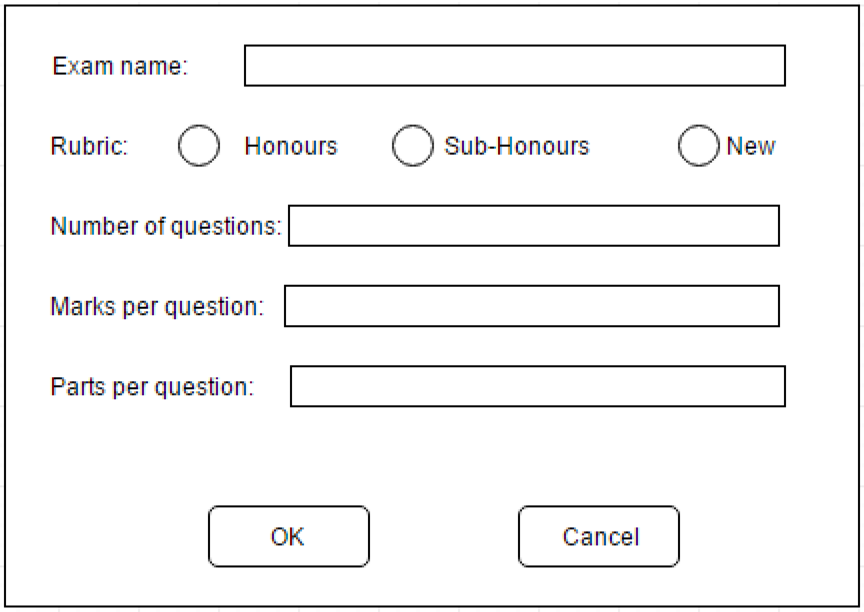

Figure 4: Create new exam design

Once create a new exam was clicked, a new dialog box would appear. This would have a text field so that a name could be entered for the exam that would appear in the exam list. The rubric for the exam was also needing to be entered. The options for this rubric would be to choose the default Honours or Sub-Honours rubric, saved by the system, or create an entirely new rubric altogether.

To create a new rubric a number of questions would need to be entered and the corresponding marks per question would need to be added. Later designed was an option to add parts per question which would split the questions into parts depending on what the user entered. Then the new exam could be confirmed by the ok button or the whole dialog box cancelled.

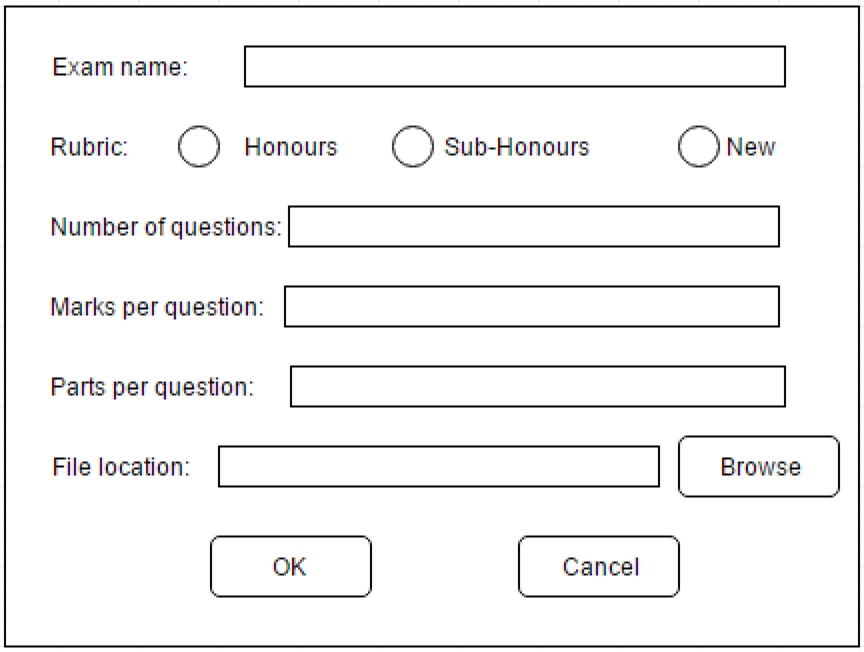

Figure 5: Upload exam design

Uploading an exam has a very similar to design to creating a new exam. Again, the user chooses a name and the rubric for the exam. However, this time the application asks for a file which can be found using the browse button. There will be an option to look through txt, docx, pdf, and tex files that are desired to be uploaded into the system. This should read the text from the file into the application text area to the related exam once the file has been found and ok has been pressed. There is an option to cancel at any point also.

The save option would firstly, if selected, check whether an exam was selected and display an error if not. If an exam was selected that has not been previously saved before the save as dialog, explained below, would be opened. If the exam had previously been saved, the system would save the text within the text area to the exam. It was also designed to save the text to the file that had previously been saved as by the user.

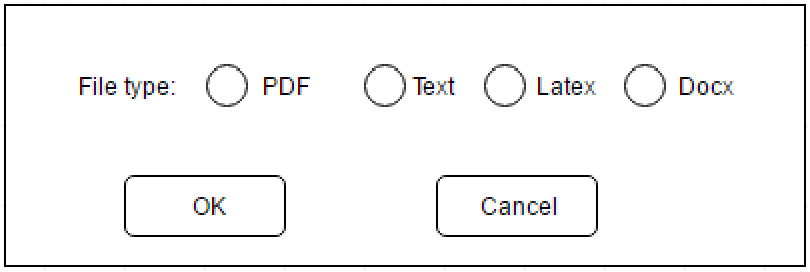

Figure 6: Save as design

If save as was selected, the application was designed to have the option to select a file type for the exam to be saved in, being pdf, text, tex and docx format. Originally, files were planned to be able to be saved as pdf. However, as discussed with my supervisor as this was simply text format, saving as a pdf would not really be required so this was not fully implemented. Then once the type is confirmed, a check is made that an exam has been selected. If one has been selected, a file destination can be selected and a file name entered that will be saved locally and to the application exam's file destination. Again, cancel could be chosen to give up on saving.

Edit Overview

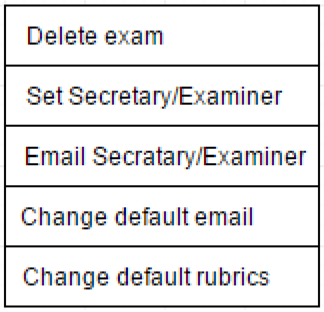

Figure 7: The edit menu options design

At first, there was no way to delete an exam stored within the system so it was obvious that a way to remove an exam from the exam list was needed.

