

# **OCR Computer**

# **Science Coursework**

**Jude James**

**Candidate Number - 6120**

**Centre Number - 11022**



# Table Of Contents

<b>1.1 Features that make the problem solvable by computational methods</b>	<b>3</b>
<b>1.2 Suitable stakeholders</b>	<b>4</b>
<b>1.3 Information gathering</b>	<b>4</b>
1.3.1 Planning	4
1.3.2 Gathering	5
<b>1.4 Existing solutions</b>	<b>5</b>
1.4.1 Existing Solution one	5
1.4.1.1 Pros and cons	9
1.4.2 Existing Solution two	10
1.4.2.1 Pros and cons	12
<b>1.5 Features of the proposed computational solution</b>	<b>13</b>
<b>1.6 Limitations of the proposed computational solution</b>	<b>15</b>
<b>1.7 Requirements for the proposed solution</b>	<b>15</b>
1.7.1 Hardware requirements	15
1.7.2 Software requirements	16
1.7.3 Design & functionality	17
<b>1.8 Success criteria</b>	<b>26</b>
<b>2.1 Decomposing the problem</b>	<b>29</b>
2.1.1 Decomposition Diagrams	29
<b>2.2 Structure of the solution</b>	<b>31</b>
2.2.1 Entity relationship diagram	31
2.2.2 Normalisation	32
2.2.3 Definition of data requirements (design data dictionary) and validation required	34
2.2.4 Interface design	38
<b>2.3 Algorithms</b>	<b>46</b>
2.3.1 SQL Queries	46
2.3.2 Algorithms	50
<b>2.4 Usability features</b>	<b>58</b>
<b>2.5 Key variables &amp; data structures</b>	<b>59</b>
<b>2.6 Test data during the iterative development</b>	<b>64</b>
<b>2.7 Further data to be used in the post development stage</b>	<b>75</b>
2.7.1 System testing	76
2.7.2 User testing	78
<b>3.1 Stage 1 Database</b>	<b>79</b>
3.1.1 Prototype 1	79

3.2 Stage 2 login form	81
3.2.1 Prototype 1	81
3.2.2 Prototype 2	84
3.2.3 Prototype 3	93
3.3 Stage 3 sign up form	101
3.3.1 Prototype 1	101
3.3.2 Prototype 2	105
3.3.3 Prototype 3	130
3.4 Stage 4 home page form	143
3.4.1 Prototype 1	143
3.4.2 Prototype 2	151
<b>3.5 Stage 5 Students page</b>	<b>157</b>
3.5.1 Prototype 1	157
3.5.2 Prototype 2	165
<b>3.6 Stage 6 Student performance page</b>	<b>173</b>
3.6.1 Prototype 1	173
3.6.2 Prototype 2	181
<b>3.7 Stage 7 Grades page</b>	<b>189</b>
3.7.1 Prototype 1	189
3.7.2 Prototype 2	200
<b>3.8 Stage 8 Reports page</b>	<b>209</b>
3.8.1 Prototype 1	209
3.8.2 Prototype 2	218
<b>3.9 Stage 9 Edit details page</b>	<b>226</b>
3.9.1 Prototype 1	226
3.9.2 Prototype 2	232
<b>3.10 Stage 10 Generate report page</b>	<b>242</b>
3.10.1 Prototype 1	242
3.10.2 Prototype 2	249
<b>3.11 Stage 11 Report generator page</b>	<b>256</b>
3.11.1 Prototype 1	256
3.11.2 Prototype 2	265
<b>3.12 Stage 12 Final paragraph page</b>	<b>287</b>
3.12.1 Prototype 1	287
4.1 System testing & end user testing	292
4.1.1 System testing	292
4.1.2 End user testing	311
4.2 Evaluate the solution	314
4.3 Further development	317
4.4 Effectiveness of usability features	318
4.5 Limitations and potential improvements	329
4.6 Maintenance issues	330

# 1. Analysis

## 1.1 Features that make the problem solvable by computational methods

There are over one million students across the UK in secondary education. Teachers must create and provide reports for all their students, these consist of grade reports and written reports. The written reports must be a unique, personalised paragraph or two sent to parents talking about how their student is performing, their behaviour, grades, etcetera. A teacher may have to write up to 100 reports depending on the number of students they teach, this process has become tedious, time consuming and hard to manage. This has been identified as a problem and the process needs to be improved. Teachers have contacted me hoping to design a piece of software that will allow them to generate written reports for their students quickly and easily. This software will be referred to as a solution as it is solving the problem teachers are having. The solution will allow teachers to generate reports for their students, it will allow them to view and manage their students' important information such as grades, subjects, attendance & punctuality. Some advantages of this being a computation solution is it will include a graphing tool to easily view student data and compare between classes or subjects. It will also include a table view to scroll through student data easily. It will include login functionality for teachers to log into their account and view their students only. These features make the problem solvable by a computational method because the solution will contain a central database storing student information, extending this database will be other tables storing teacher login information, students grades, previous reports. Other computational methods involve the use of sorting algorithms so teachers can sort and filter through their students, the reports they are working on, or classes they teach. To create the written report the solution will use an algorithm that will adjust the tone of the report based on simple input values the teacher enters or values that already exist in the database. The solution will make use of decomposition and procedural programming by breaking down large processes like updating a database with new information into smaller sub-processes. Abstraction will also be used to increase the safety of the program and reduce complexity, increasing efficiency.

## 1.2 Suitable stakeholders

The main stakeholders for the solution are teachers, this software will benefit them the most because they have to write the reports for their students and view their grade information to focus on helping them reach their targets. Higher positions within schools can also use this software, head teachers or principals can view the students' performances across all subjects and see where improvements can be made within the school. Mr. Nicolston is a head teacher at Trentlock Secondary School who initially contacted me with the idea of creating a new software.

Stakeholder	Position	Interaction
Ms. Armstrong	Admin	Managing the school database, adding/removing and organising data.
Mr. Wadud	Science Teacher	Logging Into his account. Creating reports for his students and saving them to the database.
Mr. Nicolston	Head Teacher	Overviewing the databases, seeing which subjects or students are falling behind so improvements can be made.

## 1.3 Information gathering

### 1.3.1 Planning

I will carry out two interviews with the head teacher Mr Nicolston. One interview will contain questions about the existing solution to identify problems he and the stakeholders are facing. The second interview will contain questions about the new solution to gather information on the specific requirements and exact needs they have. On top of this I will create a questionnaire for various staff members to fill out to gather more information.

### 1.3.2 Gathering

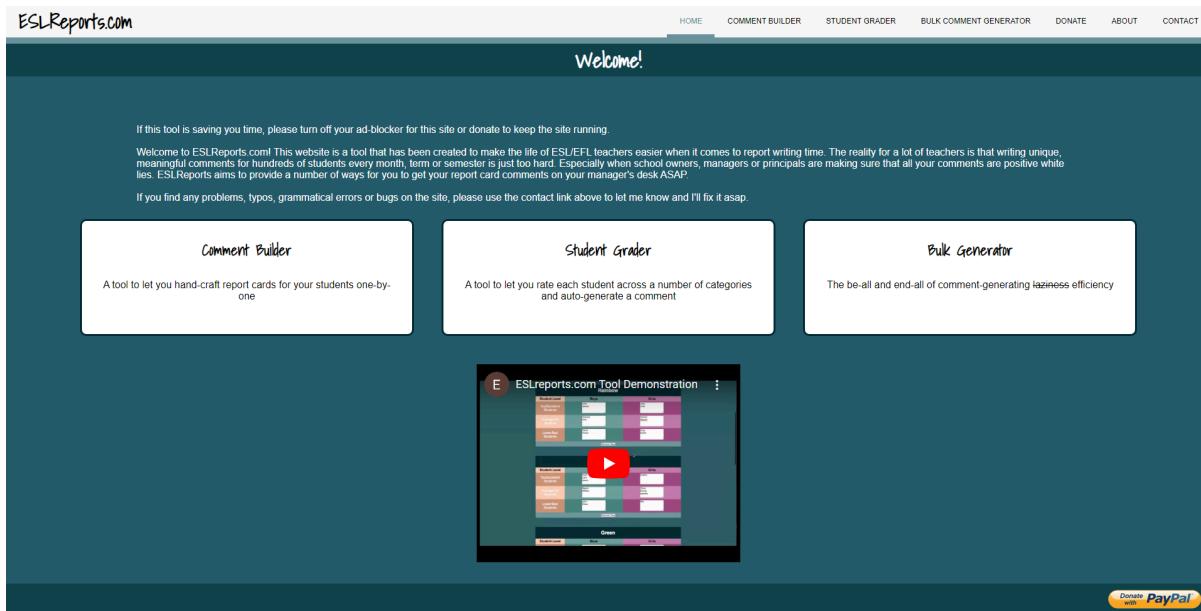
I emailed Mr. Nicolston, we exchanged information and arranged a time and place to meet for an interview on the existing solution. The interview was successful and I gathered useful information on the problems the existing solution is having and their frustrations with it. I asked them several questions to help me understand what the existing solution offers and what it doesn't. Following this I carried out a questionnaire for the staff members. This allowed me to get quick feedback in the form of short answers or tick box questions to see what the overall opinions are or what the majority of people want from this solution. Finally I had a second interview with Mr. Nicolston focusing on the new solution. The interview contained a series of questions that focused on what Mr. Nicolston wanted it to do. He told us the specifics like what colour the background should be, what font the text should be, the scaling of the user interface. All decisions when designing and implementing were impacted by Mr. Nicolston.

## 1.4 Existing solutions

There are two existing applications that allow teachers to generate a written report for their students. I will look at each existing solution and identify the features that I can adopt and the features that I can adapt to make the software better.

### 1.4.1 Existing Solution one

The first existing solution is a web application called 'ESLReports.com'. This website focuses on teachers who speak English as a second language (ESL) and helps them generate a report for a student. The website has a simple user interface, it consists of a blue background throughout every page and basic white colour for the text, text boxes, radio buttons and other simple interactables. The main page contains three main sections that you can click on, 'comment builder', 'student grader' and 'bulk generator'.



The first section to look at is the comment builder. Clicking the button takes you to the next page. This page, like all the pages, is consistent with the first page in design and functionality. Once on this page it gives three presets: 'Anything Goes', 'Mostly Positive' and 'Only Positive'.

- Click on one of the buttons at the top of the page to change what kind of comments are displayed in the dropdown boxes below.
- Type the name of the student and select their gender.
- Optional: Type the name of a subject you have been studying in class if you want to include a subject sentence.
- If you're unsatisfied with the comments available for any subject, click the refresh button next to that type.
- Click on each of the dropdown boxes and choose a sentence for each category that you want to include in your report.
- **IMPORTANT**
  - Press the *Add to Comment* button for each option you want to use in the order you want them to appear in the comment.
  - Press the *Finish Comment* button to replace all the pronoun and name markers with the correct values.
- Press the **Clear Comment** button to clear all data and begin again.

*Click to show/hide instructions*

When you click on one of these buttons it extends the page to show more options, this is where the teacher can start to create the report. The first part is the student details section which requires the teacher to enter the student name (string) , gender (boolean) and subject (string).

Student Details	
Student Name:	<input type="text"/>
Gender:	<input checked="" type="radio"/> Male <input type="radio"/> Female
Subject:	<input type="text" value="eg. Phonics"/>

The second part is the comment selection section, this is where your choice earlier makes a difference. The comments depend on if you picked 'Anything Goes', 'Mostly Positive' or 'Only Positive'. The comments are split up into different categories: introduction, behaviour, speaking, reading, writing, listening, comprehension, subject and conclusion. The teacher can then click a randomise button icon to change the sentence in the category until they are happy. Then the teacher clicks the 'add to comment' button to add that sentence to the final paragraph at the bottom.

Comment Selection - Anything Goes		
Category		
Introduction	 (name) is mostly well behaved and attentive in class and is happy to help other students when they need assistance.	<input type="button" value="Add to Comment"/>
Behavior	 I would like to see (him/her) settle down and pay more attention.	<input type="button" value="Add to Comment"/>
Speaking	 (He/She) has quite a strong vocabulary and (his/her) answers to questions are usually accurate.	<input type="button" value="Add to Comment"/>
Reading	 (name) is able to read at an appropriate level for this class.	<input type="button" value="Add to Comment"/>
Writing	 (name)'s writing is good, but I have told (him/her) to improve (his/her) handwriting as sometimes it is quite unreadable.	<input type="button" value="Add to Comment"/>
Listening	 (He/She) has a lot of issues understanding people who speak with different accents.	<input type="button" value="Add to Comment"/>
Comprehension	 (Name) has a reasonable understanding of the materials we have studied until now.	<input type="button" value="Add to Comment"/>
Subject	 (Name) has proven to have only an elemental understanding of the (subject) material we have covered in class.	<input type="button" value="Add to Comment"/>
Conclusion	 (name) is very keen and is progressing rapidly. Keep up the good work!	<input type="button" value="Add to Comment"/>

The final part is the comment section. There is a textbox at the bottom of the page with the final written report that you can edit if you want to make final changes.

Comment	
<input type="text"/>	
<input type="button" value="Finish Comment"/>	<input type="button" value="Clear Comment"/>
Press the Finish comment button to replace name and pronoun markers in the comment with actual values.	Press the clear comment button to clear all text and begin again.

The comment builder is the section that relates mostly to my solution but the other two sections do similar things. The student grader requires the teacher to enter the same field again and then click several radio buttons which have the same nine categories as the comment selection section. The radio buttons range from 1 to 5, with higher being more positive. Then the teacher clicks the generate comment button below and the report is made. This is similar to the comment builder section but gives less control, which is why it will be ignored as it is a more limited version.

Student Name: [REDACTED]

Gender:  Male  Female

Subject: eg. Phonics [REDACTED]

Introduction:	<input type="radio"/> N/A <input checked="" type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5
Behaviour:	<input type="radio"/> N/A <input checked="" type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5
Speaking:	<input type="radio"/> N/A <input checked="" type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5
Reading:	<input type="radio"/> N/A <input checked="" type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5
Writing:	<input type="radio"/> N/A <input checked="" type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5
Listening:	<input type="radio"/> N/A <input checked="" type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5
Comprehension:	<input type="radio"/> N/A <input checked="" type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5
Subject:	<input type="radio"/> N/A <input checked="" type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5
Conclusion:	<input type="radio"/> N/A <input checked="" type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5

The last section is the bulk generator, this generates multiple reports in one go for six students. The teacher must enter the class name, then enter the names of three boys and three girls in their class each with one of three levels of behaviour. This again is a more limited version of the comment builder and only creates reports for six students.

Add Class

Class Name: [REDACTED]

Year 7			
Subject <input type="text" value="eg. Phonics"/> (Optional)			
Student Level	Boys	Girls	
Top/Excellent Students	<input type="text"/>	<input type="text"/>	<input type="text"/>
Average/OK Students	<input type="text"/>	<input type="text"/>	<input type="text"/>
Lower/Bad Students	<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="button" value="Remove Class"/>			
<input type="button" value="Generate Comments"/>			
Pressing this button will generate comments for all Classes and Students			

#### 1.4.1.1 Pros and cons

There are several problems with this existing solution. One problem identified is the categories are based on the students English language skills however many students in secondary education and sixth form take a much wider range of subjects. My solution will improve upon this by using a database with the subjects the students take and allowing any subject to be mentioned. Another problem is the limited number of fields, the teacher can only input first name, gender and just one subject. Improving this my solution will use the student database to automatically give a list of students to the teacher to pick and also provide more information like form, parent name, list of subjects, grades and so on. A third problem is the limited variety of comments a teacher can make. There are three options that determine what the sentence will be like, all of which are positive. If the student has poor behaviour the teacher cannot filter for that option. Our solution will give the teacher much more control and fine tuning the sentences based on several input values stored on the student. The final problem with this existing solution is that once a report is generated, since it's a website the teacher has to copy and paste the text into the correct software for their school. My solution will avoid this by allowing the teacher to insert the report into the database and the report will be linked to a student already.

Despite these problems, this website contains many positive features that I will aim to adopt and bring into my solution. One good feature is the ability to make final changes in the comment section before the report is sent out. This textbox is displayed at the bottom of the page and allows a teacher to make final changes or adjustments to the report, without being tied to any pre-written comments, they can

simply edit the final text and then push. A second good feature is the ability to rank students from 1 to 5 in different categories. These categories revolve around English language skills however the functionality can still be adopted and adapted to better fit my solution. Having these rankings gives a lot of control to the teacher as they will be able to determine the final outcome of the report. The final positive is the easy to use user interface, there was no confusion navigating and getting the required outcome, it worked quickly and well. I will adopt this simple design language and usability.

## 1.4.2 Existing Solution two

The second existing solution is a software named report writing assistance V2. This software must be purchased by teachers to use and it similarly generates reports for students by a system of filling in fields as input and generating a report as an output.

The main page has an overview of the different sections including student details and phrase categories. The student details are very limited, it contains the student's first name field and students gender. There is a report comment box, allowing the user to type a custom comment.

Student's First Name	Gender (M/F)	1 Select Phrase Category using Slider above
ssddd	<input type="button" value="▼"/>	Opening Comments
Select phrases from the statements below by clicking the option button at the right hand side		
<p>sss has worked very well this year, consistently producing work of a high standard.</p> <p>sss has participated well in all activities, he/she always shows a conscientious and committed attitude towards his/her work.</p> <p>sss has worked very well, displaying a keen eagerness for the work and showing a very good background knowledge.</p> <p>sss has worked very well this year, displaying a conscientious and committed attitude throughout the year.</p> <p>sss has made very good progress this year, consistently producing work of a high standard and displaying a conscientious</p> <p>sss has worked well this year, demonstrating a sound understanding of the work covered.</p> <p>sss has worked well this year, consistently producing work of a good standard.</p> <p>sss participates well in most activities, showing a good level of concentration.</p> <p>sss has worked reasonably well this year, producing work of a sound level.</p> <p>sss has shown a good level of commitment towards his/her studies and has been successful in attaining his/her target levels.</p> <p>sss has worked well this year but can be a bit reserved during discussion work, being willing to answer a question but rarely</p> <p>sss has worked well, but has struggled at times with the depth of content and when recalling specific technical terms.</p> <p>sss has worked inconsistently this year, producing work of a variable standard.</p> <p>sss has found it difficult to connect with the subject content this year, finding it difficult to maintain his/her concentration</p> <p>sss has struggled this year with the depth of knowledge required, but has remained positive in his/her approach to his/her</p> <p>sss has found it difficult to work throughout each lesson, being easily distracted and can be a distraction to others,</p> <p>sss has made less than the expected level of progress this year, mainly due to a lack of focus during the lessons and a</p> <p>sss has found the lesson content difficult to process and has struggled to remain on task, consequently he/she has made les</p>		
Report Writing Assistant		

Another page contains the student details form, it is a spreadsheet containing the fields: forename, surname (optional), gender and finished report comments. You can fill in the student details on the first page and it will populate this page, then once the report is written it will be added here too.