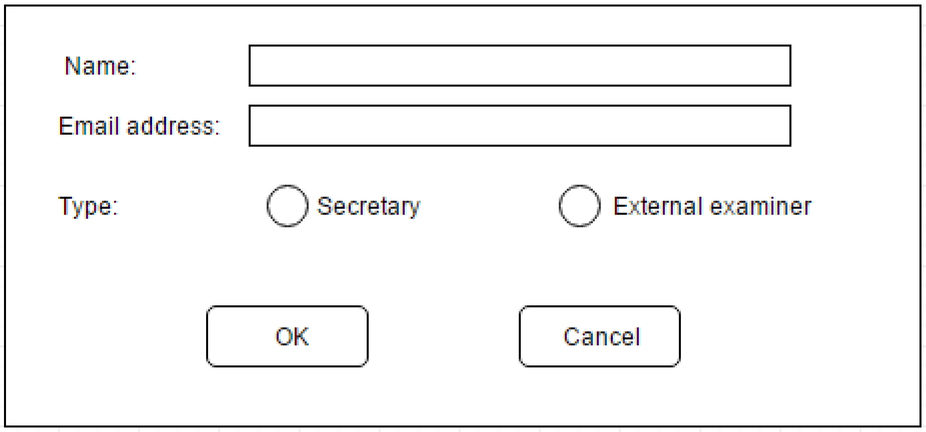

Figure 8: Set secretary and external examiner design

Once an exam was selected you could set a secretary or external examiner to it. This would ask for the user to enter the name, email address and to select whether the person was a secretary and external examiner. Once these details were confirmed, they could be added to the secretary and examiner list for the corresponding exam. An option was once again needed to delete a person from this list. Within this list, a person could be selected that could then be emailed.

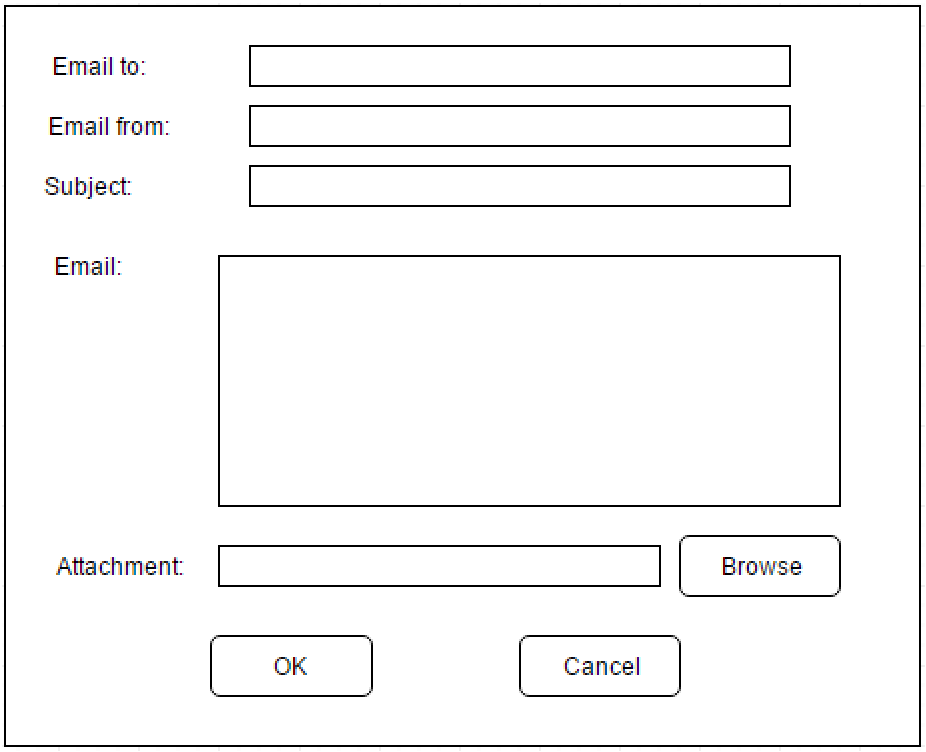

Figure 9: Emailing secretary and external examiner

As seen in figure 9, this has a label containing the email address to be sent to taken from the selected person from the list. Then, the email from would be taken from the default email address file which could potentially be changed and a password associated with the email would be required. A subject line could then be entered and the application’s default email could be edited. There is an option to add an attachment to the email and once the required elements were entered the email could be sent.

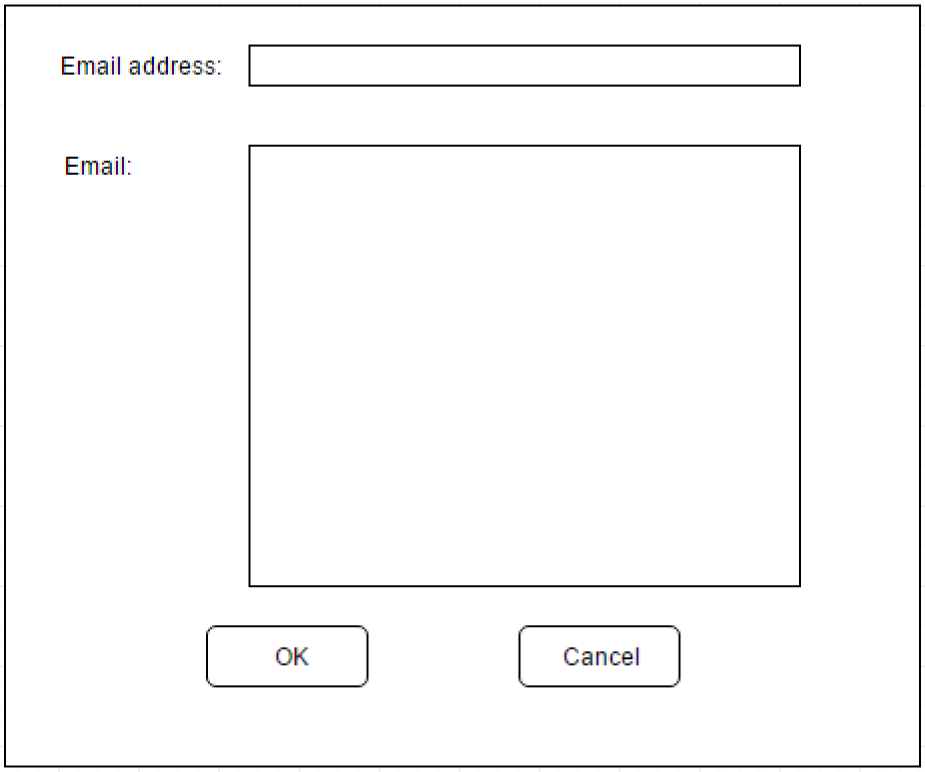

Figure 10: Default email

Within this, the user can create what the default email address and default email will be for the application. This will be saved to the corresponding files to be used for emailing secretaries and external examiners as discussed above.