Forename	Surname (optional)	Gender (M/F)	Finished Report Comments
Lucy			Lucy has worked very well this year, consistently producing work of a high standard.
John			During written tasks he/she regularly asks questions about the work to extend his/her understanding and to make links with other areas of the curriculum. sss should try to add extra detail to his/her answers to justify the reasoning for his/her answers. sss should try to add extra detail to his/her answers to justify the reasoning for his/her answers. sss should try to add extra detail to his/her answers to justify the reasoning for his/her answers.
Test			
Example			

The next page is the comments bank page, it lists all the comments a teacher can add to the report in a large table. The comments are split into 3 categories shown by green, amber and red. These reflect the behaviour and performance of the student, with red giving negative feedback and green giving positive feedback. The comments are split into 6 categories: Opening, Behavioural, Progress, Strengths, Weaknesses and Closing.

	Opening Comments	Use # where you want the student's name to be inserted. >>>>Scroll Right for next comment category
1	# has worked very well this year, consistently producing work of a high standard.	
2	# has participated well in all activities, he/she always shows a conscientious and committed attitude towards his/her work.	
3	# has worked very well, displaying a keen eagerness for the work and showing a very good background knowledge.	
4	# has worked very well this year, displaying a conscientious and committed attitude throughout the year.	
5	# has made very good progress this year, consistently producing work of a high standard and displaying a conscientious attitude throughout each lesson.	
6	# has worked well this year, demonstrating a sound understanding of the work covered.	
7	# has worked well this year, consistently producing work of a good standard.	
8	# participates well in most activities, showing a good level of concentration.	
9	# has worked reasonably well this year, producing work of a sound level.	
10	# has shown a good level of commitment towards his/her studies and has been successful in attaining his/her target levels.	
11	# has worked well this year but can be a bit reserved during discussion work, being willing to answer a question but rarely offering up ideas of his/her own.	
12	# has worked well, but has struggled at times with the depth of content and when recalling specific technical terms.	
13	# has worked inconsistently this year, producing work of a variable standard.	
14	# has found it difficult to connect with the subject content this year, finding it difficult to maintain his/her concentration throughout each lesson.	
15	# has struggled this year with the depth of knowledge required, but has remained positive in his/her approach to his/her work.	
16	# has found it difficult to work throughout each lesson, being easily distracted and can be a distraction to others.	
17	# has made less than the expected level of progress this year, mainly due to a lack of focus during the lessons and a reluctance to engage fully with the lesson content.	
18	# has found the lesson content difficult to process and has struggled to remain on task, consequently he/she has made less than the expected level of progress.	
The following pronouns will be switched automatically according to the student's gender setting: He/She, He's/She's, he/she, his/her, his/hers, him/her, himself/herself, His/Hers, His/Her		

There is also a help page containing instructions on how to use the software.

This spreadsheet will only work if you enable macros in the security settings of Microsoft Excel - macros are small pieces of visual basic computer code that allow complex tasks to be carried out automatically by pressing a button. Macros can also be used by malicious programmers to carry out tasks similar to computer viruses and so most people have the macro security setting in Excel turned off to prevent malicious codes running on their machines. This spreadsheet will require you to enable macros when you open it to carry out the automatic functions. Open the Excel options menu then "trust center" settings and select the option for "macro settings" then select the option "disable all macros with notification". Excel should then prompt you with "Security Warning - macros have been disabled" and an "options" box select it then "enable this content" whenever you open this spreadsheet, this will not affect the macro settings for other spreadsheets. Remember when saving this spreadsheet it must be saved in a macro enabled .xlsm format.

#### Entering new comments into the Comment Bank sheet

You can create your own personalised comments into the comment bank.

Each of the following phrase categories have space for 18 graded comments: Opening, Behavioural, Progress, Strengths, Weaknesses, Closing, then 3 user defined categories. When entering new comments into the comment bank use the # where you want the name of the student to be inserted. The report writer will automatically change gender specific pronouns (he/she his/her etc) for each student using the gender assigned in the student details sheet, enter He/She or his/her in the comments as shown in the box below the phrases in the Comment Banks sheet. Phrases can be organised within each comment category into 6 top, 6 medium & 6 lower achieving comments. It is possible to enter long statements but even though the last section of the statement may not always be visible in the comments selection window all of the statement will be pasted into the student comment window.

#### Entering Student Details

Upload your student names by copying and pasting the names of the students from your register spreadsheet into the Student Details Sheet. Add M/F for the gender of each student. Up to 50 student names may be copied in at once.

#### Creating a student report comment

Open the Main sheet.

The name of the student and their gender should appear in the appropriate box.

Select the desired comment from the list by clicking on the option circle to the right of the comments list. Select the next comment category and select the comment wanted. You may also free type in the report comment box. When you have finished composing your comment click on the button below the comment box to copy the report comment which you can then paste directly into your reporting software package or word. Then select the next pupil and repeat. All comments will appear in the Student Details sheet. When all comments are completed click on the "Click to save" button and you will be prompted for a location to save an Excel spreadsheet of the entire group comments in the format "report comments for ???,xls".

Copyrighted © john\_leather@hotmail.com

### 1.4.2.1 Pros and cons

This existing solution has a few problems. One problem is the user interface is unclear and hard to use. The comments bank page contains all the comments in a spreadsheet and in order to view them you must scroll horizontally for ages, it's clunky and slow. A second problem is it uses a spreadsheet to store the name and gender of the student which is a very limited approach and gives you little control. Similar to the last existing solution, since this is not connected to a database of students, it cannot compare the students to other students. Looking at this flaw, my solution will aim to feature a system of comparing students to other students or the class average. It will calculate the class average and compare the student to that average in a comment that is generated.

On the other hand there are some strong features in this solution that will be adopted. One good feature is the help page. This gives instructions on how to use the software. Even though a system like this should be intuitive and simple to use it is still a good feature. People who are confused can look here and see how to use the software correctly. It can contain instructions but also tips on how to be efficient and get the job done as quickly as possible. I will improve upon this feature by improving the UI. There will be a button on the corner of the page in the form of a question mark, clicking it will bring the help page, this way is easier to access and more clear. A second positive is the huge variety of comments in the comment bank. A teacher doesn't want to sound repetitive or generalise the reports for each of their students, the comment bank gives plenty of options to express how the teacher feels about the student and their performance at school.

# 1.5 Features of the proposed computational solution

The solution will contain many features which will be outlined here. These features will be formed from the existing solutions in section 1.4 and based on the requirements in section 1.7.

1. The staff will be greeted with a login page when they first launch the program, this will allow them to enter their username and password. If they have not created an account there will be a sign up button which will take them to a sign up page. This satisfies requirements 14 and 16.
2. When creating an account the staff will be required to enter their first name, last name, email and password. A series of validation will be carried out. Their password must be at least 8 characters long and contain a special character, their username must be between 6 and 30 characters and their email must be in the correct email format. If they give empty or invalid data the program will give an error message saying their mistake. If their username already exists within the database the program will also give an error message. This satisfies requirements 17, 18, 19, 20, 21, 22 and 23
3. Once they have logged in they will be greeted with the home page. This will be the central part of the program, it will be a larger window with a dark blue background. It will contain a large clear title and clear large buttons that say what their purpose is. This is based on the design in solution 1. There will be several buttons taking the user to different pages. There will also be a sign out and quit button allowing the user to close the program or sign out to change user whenever they want. This satisfies requirements 28, 7, 9 and 12.
4. The staff will be able to go to a page where they can update their details, this page will allow them to update all their information if they need to. It will allow them to update their first and last name, username and password, email, address and national insurance number. They will be able to cancel the changes with a cancel button or save the changes to the database. If their username already exists the program will give an error message. This satisfies requirements 28 and 34.
5. The staff can navigate to the students page from the home page, this will contain a table listing all their students and key information on each student. This table will

use the school database to display the staff's students. They will be able to scroll through the table and search for the students full name, or student ID. They will also be able to sort each column from descending or ascending. This satisfies requirements 24, 36 and 37.