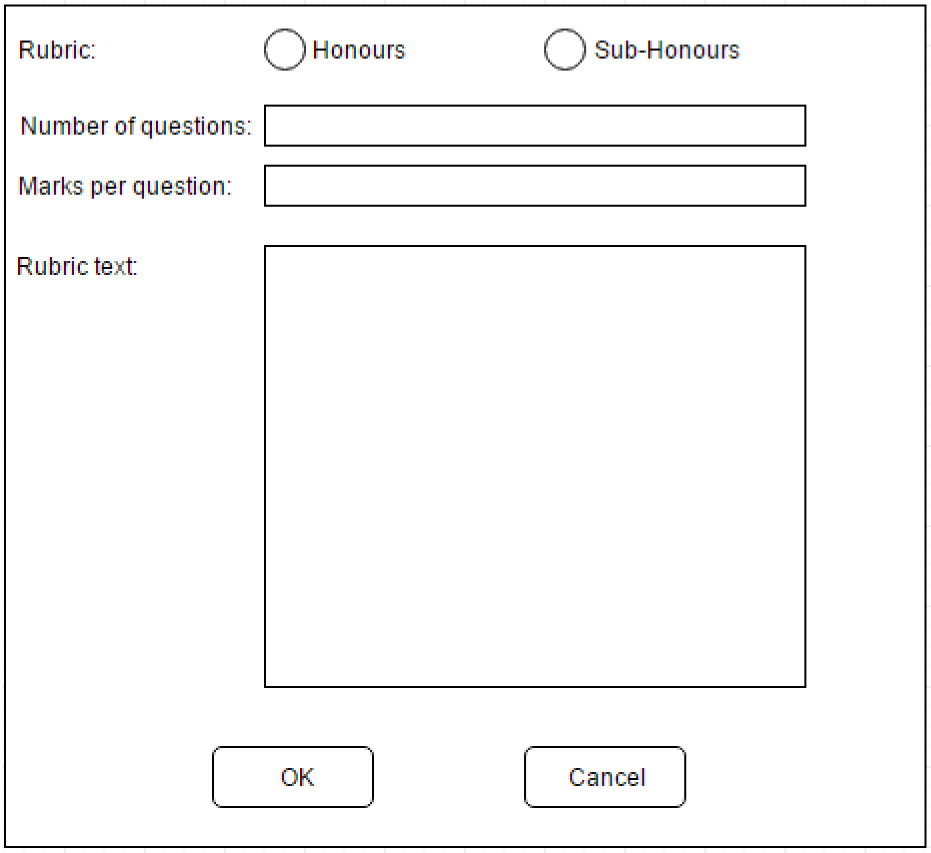

Figure 11: Default rubrics

There is an option for a user to change the default rubrics for an Honours and Sub-Honours exam for the application. The user must select which rubric they want to change and then select the number of questions, the marks per question, and the main body text of the default exam. If a user switches between the Honours and Sub-Honours rubric, the rubric text was designed to change depending on the currently selected rubric.

Tools Overview

The tools menu item was planned to contain two features being the spell check and mark check tools. The check marks option would look at the main exam text and then display the current amount of marks in the label in the top right of the interface. It would check for the number of marks contained within square brackets between questions and calculate the total number of marks contained within a paper. Sometimes, exams like to have an extra total of marks at the bottom of the question so the system was designed to ignore this unless there was only one part to the question.

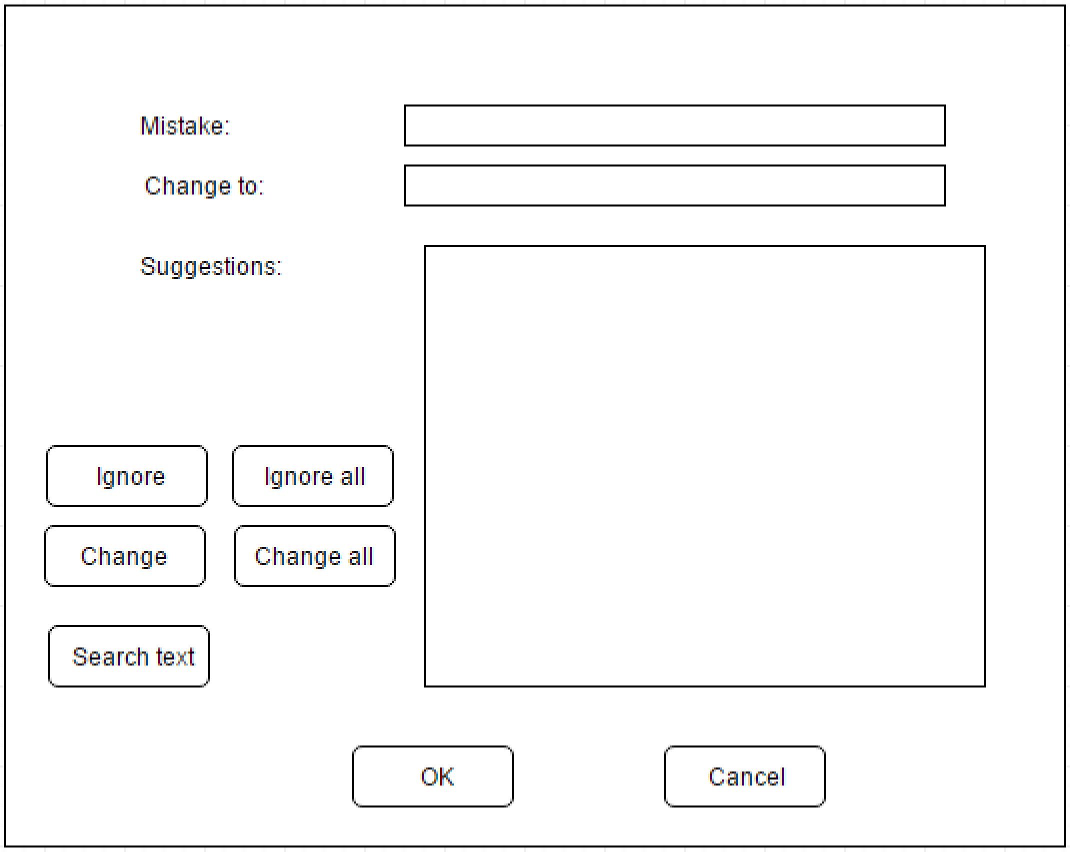

Figure 12: Spellchecker design

The spell checker was designed to get the user to search the text first to find any spelling mistakes within the exam compared to the system's dictionary. Then, if a word was not found, it would be displayed to the user and a list of possible changes would be suggested. The user can select which word from this list that they desire the spelling mistake to be changed to. There is an option to ignore the current change, ignore all changes, change the current mistake, or change all mistakes found. Then once ok was clicked, the exam's text would be updated.

7.4. Other Design Decisions

The exams and rubric classes were designed to be serialisable so that they could easily be stored to files once closed in the default files folder. This made it easy to reopen them later so that their states and information could be accessed where they left off when the application was last closed. The help tab was also added to give information about the application to the user.

For the default rubrics, I went through past papers and analysed their rubrics at Honours and Sub-Honours level. The unique rubrics were noted down and the number of times that rubric occurred was counted. The rubric with the highest amount of occurrences was used. For Honours, it was found that 100% of the papers looked at had 4 questions with 20 marks each so this was used as default. Sub-Honours rubrics were very varied with an unclear majority for a default rubric as each course had a different one. However, the most common rubric due to quantity was a paper with 2 questions and 25 marks for each so this was made to be the default.

8. Implementation

8.1. Technologies Used

Core Technologies

The project itself is a Java based application made using Eclipse. The main reasons Java was used over other languages, were better experience overall and interface creation would be a lot simpler. Eclipse was also used as it was readily available and, initially at the start of the project, so that Eclipse's Rich Client Platform could be used. Although, this was later dropped for JavaFX due to problems with the software.

JavaFX and Scene Builder

JavaFX is a set of media packages allowing users to design and create rich client applications across diverse platforms [2]. Scene Builder provides a visual layout for JavaFX without the need for writing code allowing the user to drag and drop positioning for a GUI JavaFX scene [2]. The exam application was firstly laid out in Scene Builder based off the original design. Once finished, an FXML file would be created which could be loaded using an FXML loader in the code to display to the user.

An FXML file needs a controller class for the code to interact with the user interface. In the controller class attributes and methods can be used by Scene Builder by inserting @FXML before them. Then, components within the scene, such as a text area, could be set to this unique attribute name and actions could call these methods, for example, when a button is pressed. The exam application had many FXML files for certain menu items which each had an associated controller class linked to them.

Scene Builder had a reasonably steep learning curve and was quite pernickety but, when I got past that, was really useful for building an interface with all the relevant components as they were right there for the user to see. JavaFX was also very simple to use with a lot of documentation along with it meaning delivery of an application was very smooth. Both are very readily available for Eclipse and have flexible licensing.

Javax Mail

The Java mail API allows a platform and protocol independent framework to build mail and messaging applications [3]. This can be used to set channels and port properties for communications through mail protocols. An email can be created by building up all the relevant parts needed to create one such as the subject and recipients.

This was used by the application for sending emails through the live SMTP using Outlook and Hotmail accounts. An email was also built using this for transfer based on the user's input. The tool was used as it seemed to be the main mailing platform for Java and was easy to understand.

Language Tool
The language tool package is used on text for checking spelling and listing any errors that appear based on the language of the tool created. Words to be ignored or accepted can also be selected for the language tool [4].

This was chosen to be used as there were examples of spellcheckers online using the tool that appeared to work correctly with slight improvements to the code. It was used by the application's spellchecker to find any mistakes in the exam text and list the grammatical and spelling errors within the text area. A list of possible suggestions was also listed for each mistake so that the user could change it.

XMLBeans
XMLBeans is a technology that can access XML within Java by setting it to Java types. It provides many ways to look at the XML for Java by looking through the XML schema and can also support XML DOM [6].

XMLBeans can be used for word document files and the traversing and saving of these documents making it appealing for the application. At first it was quite confusing but, when looking deeper into documentation, it turned out to not be as complex as first implemented. The library was to be used for the uploading and saving of words files within the application.

PDFbox
The apache PDFBox library is used a free to use Java tool for PDF documents specifically. This allows the creation of new PDF documents, manipulation of current documents, and the ability to extract content from documents [7].

As a result of this, the tool was very helpful for the reading and manipulating of PDF documents for the system. A PDF file could easily have its text extracted for the exam text to be input into the application.

8.2. System Architecture

8.2.1. Exam Data Structures
An exam itself was structured to have a contents, a rubric, a name, if it was saved, the file type of when it was last saved, the file location of this, and a list of people that could be emailed. The contents are for the main text of the exam to be saved and edited within the application. Each exam has a rubric that is assigned to it so that the marks can be checked.

The exam name is needed to show up in the exam list as a nice way for the user to identify each exam. There needs to be a Boolean for if an exam was saved so that the application knows to either show the save as dialog or simply save to the last location and file type which is also why the file type and location are also stored. Each exam has a list of email recipients that can have secretaries and external examiners added to it.

A Rubric was implemented to have a number of marks, a number of questions and the marks per question. This was mainly needed for checking the marks to check if the exam text has the same amount of marks as the set exam rubric.

Two further classes extended from this were needed for the default Sub-Honours and Honours rubrics which also contained a default string of text to be automatically entered for an exam's text if the user selected said rubric. I decided to implement two different classes so that, if a user has not assigned a default rubric yet, the constructor had a string to decide the default rubric text for each rubric.

Another data structure used by the system was for an email recipient. Each person has a name, email address, if they have been emailed, and their type. The type could be either a secretary or an external examiner. I made the emailed attribute a string opposed to a Boolean to save a slight amount of code when, instead of making it if the Boolean was true then "Yes" would be displayed, it could just be set as "Yes". These details would be needed for the user to be displayed in the emailed table so that they could understand who they were emailing and if they have been emailed.

All of these structures had to implement serializable so that they could be stored in a file for later use as, if one did not, an error would appear. This caused a problem at one point where I didn't realise the HonoursRubric class did not implement serializable so my Honours exams could not be serialized to a file.