6. The staff can also navigate to the grades page and student performance page from the home page. These pages will once again contain a table listing the students and other information on their grades and performance. These tables will use the school database to display the staff's students. They will again be able to scroll through the table and there will be search functionality for full name and student ID. The grades page will contain filters so the staff can filter for a specific category. This satisfies requirements 37 and 41.

7. There will be a reports page that the staff can navigate to from the home page once again. This page will contain all their previous reports they have written on a table, there will also be a large text box allowing them to view and edit the reports if they wish to. There will be search functionality for the students full name, student ID and date modified. There will be sort functionality for each column. This satisfies requirements 37 and 43.

8. When a staff wants to create a report for their student they can click the generate report button which will take them to a page where they must select a student. This page will contain a list of students they can select and also important information on each student. Once they select a student they can click the continue button taking them to the generate report page. This satisfies requirements 48.

9. Once on the generate report page, the staff can select from a series of radio button groups ranking their students from 1 to 5. They can also toggle a series of tick boxes to determine what the final report will contain. This is taken from existing solution 1. Once they are happy with the inputs they have given they can click the generate button which will take them to a new page containing the generated report. This satisfies requirements 49.

10. The final paragraph page will contain a large text box taking up most of the page with the final paragraph there. The staff will be able to edit this page if they want to make any changes to the report. This is taken from existing solution 1. They can also press a regenerate button to randomly generate the report again based on their inputs from the last page. Once they are happy with the report they can press the save button, this will save the report to the database and return them to the page

where they can select a student, allow them to continue and write a report for the next student. This satisfies requirements 51, 53, 54 and 55.

## 1.6 Limitations of the proposed computational solution

The stakeholder wanted a feature that shows photographs of the students when clicking on their name. This feature may be possible but is not feasible due to the time constraints so it unfortunately cannot be implemented. It would require a way for Delphi to link to existing school systems like SIMS to access photographs, or manually accessing the photos of all students and adding them to the database, which is too time consuming.

Another feature highly requested by the stakeholders was printing functionality built into the system; this would be able to format the reports into a selected page size and print them to nearby printers. This feature would not be possible due to the limitations of Delphi 7, it is not possible with the current version to connect to a printer or format text to a page size. Therefore this feature cannot be implemented.

## 1.7 Requirements for the proposed solution

After gathering information with the stakeholder and looking at existing solutions, a list of requirements for this solution can now be made. These requirements will be an important section to look back on and allow the project to stay on track. This will include hardware and software requirements for the solution to work and all the design and functionality requirements.

### 1.7.1 Hardware requirements

Number	Feature	Explanation and justification
1.	Access to basic I/O peripherals. This includes a basic full sized keyboard, optical or laser mouse, monitor with at least 720p display and common	The software will require basic input and output devices including a keyboard, mouse and monitor. This is to allow the user to interact with the system as intended. The

	aspect ratio.	stakeholders already have access to these peripherals which means they won't have to spend money to get this new solution into place.
2.	Access to basic windows computers with minimum specifications. This would be around 4GB of RAM, Dual-core 2Ghz CPU, 2GB free storage space available.	The software will require a basic windows computer. It does not have to be a high performance machine as the solution will be able to run on any machine. Schools and teachers have access to basic computers so they again won't have to spend money on expensive hardware to run the software.

## 1.7.2 Software requirements

Number	Feature	Explanation and justification
3.	Delphi 7 software installed.	The solution will be developed using delphi, a windows based object pascal development environment. The application will be a delphi application so delphi must be installed on the computer. The stakeholder outlined budget constraints for the school, therefore the solution must not require any expensive software. The school already owns delphi 7 so there will be no additional costs to run this solution.
4.	Microsoft Access 2000 or later installed.	The solution will use a school database, this database is crucial for the solution to work and will contain important information on the staff and the students, without it the solution cannot function. The database will be in Microsoft Access 2000 version so this must be installed on the systems. The stakeholders already own

		Microsoft products including Access so this comes as no additional cost to the school.
5.	Functioning windows operating system. This includes windows 7, 8, 8.1, 10, or 11.	Delphi 7 will be able to run on a windows operating system. The school's existing systems run on windows. The staff will be familiar with this software and there will be no additional cost having to switch operating systems.

### 1.7.3 Design & functionality

Number	Feature	Explanation and justification
6.	There must be a login and sign up form with a small vertical window size.	The user should be greeted with a login page and have the option to sign up which will take them to a sign up page. These forms should be smaller than the main forms as they don't need to be large since they aren't displaying lots of information. It is also indicating to the user they have not entered the main program yet.
7.	The main program forms must follow a 16:9 aspect ratio and be smaller than the monitor size.	The main forms must follow the standard monitor aspect ratio, they should also be smaller than the monitor to allow for a wider range of monitors. By doing this monitors with a 720p display or lower will still be able to run the program and view each form fully.
8.	The login form must contain the school logo and name of the application.	This is a visual change that would improve the look of the program, by showing the school logo and name of the solution the user knows what they are about to use and it makes the program look more complete and trustworthy. This was requested by the

		stakeholders from the questionnaire during the planning section.
9.	The login form must contain a username and password edit box so the user can login.	In order for the user to login they need to input their username and password so these edit boxes are mandatory.
10.	The login form must contain a 'forgot password' button so the user can reset their password.	If a user forgets their password they should be able to reset it, otherwise they will have to create a new account which will cause duplication in the database and is more time consuming.
11.	Each form should have a dark blue background with contrasting white text.	The software should be easy to look at and have a clear layout, by having a dark blue background it creates a good look without being distracting. During the interview with Mr. Nicolston this was outlined as a key requirement.
12.	Every title, button and edit box should be clearly labelled.	Once again the software should be easy to use, the user should understand what everything does right away, this requirement will help with this.
13.	Verdana font should be used throughout each form. Verdana size 16 bold should be used for each button and size 16 regular for each edit box. Each title should be size 48 bold.	The verdana font is a clean professional look, it will improve the UI significantly if this font is used throughout the program with consistent sizing and bold or regular formats.
14.	Program must have login functionality, staff should be able to login with their own account.	The school wants to store the data of their staff and provide privacy and security to the staff. An account is the best method of doing this, therefore it is best that they have a way to login with their user details.
15.	Program must display an error message to the user if their	This validation is used to make sure the staff know their login

	username or password is incorrect.	details and ensure nobody else can access their account.
16.	Program must allow users to sign up with a sign up page	If a staff member does not have an account but wishes to use the solution they need to be able to sign up.
17.	Program should require the user to enter a username, password, full name and email address before signing up.	The user needs to enter their details so they can be saved in the database and displayed. If they enter their username and password they won't have a default username or password which could be unsafe.
18.	Program must display an error message if a text box is left empty or the value entered is invalid.	There will be a large amount of validation in the sign up page, if a user has inputted invalid data, they must know which field was invalid instead of having to guess.
19.	The password entered must be at least 8 characters and require a special character.	To ensure security the program will validate the password making sure it's long enough and contains a special character. This means it's harder to hack the user's account.
20.	The username must be between 6 and 30 characters and contain no illegal characters.	The username must also follow a standard format, there will be a list of illegal characters it cannot have and it must have a realistic feasible length.
21.	The email address should be the correct format.	The email address must be the correct format otherwise a user can put anything they want and this will cause errors down the line. If an email system is implemented the email must be valid.
22.	Program must tell the user if the username already exists.	The login page will require a username, this means there cannot be duplicate usernames. Once validation is complete the program must display an error

		message saying the username already exists.
23.	Program must save the sign up details to the database once sign up is successful and return the user to the login page.	If the sign up is successful, the details they entered should be saved into the database and they should be greeted with the login page again so they can quickly log in right away.
24.	All databases must be in 3rd normal form.	This is a basic requirement for any database because it prevents any data duplication and errors when accessing using SQL.
25.	Program should have confirmation messages throughout.	This prevents the user from accidentally changing a setting they didn't want to. For example the sign up page should confirm if they want to continue or not.
26.	Each page should contain a large title at the top with the name of the page.	The user should always know what page they are on, the best way of doing this is having a large title at the top of each page, this makes it clear to the user.
27.	Each page should contain a 'Go Back' button at the top right to return to the previous page.	To navigate the program quickly there should be a back button on each page, this will go to the previous page, if a user accidentally misclicks they should easily be able to go back. By having this at the same position on each page the user will become familiar with it and can use it quickly.
28.	Home page should contain a sign out or quit button.	The home page will be the central part of the program which will be accessed often. A quit button allows the user to quit the application once they have done using it, a sign out button allows the user to sign out and takes them back to the login page. If there is a swap with staff members

		this is useful.
29.	Home page should contain 6 buttons going to 6 different forms. These are students, grades, student stats, reports, edit details and generate reports.	The home page should be the central point of the application, all the main pages should diverge from the home page, this is the quickest way of navigating the program, having all the main pages at the home page means the user can quickly use the program and get to anything they want.
30.	Home page should contain a profile picture of the staff logged in at the corner of the page.	To improve the UI and user experience a profile picture should be displayed at the corner of the home page, this adds a personal element and users will enjoy it. This was requested by the stakeholders through interviewing.
31.	Edit details page should contain edit boxes for first name, last name, email, address, phone number, national insurance number, username and password.	The edit details page should allow the user to change or add all possible details they can, some of these details may not be required or given during the sign up process so by having these edit boxes the user can now set all their details. This was another key requirement that came about during the interviewing.
32.	Edit details page should have a save button and a cancel button on the right size of the page.	If the user starts entering new details and changes their mind or makes a mistake they should be able to cancel to reset their changes. A save button is required so they can save their details.
33.	Program should allow the user to update their details in the edit details page.	If there is a change of details like address or surname they need to update that information on their account and the database.
34.	Program must tell the user if their new username already	Similar to before, logging in requires a username for one