8.2.2. Interface Infrastructure

The main Java class is used to initialise the application's overview FXML file. The start method is used to call the initial application stage. Firstly, it reads in from the default Honours path and checks if the file is empty or not. If it is not empty, a file input stream is created and an object input stream is used to read in serialized objects from this file stream. Then, the default Honours rubric that the entire system can then use, is set to the object from the file. This is also done for the default Sub-Honours rubric and exam list, except for the exam list where an FXCollection is made from the ArrayList in the file which is needed by Javafx to be used in a table view.

The primary stage was then set and the title to be displayed for the application was chosen. An on close request event was needed so that the information was not lost when the exam was closed as before all the exam details would not be saved within the application. Therefore, this event was needed to loop through and add all the exams within the exam list to a new array which could then be saved to the default exam file. This is another reason why an exam needed to store the list of secretaries and examiners so that they were not also lost.

The initRootLayout() method was then called to load the overview FXML file and then set its controller with the primary stage being the main stage. The application could then be shown to the user. Methods were needed for the exam list and primary stage so that other controllers could access these.

Controller
The controller class was used as a controller for the overview FXML file in the application. This controller, firstly, is initialized to set the exam table column contents to each exam's name in the list by converting it to a Simple String which is needed for JavaFX.

Two listeners are also created for when a different exam is selected in the table which call a different method each. One calls the showExam method, which sets the text field to the chosen exam's text and resets the rubric label. The other called the showEmails method. This creates a collection for JavaFX to use based off the currently selected exam's list of people that can be emailed. Then, the emailTable could be filled with this collection and each column's information was set to the relevant field.

The rest of the methods within this class were to be executed for when each of the menu items were chosen. Most of these methods simply loaded the relevant FXML file, created a new stage with the correct title, loaded the controller class associated with the FXML file, and finally showed the stage, waiting for its response.

For emailing a secretary or examiner, it was also needed to set the text of the emailTo text field as an extra feature for the user and showEmails was needed to be called again so that it would update the table. Of course, some of these options needed an exam to be selected, so the application would show an error letting the user know that an exam would need to be selected before continuing.

Although, there were certain menu options that did not need a new window so they were implemented in this class. The about menu would simply send an alert detailing information about the application.

For saving normally, if the currently selected exam had not been saved, the save as FXML file would be displayed and its associated controller loaded. Otherwise, it would save the contents of the current exam to the text in the text area and check the file extension of the exam. For a latex and normal text file, it would simply write to a file the text contained in the text field with the right extension. A word document would require a new XWPFDocument to be created with a paragraph that is set to the exam's text and then written to a file. Deleting an exam, or a secretary or examiner was done by sending an alert to the user and if the response was ok, it would be removed from the respective table.

Checking an exam had the correct amount of marks compared to its set rubric proved to be difficult. A pattern was created to resemble the form of [?? mark(s)] with the question marks being a group of numbers. A string was also created that would be eventually output to the rubric label. Two matchers were needed, as if only 1 pattern was found the total marks per question would behave differently. A loop was created to iterate through the number of questions in the exam.

Then, a check was made to see if the text contained a question and was followed by the next sequential question. If so, a string was created as a substring from the text between these questions. The matchers then checked for any instances of the pattern in this substring. The first counter would count the number of times the pattern was found. The second matcher would then use this count to decide what would be displayed in the label for the current question.

If the count was more than one, a check was made to see if the pattern matched the total number of marks for that question. If it did, then it could be ignored, otherwise it would be added to the current mark. This was implemented for situations where an exam likes to total the number of marks of the question at the end of the question so that this should not be added to the total. Unless, of course, there is only one part to the question.

Once all the matches had been added to the current mark, it was added to the output string to display the amount of marks found for that question. This current mark was then added the total which was displayed at the end to show how many marks out of the total the exam had. This would be done until the last question where the only difference this time was the substring from the main exam text had to be done from the end of the string instead of from the next question. If a question was not found in the exam, the questions total mark would simply be set to 0.

DefaultEmailController

This was simply to set the default email address and default email kept by the system. I decided to add a default email later as the same person would most likely use the same address for the application so it would save some typing for the user.

I also later realised that my default file paths contained my local files in the path name, so installing on another machine would cause an error. This was changed in all classes to simply be the folder above the current one. The controller itself was initialised to read in the text from these files and set the stages contents to these strings. Then, if ok was pressed it would write to the default files the text entered within these fields.

DefaultRubricController

To set the default rubrics, firstly the class is initialised to add the Honours and subHonours radio buttons to a toggle group. Without this both could be selected at the same time causing confusion to the application. A listener was then added for when the toggle group was changed. If the Honours rubric was selected, the rubric text area would display the current default Honours rubric and the same logic goes for selecting Sub-Honours.

Another listener was also added for when the number of questions was changed. This was added as discussion with my supervisor brought up the idea of automatic numbering for the system. If the user changed the number contained within the number of questions any text after the first question would be removed and text would be appended based on the new number of questions.

It was a tough decision to implement it like this as there was worry if a user would have entered in a lot of text between questions. I then decided that for the default rubrics there would not be much text, if any, between questions so this would be sufficient.

If ok was pressed, the application would first check that all fields had valid input by calling the validInput method. This would generate an error message for an alert to be displayed if certain conditions were not met.

A condition to be noted was if the marks per question contained a list of numbers in the form ?,?,?. If this was met, it would then count the amount of numbers by splitting the string by the commas and checking if this number matched the total number of questions, otherwise there would be an error.

If the input was valid, the relevant information was gathered that was required for the rubric data structure. Then, depending on the radio button selected, the system would set the default rubric for the application to use and save this to the corresponding default file location.

SecretaryExternalExaminerController
Adding a secretary or external examiner required the person's name, email address and whether they were a secretary or examiner. The controller was initialised to have a toggle group for selecting the secretary or examiner radio buttons. If ok was entered, a check was made for valid input similar to before.

It would specifically check if a valid email was entered by using a reasonably thorough regex expression that should cover basic email addresses. An email recipient would then be created if all fields were entered correctly which would then be added to the currently selected exam's list of recipients.

EmailSecretaryOrExaminerController
When the stage is created, it firstly sets the email text area to the default email and the email from field to the default email address. The attachment text field is also disabled so an invalid file path could not be typed by the user.

If the browse button is selected, a file chooser is opened for the user to choose an attachment. The attachment text field is then set to the user selected file and was implemented to still be disabled so that the user can't accidentally change the chosen file.

Again, once valid input was entered after ok was pressed, the email could then be created. Properties are then set up for live host as I was only able to implement sending emails using outlook addresses and not for Gmail which will be discussed later. The session then authenticates based on the email address and password.

A message is created from the email text, the subject, and the recipient for the message to be sent. If there was an attachment selected, a multipart message was created. Part of this would contain the email text as a body part and the other would be based on the file location selected by the user to create a multipart message.

Once the message had been made, it could be sent along the session that was initially created. Then, the recipient that was selected to be emailed had their emailed property set to Yes. There is not a way to refresh a table using JavaFX so the implementation had to think of a way around. This was done by setting the columns to not visible and then immediately visible again as a brute force solution to the problem.

NewController
This controller was needed for when the user wanted to create an entirely new exam altogether. A toggle group was created so the user can select one of the three rubric options. If a user selected to create a new rubric, they would have to enter the entire details of this rubric.

If ok was clicked, once again valid input would be checked for the fields. A later addition, due to supervisorial discussion, was to allow the user to enter the parts for a question. A check was needed to make sure this was entered in a valid list format and contained the same amount of questions as the number of questions.

This helped notice a problem in the implementation where if the marks or parts per question was checked without a number of questions entered an exception would appear that was not handled by the application. This was fixed so that a check was made beforehand as to whether the number of questions text field was empty. It was also important to implement these checks so that the parts per question did not have to be entered in but if it was, it should be in a valid format.

Once the input was validated, a new exam could be created using the user inputted name. If one of the default rubrics was selected, the exam would be created with the application's default Honours or Sub-Honours rubric.

If the user decided to add parts per question, then it would be split by the commas to gather the number of parts for each question. If the part was equal to one, then no parts would need to be added so the loop could just continue. Otherwise, strings containing a value starting at the character ‘a’ and the final text to insert were created. It would loop through the number of parts, increasing the character value by 1 and appending each character in brackets to the text.

These characters would then be inserted after the associated question within the default rubric text and was done for every question. The final text created by this was entered as the exam text and the final exam was then added to the exam list. If a new rubric was selected with parts, then it would input to the exam text just the questions and parts. If no parts, the questions would simply be added.

UploadController
Uploading an exam was very similar to creating a new exam. A user would still need to enter a name and select or create a rubric for the exam to use, except this time the exam text would be the text from an uploaded file.

The browse button acted the same way as choosing an email attachment except a user could only choose between txt, pdf, docx, and tex files to be uploaded. Once the input was validated as usual, an exam was created from the user input. The text for the exam was made by calling the getUploadText function.

This would firstly find the file's extension and act accordingly. For txt and tex files, the file would simply be read to a string. A pdf file would be read by a PDFReader and the number of pages would be counted. For each page the text would be extracted, appended to a string, and then this final string was returned. A docx file required a POITextExtractor that extracted the text from an XWPFDocument created from the given file. Then, this new exam could be added to the exam list.

SaveAsController
A toggle group of file types was firstly initialised for the controller. If the user did not select one of these file types or an exam to save, an alert error would appear. Otherwise, an extension string was initialised based on the file type chosen.

Then, a file chooser was created with extension filters of the possible file types. I implemented it like this so that a user could filter out the folders to only see the files of the same type so that they don't accidentally overwrite a file of the same name. If the file was not null, the currently selected exam would have its contents saved as the current text, and also its file location, type, and would have its isSaved value to true.

The exam text would also be saved to a file by calling the saveFile method using the exam contents, file, and extension. For a text and latex file, this method would simply write the text to the designated location with latex files having a tex extension.

A docx file would be saved by creating a new XWPFDocument and creating a temporary paragraph for it. This paragraph would have the exam text added to it and then the document was written to a file output stream with the docx extension.

As mentioned before, it was decided that implementing saving to pdf files would not be worth doing as a pdf of simply text was not really required. However, the unfinished code that was attempted early on was left in, incase future work meant pdf files would be worth saving to.

SpellingController
This controller was based off the article written by Jeramy singleton [6] which had errors that needed fixing and editions to fit with the rest of the application. Firstly, the ignore and change buttons were disabled so that they could not be pressed until the text was searched as before the user could break the application by pressing these.

Then, once the search text button was pressed, the buttons were enabled and a string was set to be the exam's text. This string would be used for the replacement text for an exam. The possible changes table was set to the list of suggestions and a listener was added for when a possible change was selected, the changeTo text field would change to the suggestion.

The text was searched for any mistakes using an American English JLanguageTool. For some reason, normal English was just for grammar so American had to be used for spelling mistakes. If no errors were found an alert was displayed letting the user know no errors were found. Then, all the misspelled words in the exam text were added to a new list and the nextMisspelledWord method was called.

Before, there was no indication that all the mistakes had been looked at so an alert was added to indicate to the user this. The first mistake was displayed and any suggestions for this word were added to the suggestion list. Originally, there was no indication that there were no suggestions available and the application would fail, so I added to the options "No suggestions available" and disabled the table so that sentence could not be selected by the user. Otherwise, the first word in the suggestions would be displayed to the user.

The user would then have the option to ignore, ignore all, change, or change all. If ignore was chosen, the position was moved on and the next word was looked at. By pressing ignore all, all the buttons would be disabled and fields cleared to the user.

The change button would call the changeMispelledWord method with the current misspelled word and the changeTo field's text. Within this method, the replacement string had any contained mistakes replaced and the position moved on.

This caused a lot of trouble at first as if a space was missed before punctuation, replace all would behave very strangely and replace everything or crash. This is because the replace all uses regex and certain punctuation means something different.

For example, a full stop in regex means any character, so if a space was missed beforehand and the user decided to change this, all spaces and its following character were replaced with a full stop. This was solved by adding separate conditions with escape characters.

Change all would simply call this method for all mistakes and replace them with the first suggestion. The first algorithm did not account for zero suggestions so a check had to be added to combat this. Then, when ok was clicked, the exam text area was changed to the new replacement string.

9. Evaluation

9.1. Comparison to Objectives and Requirements

A good source of evaluation is looking at the requirements and the objectives set out in the DOER document at the start of the project. However, the project's original goals shifted around during the year with original objectives changed and more added as goals start out reasonably vague for such a large project. Looking at these goals, it is possible to see if the project as a whole met the specification it set out to complete.