	exists when updating their username.	specific user, so there cannot be duplicate usernames.
35.	Students page must contain a large table in the centre of the page with 3 search boxes below which are: first name, last name and student ID.	In order for the staff to navigate the students table there should be search boxes easily readable and accessible. Having these search boxes at the bottom of the page makes it easy to search for a student.
36.	Program must allow staff to view their students in a database grid from the students table.	Staff will want to view their students, by adding this form it will contain a large table showing their students and important information.
37.	Program must allow staff to search for the students full name or student ID.	The staff may have many students so searching for their name or ID is very important to save time. This feature was a key requirement outlined during the second interview with Mr. Nicolston because he said it is important to search through the large number of students.
38.	Student performance page must contain a large table in the centre of the page with 3 search boxes below which are again: first name, last name and student ID.	In order for the staff to navigate the student performance table there should be search boxes easily readable and accessible. Having these search boxes at the bottom of the page makes it easy to search for a student.
39.	Grades page must contain 6 drop down boxes for filtering grades which will be: working at, target, predicted, end of year, subject, year.	If the staff want to search for a specific grade, like all students with an A* last year, or all students with a B in maths they can do so with these drop down boxes. By having these at the top of the page it is clear to the staff how to use them and easy to filter the data whenever. During the questionnaire this was a key requirement that came up, staff

		members wanted to be able to filter out and view grades quickly which they could not do with the current system.
40.	Grades page must contain a table in the middle after the filter boxes and then 3 search boxes below which are: first name, last name and student ID.	Like the other pages there should be large search boxes at the bottom of the page and the table in the middle of the page, having a large table in the middle allows the staff to view all the details and the search boxes at the bottom allow them to search for students quickly.
41.	Program must allow staff to view students' information in a database grid from various tables in the database. This would include grades, attendance, achievementPoints.	The staff need to know important information about their students to help when creating their report, these forms allow the staff to quickly check how their student is doing and make a decision.
42.	Reports page must contain a table on the left of the page and an edit box on the right of the page displaying the reports. Below must include 3 search boxes: date modified box, full name and student ID.	There should be a large table on the left side of the form displaying the students and on the right side there should be a large text box displaying the reports. This allows the staff to easily view the table and read through the reports.
43.	Program must allow staff to view their previous reports written for their students.	This allows the staff to view their previous reports and also make edits to their reports if they are not happy with it.
44.	Program must hide certain data from the database from the staff.	The reports the staff write are private to them, so that information cannot be shown to other staff members. Also, a staff member won't care about a student they don't teach and are not writing a report for.
45.	Generate report page must include a list box in the centre of the page showing a list of	The staff must be able to select a student before continuing to the next section, having a list box in

	students.	the centre of the page is the best way of displaying this information, it will contain a list of students and the staff can easily scroll through and select the one they want.
46.	Generate report page must include 6 boxes below showing student information, these will be: student ID, year group, form group, candidate number, parent email and student email	When the staff selects a student they might want to know important information on the student before creating a report, similar to the report page these fields will be displayed to easily show the staff information at a glance. They will be updated whenever they select another student.
47.	Generate report page must have a continue button at the bottom right to continue to the next page	Once the staff has selected a student they need to go to another page to create the report, a large continue button at the bottom right is a clear way to indicate to the staff how to move to the next section.
48.	Program must require the staff to select a student they want to write a report for.	Before writing the report, the staff need to tell the program which student they want to make the report for, this stage needs to be quick and simple for the staff to do so they can do multiple students.
49.	Report generator page must contain a series of radio buttons defining how the final report will be structured. The radio button groups should be from 1-5 and the sections should be: work completion, homework completion, work quality, homework quality, behaviour, and lesson involvement.	The staff must decide how good or bad the report is for the student. By displaying several radio groups ranking from 1 to 5, the staff can choose what the final report will be like. This is crucial as it makes the reports personal and not just generic or repeated for every student.
50.	Report generator page must contain a series of checkboxes to control which section to	The staff should be able to control the outcome of the report, these check boxes allow the staff to pick

	<p>include in the report, these will be: Include introduction, include conclusion, mention grades, mention attendance and punctuality, mention behaviour and achievement points.</p>	<p>what will be included. If the staff don't want a conclusion they can simply untick it. This was requested by Mr. Nicolston during the interview we had, as a key requirement.</p>
51.	<p>Final paragraph page should display a large text box with the final report.</p>	<p>In order for the staff to see the report clearly and make changes easily, a large text box containing the final paragraph should be present.</p>
52.	<p>Final paragraph page should contain 3 buttons below the text box: save, go back and regenerate.</p>	<p>There should be 3 clear buttons below the report, these names will make it clear what each button does and having them at the bottom of the page makes it easily accessible.</p>
53.	<p>Program must contain a button to regenerate the report, creating a new random report but still based on the inputs the staff gave.</p>	<p>If a staff member is unhappy with the report they should be able to press the regenerate button, this will repeat the algorithm used to create the report and change it again.</p>
54.	<p>Program should allow the staff to make changes to the final report before saving it.</p>	<p>The staff will know more about the students than the program will, they are likely to want to add more to the report at the end or make changes to it. The text box should be editable so they can make changes before saving. This was a key requirement outlined during the planning section.</p>
55.	<p>Program should have an option to save the report to the database.</p>	<p>Once the user is happy with the report they should be able to save it to the database so it is stored there and accessible any time by going to the database table that was mentioned earlier as a requirement.</p>

56.	Program should have an option to email the parents of the student with the report attached.	Once the staff has created the report they should be able to email the parents of the students with the parent email field from the database, this will mean the report can get to the parent quickly.
-----	---	--

## 1.8 Success criteria

This section will outline the features that the solution must have to make it successful from the perspective of an end-user.

Number	Criteria	Explanation and justification
1.	Staff must be able to successfully login to their account.	The school wants to store the data of their staff and provide privacy and security to the staff. An account is the best method of doing this, therefore it is best that they have a way to login with their user details.
2.	Staff must be informed if they entered the wrong username or password so they can correct it.	This validation is used to make sure the staff know their login details and ensure nobody else can access their account.
3.	Staff must be able to successfully create an account using their username, password, full name and email.	The staff needs to enter their details so they can be saved in the database and displayed. If they enter their username and password they won't have a default username or password which could be unsafe.
4.	If the staff enter invalid or empty data they must be informed.	There will be a large amount of validation in the sign up page, if a staff member has inputted invalid data, they must know which field was invalid instead of having to guess.

5.	Staff must be informed if the username they entered already exists.	The login page will require a username, this means there cannot be duplicate usernames. Once validation is complete the program must display an error message saying the username already exists.
6.	Staff should be able to sign up and then automatically return to the login page successfully.	If the sign up is successful, the details they entered should be saved into the database and they should be greeted with the login page again so they can quickly log in right away.
7.	Staff should successfully be displayed a confirmation message.	This prevents the staff from accidentally changing a setting they didn't want to. For example the sign up page should confirm if they want to continue or not.
8.	Staff must be able to successfully update their details.	If there is a change of details like address or surname they need to update that information on their account and the database.
9.	Staff must be informed if their username already exists.	Similar to before, logging in requires a username for one specific user, so there cannot be duplicate usernames.
10.	Staff must successfully be able to view their students in a database grid from the students table.	Staff will want to view their students, by adding this form it will contain a large table showing their students and important information.
11.	Staff must be able to search the students full name or student ID successfully.	The staff may have many students so searching for their name or ID is very important to save time.
12.	Staff should successfully be able to view various tables in the database. This would include grades, attendance, achievementPoints.	The staff need to know important information about their students to help when creating their report, these forms allow the staff to quickly check how their student is doing and make a decision.