All of the primary objectives were met considering any changes to them, all secondary objectives were met, and half of the tertiary objectives were met. Also, looking at the requirements, most of these created were completed. As a result of the focus on parts of the system moving, some of these had to be omitted.

Primary Objectives

The primary objectives were achieved almost in all of their entirety. Any changes to the original objectives are noted below with some additions.

1. *To have a tool which gives the ability to write and format an exam paper ensuring that there is a consistent page layout for all papers.*

   This is apparent from the design and implementation of the exam application where it is possible for a user to write and make changes to an exam. Although there are not any advanced options for a user to change the page layout, any exams created within the system have the same page layout keeping them consistent for all papers

2. *The paper should be able to be saved as pdf or document so it can be integrated with different tools.*

   This objective was discussed with my supervisor and it was agreed that the need to save as a PDF file was not needed for simply just text. However, the ability to save text and Latex files was added to this. It is possible to see in the project folders, in the out folder, example documents being created within the system of file types docx, tex, and txt. It is also important to note that PDF file type was still allowed in the system, although not fully functional, which was planned for future work and currently this simply stops a general release.

3. *The tool should store the rubrics of papers that can be used to automate part of writing the paper itself and have the possibility to change these if the rubrics change.*

The application stores the rubrics of the papers as seen in the data structures section and with the use of the described default rubrics controller can automate text for the paper. As mentioned in the default rubric controller section these can be changed at any time if the rubrics change.

4. *Then, it should be possible to check whether the marks written in the paper match the stated rubrics.*

   The application has a menu item which is used to check the marks within a paper as mentioned in the Controller part of section 8.2.2. This allows the user to check if the marks written in the paper match the rubric set to that exam as the objective requires. There is a minor issue where if an exam ends a sentence with a number, for example "4.", then the algorithm gets confused but, when looking at past examinations, this was a very rare case with it only coming up once and again only puts back a general release.

5. *Automatic numbering and parts per question should be added to the exam paper text.*

   When discussing with my supervisor midway through the project, it was asked if the ability to add automatic numbering and splitting the question into parts would be possible so I added this as a primary objective. This was added to the application for when a new exam was created where question numbers would be generated and the parts for each question would be input and then generated as described in the new controller part of section 8.2.2.

Secondary Objectives
This objective was fully met.

1. *Spell checker to make sure everything in the paper that has been written is spelt correctly.*

   Selecting the spell check option in the application's menu opened a spell checker dialog to allow the user to go through any spelling and grammatical errors within the text and then correct them if they so desire. As described in section 8.2.2. in the spell check controller subsection, this details the implementation behind this objective's completion.

Tertiary Objectives
Half of these objectives were met with the others omitted as they were deemed to be out of the scope.

1. *The ability to set external examiners to an exam paper and automating part of this section of the exam preparation stage such as sending out an email to the given external examiner.*

It is possible for the user to set an external examiner to a paper in the application as mentioned in the set secretary or external examiner controller portion of section 8.2.2. Also, they can then decide to email the said examiner as the objective requires. However, although this was not specified during the project, emailing would have been more ideal for Gmail as opposed to Outlook.

2. *Also, the ability to set a secretary to a certain exam paper.*

   This objective was bundled with the previous objective as it was realised they both require similar functionalities. The mentioned sections also apply to this objectives completion.

3. *The tool should make sure the transmission and storage of papers is secure.*

   This objective was considered to be a different area of research that could be looked into as a completely new project and not a priority for this system. Therefore, this was not carried out as a result. However, it would have been great to keep papers secure locally and for emailing them with solid security. Although, in discussion in user testing, it was brought up how papers are usually encrypted by the administrator before transmission, meaning the importance of this system including this would be a novel feature.

Requirements
The requirements created originally are basically the objectives broken down into smaller, more manageable sections with a few added features and also give the rationale behind them so I could tell why they were there. As a result, these requirements were mostly met with the exception of a few as described in the objectives above.

9.2. User Testing
User testing was managed by simply finding volunteers to try out the application, talk them through it, and then discuss the application and the topic. Although, it appeared to be a busy time so finding many relevant subjects was difficult. This was mainly carried out to look at future possibilities of the system and what the users would like to see added so that improvements could maybe be made if required.

When speaking to an exam administrator they thought it was good to have the exams being sent firstly as raw text so that the important information was there. This could then be checked by the system so that the main body text was valid and then tidying up the papers could be done elsewhere. Users also liked the idea of having secretaries and examiners available to view which could maybe be expanded so that more people could be added to get the whole overview of the exam process.

One of the main points that users would like to see if the addition of graphics or diagrams to the application. Although, having raw text is sufficient enough for the creation process as other applications can be used to improve the format afterwards, diagrams are very common part of exam questions so it would be important to have this functionality. Another idea was to have the ability for multiple users to access this exam. This would mean people could collaborate on the process and look into what still needs to be done.

10. Discussion

10.1. Impact

As an idea, the exam application designed and created here gives a new vision into the exam paper process. Some of the decisions made can spring up some thoughts into the creation of a paper and make exam writers think about if what they are writing follows the previous paper's structure.

Although the application itself is not exactly the next Microsoft Word and would take a lot to overthrow the throne this and other documentation creating applications, with many people working on them, have on the market. The system was purely made for the intention of looking into a new field of research that had not necessarily been done before and was not planned for public use. It has certain flaws that have been discussed that would need to be reviewed before it would be considered over something like Word but, hopefully, gave an interesting perspective of the exam paper creation process.

10.2. Future Work

The exam application created is just a starting place with many future possibilities available. There are certain components that would need to be addressed for public use and further research could be done to further enhance the experience.

Some application issues would need to be refined before it can be used to its full potential. For example, as mentioned in user testing, the text area is very limited with only textual input available. This would be nice for users to be able to add their own graphics and diagrams to the exam. Also, being able to have more options to format the text would be ideal. This is why a user may prefer to use document creation applications such as Microsoft Word which very unique features, specifically when it comes to exams, are needed for the application to be used for writing papers over other systems.

Another issue that was not really touched was the CSS of the application. Focus was mainly put on functionality and not making a state of the art interface. However, it would be nice to see a slicker, more appealing style for the application which may encourage the user to try the system out. Even changing the colour of, for example, the rubric marks to let the user know the question had the correct amount of marks. Another reason this was untouched is that, sometimes, when I start messing around with styles and colours it can become more hideous than it originally was so I didn't want to ruin the simplicity.

A related issue for user friendliness would be to add some form of help documentation within the system as to how to use the system. Although the interface is very simple to use with guidance errors, some things may not be clear to a first time user so the addition of this help option would benefit them.

The emailing within the system ended up having to use the live message transfer protocol where Gmail would have much rather been preferred. This was a result of google now requiring applications to use two step authentication when sending an email. There has not been many, if any, examples of this being used in Java. However, I did some research into this and found out about the Gmail API which could be used for a two-step verification application [8]. Although, I could not get this fully working so I removed the messy code and went with Outlook instead. By creating an extra credential, the user can then be verified fully conforming to the new Gmail rules.

The security of the transmission of these papers could be done in the future, especially for unencrypted papers being sent across a network. Security is vital for exam papers so it would also be nice for the application to conduct its own encryption for when papers are saved which was mentioned during user testing.

Other future ideas could maybe be possible to give a better overview over a specific exam. The rubrics are currently very limited, therefore it could be possible in future to be able to select from a table of modules with the set rubrics for each one meaning more consistency between papers, especially for the varying Sub-Honours modules. Then, more text could be generated such as the exam title and time to complete.

As mentioned in user testing and was not originally thought of, allowing multiple users to access a paper would allow more collaboration for an exam paper. They could then see the paper's progress and let them know if anything was potentially wrong. People would be able to check others assigned to the paper and their roles or contact details and maybe more roles would be beneficial to be added to the application for a larger overview of the entire process.

10.3. Reflection

The whole solo project experience has taught me a lot and helped me to grow, not just in my coding abilities, but also my management and mental discipline. I have learnt the importance of planning things piece by piece instead of as one entire entity. Having supervising meetings really helped achieve this as plans could be made for each week and any problems I was having could be discussed. Each problem could be split into weeks rather than the whole year.

A significant drawback to my project was trying to get the Eclipse Rich Client Platform to run on the system which put the project back a few weeks as I could not set it up without it crashing and, when helped was asked for, they were also unsure. This and having personal issues, lowered my work ethic at the start of the project which was hard to overcome. However, I learned the importance of getting help when you really need it. There are so many people out there who are willing to help but only if they actually know about the problem. After this, it was easier to get past the lack of code within the first semester, which seemed to be a problem with most students, and structure each week to get the project complete, improving my mental strength and management abilities.

My programming skills also increased as a result and strengthened my knowledge with Java. I had never really made an extensive application before but, after the project, I feel comfortable with using JavaFX and Scene Builder to create an application from scratch. Also, learning how all these components interact with one another really helped cement by Java ability. Even learning about emailing in Java, although it was not implemented as originally planned, I learned a lot about the verification process and the 2-step verification Gmail now employs. The ability to be able to create applications like this is valuable information to have.

I also learned the importance of having readable and maintainable code, as it makes going back to old code a lot easier, especially for such a large system. Of course, the code will not be perfect and would need some cleaning up. Overall, adding parts to the system was really simple and did not break other sections due to keeping to these methods.

11. Conclusion

This project detailed the steps taken into creating an exam building application and the ideas for improving the exam paper writing process. It looked to be integrable with existing document creation systems such as Word or Latex editors so the files could be uploaded and saved in these file formats. However, the project was not as refined as these applications but the main goal was to focus on the exam aspect of document building.

The application manages to succeed in certain aspects such as its rubric and spell checking options which are useful for testing the validity of a paper. It provides a new look into the exam paper process that nobody has really looked at before. The system can help cut down the writing time of a paper and gives flexibility on what text to automatically generate. Hopefully, it also gives a user a slight overview of the current exam paper's process and what needs to be done.

However, the system is not perfect and, as with most new applications, improvements need to be made for future work. The limited text formatting with no option for graphics or diagrams would need to be changed for a better exam preparation experience. Although, the application takes a small step in doing this and if it was trying to be some form of Word clone, then why not simply use Word?

It is hazy the direction this project will take but when looking at the future work discussed it could possibly be a useful tool for the world. Exams are not usefully fun for anyone, students and lecturers alike, but hopefully the project can help to lessen the pain for the creation of them.

References

[1] Best Document Creation Software, [Online]. Available: https://www.g2crowd.com/categories/document-creation

[2] JavaFX and Scene Builder, [Online]. Available: http://docs.oracle.com/javase/8/javase-clienttechnologies.htm

[3] Javax Mail, [Online]. Available: http://www.oracle.com/technetwork/java/javamail/index.html

[4] Language Tool, [Online]. Available: http://wiki.languagetool.org/java-api

[5] XMLBeans, [Online]. Available: https://xmlbeans.apache.org/

[6] Singleton, J. (2015, June). Spell Checking in a JavaFX App. [Weblog]. Retrieved 18 January 2016, from http://www.jeramysingleton.com/add-spell-checking-to-a-javafx-app/

[7] PDFBox, [Online]. Available: https://pdfbox.apache.org/

[8] Gmail API, [Online]. Available: https://developers.google.com/gmail/api/quickstart/java#step_3_set_up_the_sample

[9] Pilkington, C., & Sanders, I. (2014). An online collaborative document creation exercise in an ODL research project module. *Computers & Education*, *77*, 116-124.

[10] Schultz, D. W., Schectman, H. R., Hay, J. A., Thompson, M. P., Wallace, K. G., & Kusmer, S. R. (2015). *U.S. Patent No. 9,152,728*. Washington, DC: U.S. Patent and Trademark Office.

Appendices

Appendix A: User manual

The code has been submitted along with this report contained in the ShProject folder and can be run in your own java IDE or the command line if required. Running the Main java class should load the application.

The system itself is very simple to use as the menu item names explain how they function. Although, one thing to note about the system is that a user should select a secretary or examiner first before emailing them, however an error is supplied if this is not acted out. Also with the spellchecker, it may seem to be displaying blank sometimes which is for a double spacing error which confused me at first too.

Appendix B: Testing

As the testing was mainly for the interface, testing was carried out by checking vigorously the input into each possible field and sequence within the interface to see if any errors were apparent and the output was correct. Every time a new component was added it was made sure to give the correct graphical response based on the input and make sure the rest of the system was still valid.

For the saving of files, a folder was created to check whether files being created were not corrupted and contained the correct contents. Also, if the exam was saved it was tested if these files were also being saved to. This was similar to the email component where I checked that the emails I sent were actually being sent to myself with an attachment if it was added by looking in my inbox.