13.	Staff must be able to view their previous reports for their students successfully.	This allows the staff to view their previous reports and also make edits to their reports if they are not happy with it.
14.	Staff must be able to successfully select a student to write a report for from a list of students.	Before writing the report, the staff need to tell the program which student they want to make the report for, this stage needs to be quick and simple for the staff to do so they can do multiple students.
15.	Staff must be able to successfully select different radio buttons for different sections and select various checkboxes to determine how the report will be formed.	The staff must decide how good or bad the report is for the student. By displaying several radio groups ranking from 1 to 5, the staff can choose what the final report will be like. This is crucial as it makes the reports personal and not just generic or repeated for every student.
16.	Staff must be displayed a large text box containing the final report and be able to edit it successfully.	In order for the staff to see the report clearly and make changes easily, a large text box containing the final paragraph should be present. The staff will know more about the students than the program will, they are likely to want to add more to the report at the end or make changes to it. The text box should be editable so they can make changes before saving.
17.	Staff must be able to successfully regenerate the report, creating a new random paragraph from the inputs they gave.	If a staff member is unhappy with the report they should be able to press the regenerate button, this will repeat the algorithm used to create the report and change it again.
18.	Staff must be able to successfully save the report to the database once they are	Once the staff member is happy with the report they should be able to save it to the database so it is

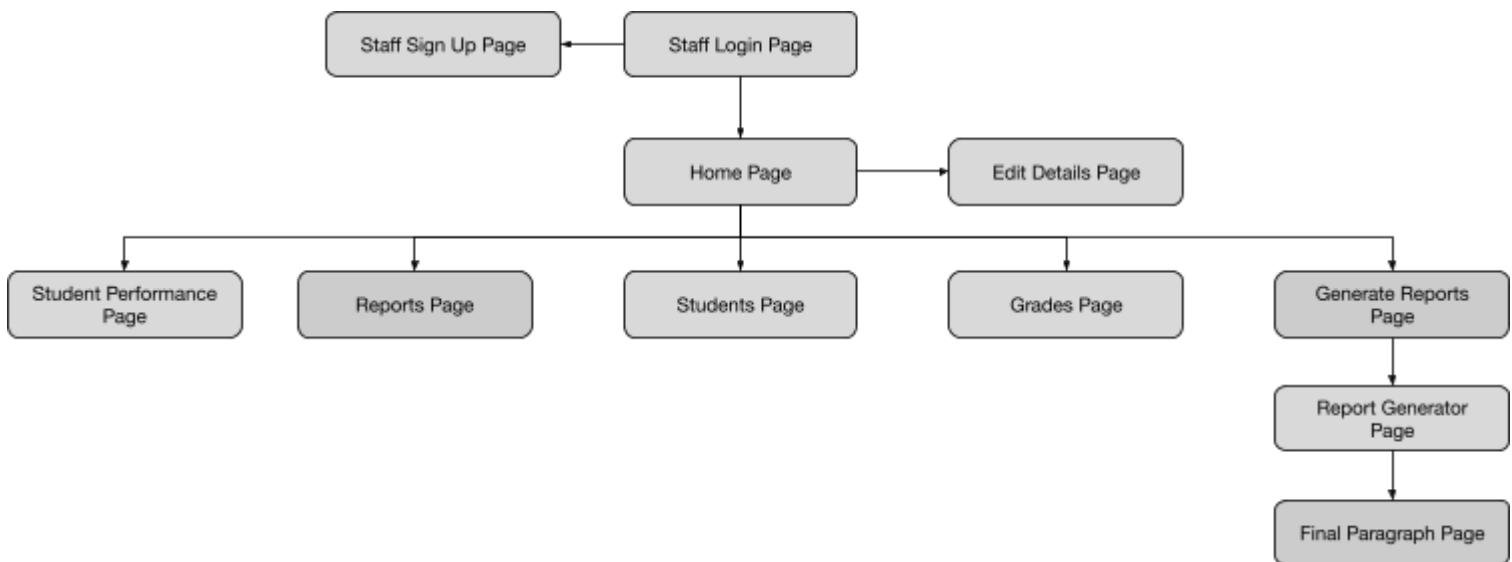
	happy with it.	stored there and accessible any time by going to the database table that was mentioned earlier as a requirement.
19.	Staff must be able to successfully email the parent of the student they wrote a report for.	Once the staff has created the report they should be able to email the parents of the students with the parent email field from the database, this will mean the report can get to the parent quickly.

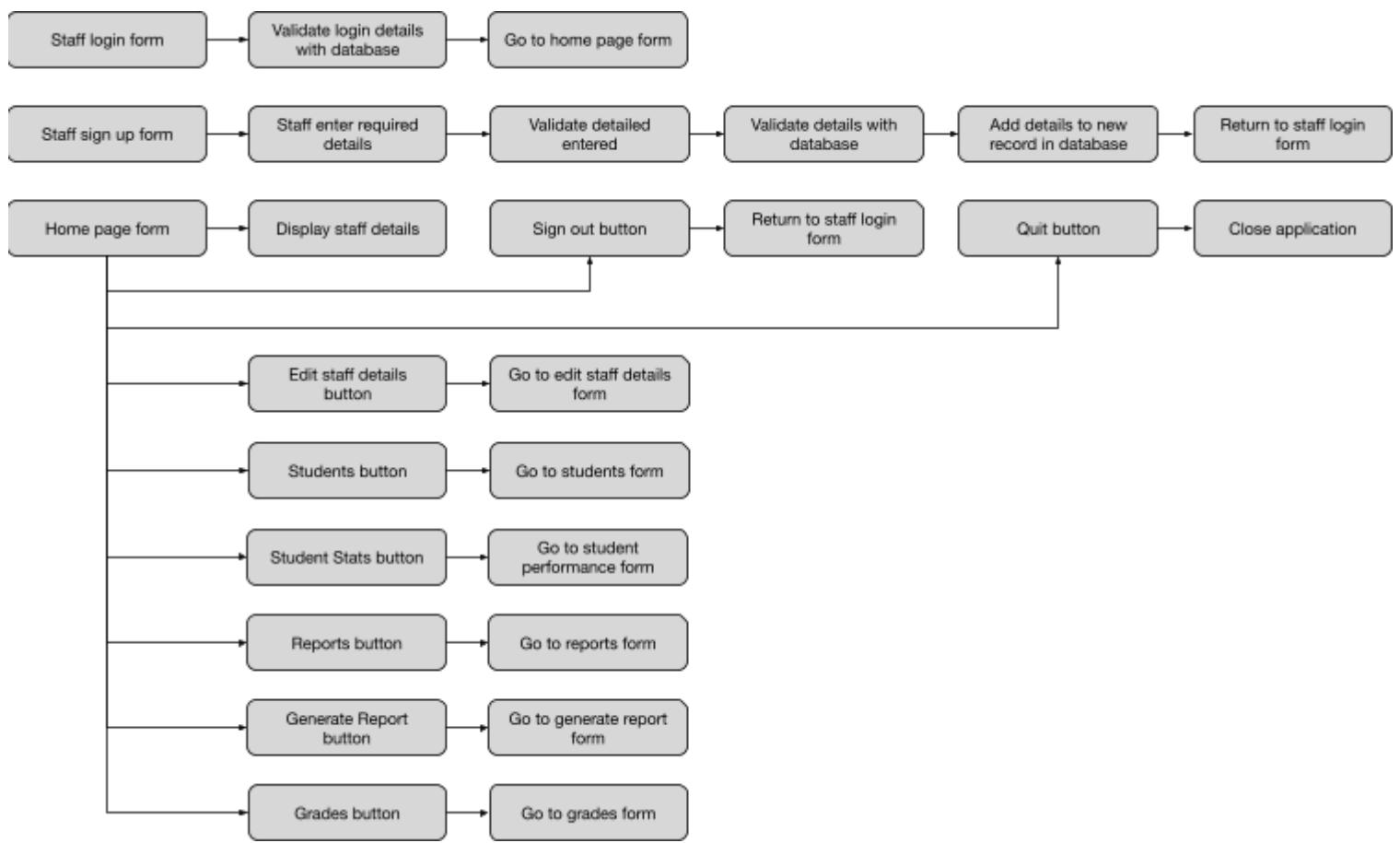
## 2. Design

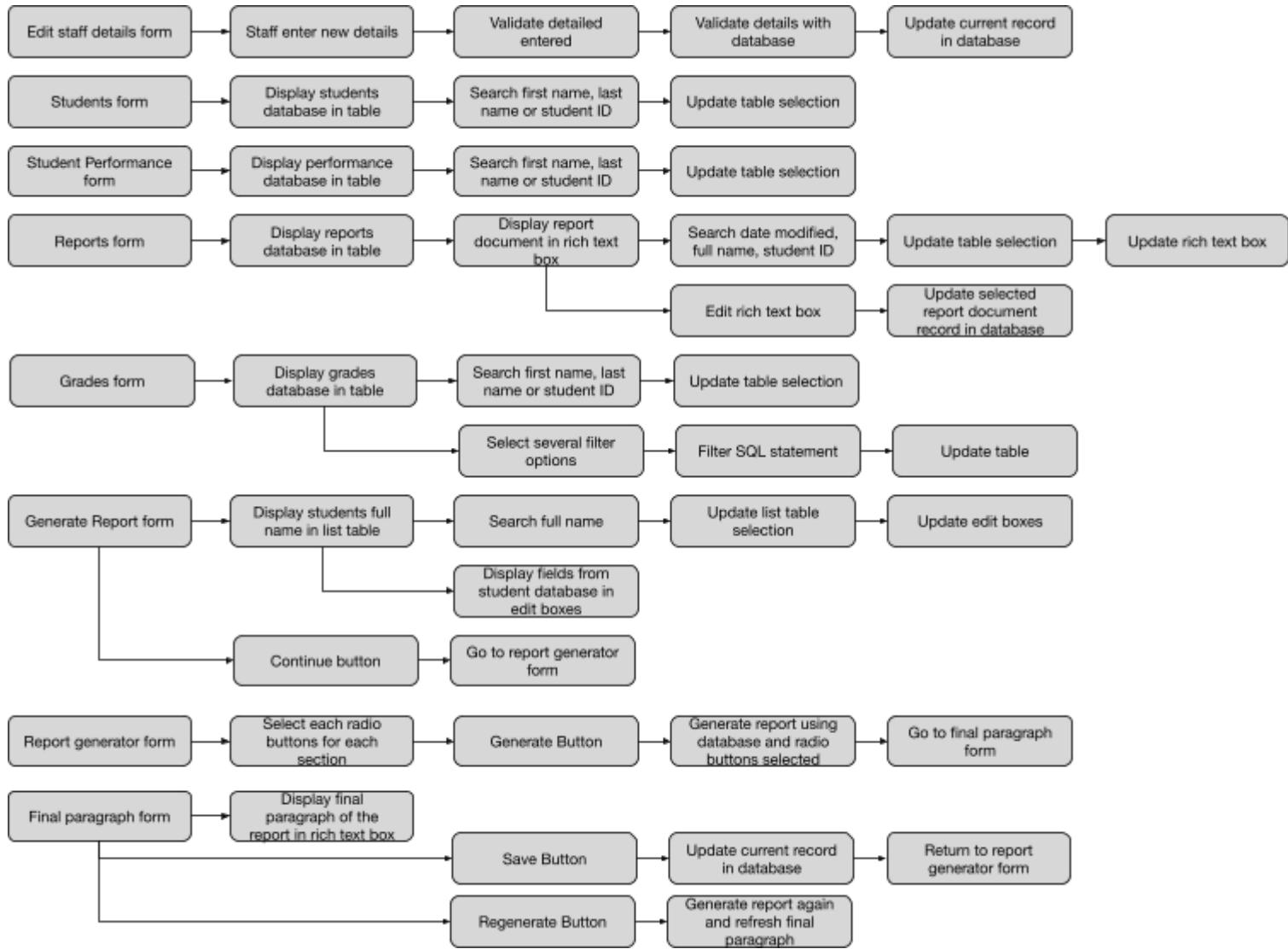
### 2.1 Decomposing the problem

#### 2.1.1 Decomposition Diagrams

The solution will contain several forms, each responsible for a different part of the software. These forms must be decomposed into a modular structure diagram beforehand. This will help gain an understanding of the layout of the solution and how each section will interact with each other.

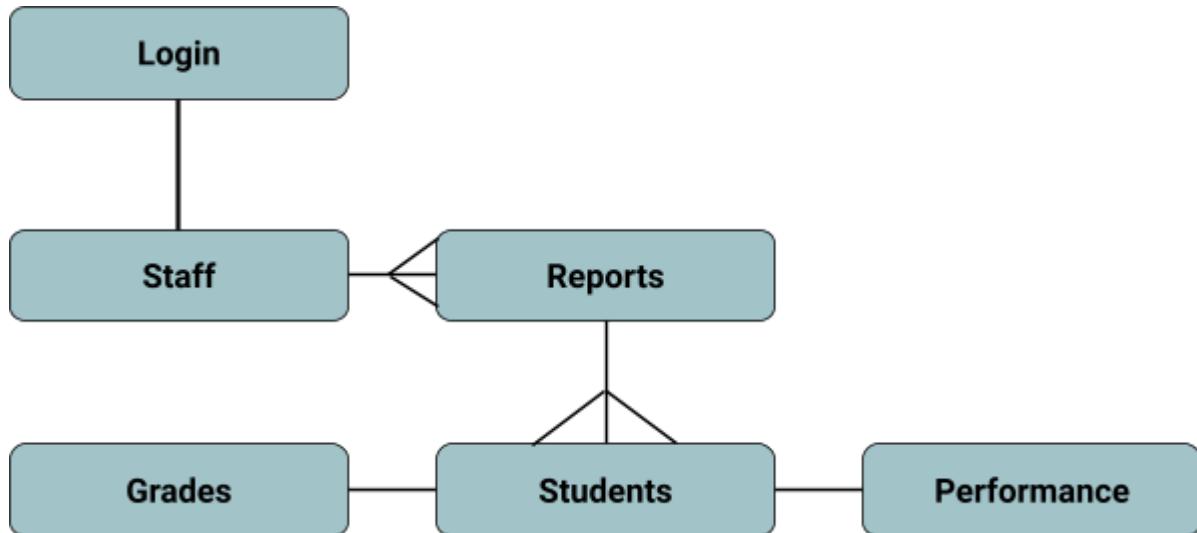






## 2.2 Structure of the solution

### 2.2.1 Entity relationship diagram



## 2.2.2 Normalisation

The database must be fully normalised, this will ensure the data is organised and there is no repeated data.

**primary key**

**foreign key**

**Unnormalised:**

staffID, staffFirstName, staffLastName, position, phoneNumber, email, address, nationalInsuranceNumber, username, password, reportID, reportDocument, dateModified, studentID, studentFirstName, studentLastName, yearGroup, formGroup, gender, studentEmail, parentEmail, candidateNumber, attendance, punctuality, behaviour Points, achievementPoints, academicYear, subject, workingAt, target, predicted, endOfYear

**1st normal form:**

1st normal form ensures each field is atomic and a primary key is defined.

**Staff (staffID, firstName, lastName, username, password, position, phoneNumber, email, address, nationalInsuranceNumber)**

Reports (reportID, reportDocument, dateModified, **studentID**, **staffID**)

Students (studentID, firstName, lastName, yearGroup, formGroup, gender, studentEmail, parentEmail, candidateNumber, attendance, punctuality, behaviourPoints, achievement points, academicYear, subject, workingAt, target, predicted, endOfYear)

### **2nd normal form:**

2nd normal form ensures all non-key fields depend on all components of the primary key. If the primary key is a single field this is guaranteed, with this database this is the case.

Staff (staffID, firstName, lastName, username, password, position, phoneNumber, email, address, nationalInsuranceNumber)

Reports (reportID, reportDocument, dateModified, **studentID**, **staffID**)

Students (studentID, firstName, lastName, yearGroup, formGroup, gender, studentEmail, parentEmail, candidateNumber, attendance, punctuality, behaviourPoints, achievement points, academicYear, subject, workingAt, target, predicted, endOfYear)

### **3rd normal form:**

3rd normal form ensures all non-key fields depend only on the primary key

Staff (staffID, firstName, lastName, position, phoneNumber, email, address, nationalInsuranceNumber)

Login (username, password, **staffID**)

Reports (reportID, reportDocument, dateModified, **studentID**, **staffID**)

Students (studentID, firstName, lastName, yearGroup, formGroup, gender, studentEmail, parentEmail, candidateNumber)

Performance (attendance, punctuality, behaviourPoints, achievementPoints, **studentID**)

Grades (academicYear, subject, workingAt, target, predicted, endOfYear, studentID)

## 2.2.3 Definition of data requirements (design data dictionary) and validation required

This section will contain a data dictionary for the database. Having an initial layout and structure, knowing which data will go where will make it easier to implement the database

**Table name** - Login

**Purpose** - Stores the username and password for the staff

Field Name	Required?	Date Type	Sample data	Validation rule	Validation Type	Key field
username	Yes	Short Text	'johnsmith'	Must be unique, must be a string that only contains allowed characters, must be between 6 and 30 characters	Type check, presence check, length check	N/A
password	Yes	Short Text	'qwertyst123'	Must be unique, must be a string, must contain at least one special character from a set of required characters for security, must be more than 8 characters	Type check, presence check, length check	N/A
staffID	Yes	Number	45	Must be unique, must be an integer	Type check, presence check	Foreign Key

**Table name** - Staff

**Purpose** - Stores key information on the staff

Field Name	Required?	Date Type	Sample data	Validation rule	Validation Type	Key field
staffID	Yes	AutoNumber	13	Must be unique, must be an integer	Type check, presence check	Primary Key

firstName	Yes	Short Text	'David'	Must be a string, must only contain letters	Type check, presence check	N/A
lastName	Yes	Short Text	'Frost'	Must be a string, must only contain letters	Type check, presence check	N/A
position	Yes	Short Text	'Chemistry Teacher'	Must be a string	Type check, presence check	N/A
phoneNum ber	Yes	Short Text	079831447 56	Must be a number with 11 digits	Type check, presence check, length check	N/A
email	Yes	Short Text	'davidfrost 123@gmail. com'	Must be a string, must be correct email format	Type check, presence check, format check	N/A
address	Yes	Short Text	SW14 9SN	Must be a string, must be correct address format	Type check, presence check, format check	N/A
nationalIns uranceNum ber	Yes	Short Text	QL 195831 K	Must be a string, must be correct national insurance number format	Type check, presence check, format check	N/A

**Table name - Students****Purpose -** Stores key information on the students

Field Name	Required ?	Date Type	Sample data	Validation rule	Validation Type	Key field
studentID	Yes	AutoNumber	230	Must be unique, must be an integer	Type check, presence check	Primary Key
firstName	Yes	Short Text	'Jamarion'	Must be a string	Type check, presence check	N/A

lastName	Yes	Short Text	'Green'	Must be a string	Type check, presence check	N/A
yearGroup	Yes	Short Text	07	Must be an integer between 7-13 for each school year	Type check, presence check, length check	N/A
formGroup	Yes	Short Text	LYG	Must be a string	Type check, presence check	N/A
gender	Yes	Short Text	'M'	Must be a string	Type check, presence check	N/A
parentEmail	Yes	Short Text	'AdamGreen75@hotmail.co.uk'	Must be a string, must be correct email format	Type check, presence check, format check	N/A
studentEmail	Yes	Short Text	'6120@trentlock.org'	Must be a string, must be correct email format	Type check, presence check, format check	N/A
candidateNumber	Yes	Number	6120	Must be an integer, must be unique	Type check, presence check	N/A

### Table name - Performance

Purpose - Stores the performance of each student

Field Name	Required?	Date Type	Sample data	Validation rule	Validation Type	Key field
attendance	Yes	Number	100	Must be an integer, must be between 0 and 100	Type check, presence check, length check	N/A
punctuality	Yes	Number	89	Must be an integer, must be between 0 and 100	Type check, presence check, length check	N/A
behaviourPoints	Yes	Number	4	Must be an integer	Type check, presence	N/A

					check	
achievementPoints	Yes	Number	12	Must be an integer	Type check, presence check	N/A
studentID	Yes	Number	230	Must be unique, must be an integer	Type check, presence check	Foreign Key

**Table name - Grades**

**Purpose** - Stores the grades of each student in each subject across each year

Field Name	Required?	Date Type	Sample data	Validation rule	Validation Type	Key field
gradeID	Yes	AutoNumber	1	Must be unique, must be an integer	Type check, presence check	Primary Key
subject	Yes	Short Text	'History'	Must be a string	Type check, presence check	N/A
academicYear	Yes	Number	2023	Must be an integer, must be a year	Type check, presence check	N/A
workingAt	Yes	Short Text	'A*'	Must be a string for each grade obtainable	Type check, presence check	N/A
target	Yes	Short Text	'B'	Must be a string for each grade obtainable	Type check, presence check	N/A
predicated	Yes	Short Text	'F'	Must be a string for each grade obtainable	Type check, presence check	N/A
endOfYear	Yes	Short Text	'C'	Must be a string for each grade obtainable	Type check, presence check	N/A
studentID	Yes	Number	5	Must be unique, must be an integer	Type check, presence check	Foreign Key

**Table name** - Reports

**Purpose** - Stores the reports for the staff to view for each of their students

Field Name	Required?	Date Type	Sample data	Validation rule	Validation Type	Key field
reportID	Yes	AutoNumber	26	Must be unique, must be an integer	Type check, presence check	Primary Key
reportDocument	Yes	Long Text	N/A	Must be a string	Type check, presence check	N/A
dateModified	Yes	Date/Time	16/1/2023	Must be a date time	Type check, presence check	N/A
studentID	Yes	Number	2	Must be unique, must be an integer	Type check, presence check	Foreign Key
staffID	Yes	Number	7	Must be unique, must be an integer	Type check, presence check	Foreign Key

## 2.2.4 Interface design

An initial design of all the forms must be planned before any real implementation of the solution, this is to get a feel for and judge how the overall design of the forms will look like for the staff when using the software. This also allows for further improvements and changes to be made later on. Graphs and email formatting will be designed here too as they will be included in the solution and are part of the interface.

**Form name** - Login

Size 24 bold text, verdana font

Light blue background colour (55BFE0)

Username

Password

LOGIN

SIGN UP

Size 16 bold text, verdana font. Labels that say what the text box is for

Equally sizes and spaced buttons and input boxes. Buttons and boxes different sizes to indicate the difference

Size 16 bold text, Verdana font.

### Form name - Sign Up

Size 24 bold text, verdana font

Light blue background colour (55BFE0)

Size 16, verdana font, several labels indicating the purpose of the text box

First Name

Last Name

Email

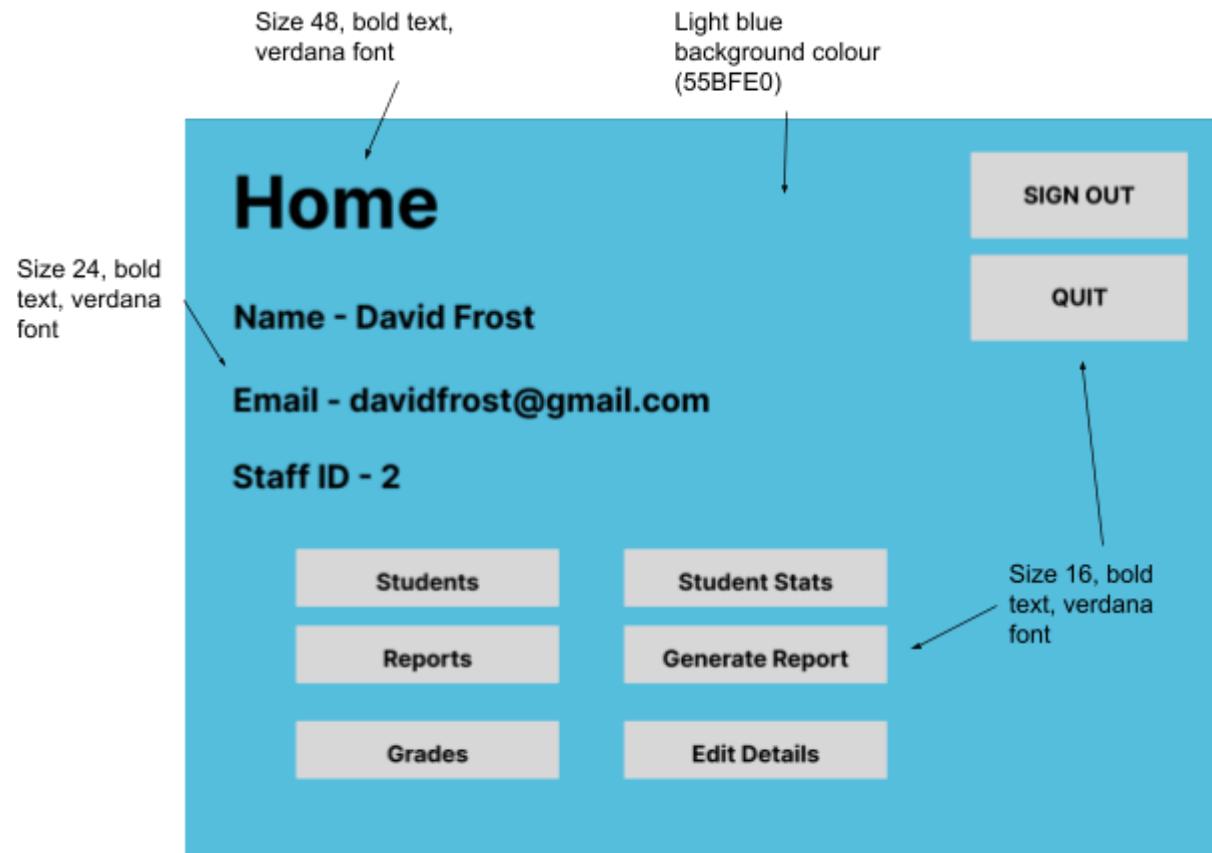
Username

Password

SIGN UP

BACK TO LOGIN

Size 16, bold text, verdana font

**Form name - Home Page****Form name - Students**

Size 48, bold text, verdana font

Light blue background colour (55BFE0)

Size 16, bold text, verdana font

Size 14, verdana font in table

Size 20, bold text, verdana font

Size 16, Verdana font, labeling the search box

The screenshot shows a web page with a light blue header and body. At the top left is the title 'Your Students' in a large, bold, black font. To the right is a 'GO BACK' button. Below the title is a table with columns: Student ID, First Name, Last Name, Year Group, Form Group, Student Email, Form Group, and Candidate Number. The table has 8 rows. At the bottom left is a large, bold, black 'SEARCH:' label. To its right are three input fields labeled 'First name', 'Last name', and 'Student ID'. A label 'Size 16, Verdana font, labeling the search box' points to the 'SEARCH:' label.

**Form name - Student Performance**

Size 48, bold text, verdana font

Light blue background colour (55BFE0)

Size 16, bold text, verdana font

**Your Students**  
(Performance overview)

GO BACK

Student ID	First Name	Last Name	Attendance	Punctuality	Behaviour Points	Achievement Points

Size 14, verdana font in table

Size 20, bold text, verdana font

First name      Last name      Student ID

**SEARCH:**

Size 16, Verdana font, labeling the search box

**Form name - Grades**

Size 48, bold text, verdana font

Light blue background colour (55BFE0)

Size 16, bold text, verdana font

**All Students**  
(Grade Overview)

GO BACK

Size 16, verdana font

**FILTER BY:**

Working in grade ^	Target grade ^	Year ^
End Of Year Grade ^	Predicted Grade ^	Subject ^

Size 20, bold text, verdana font

Working At Grade ^	Last Name	Year	Subject	Working At	Target	Predicted	End of year

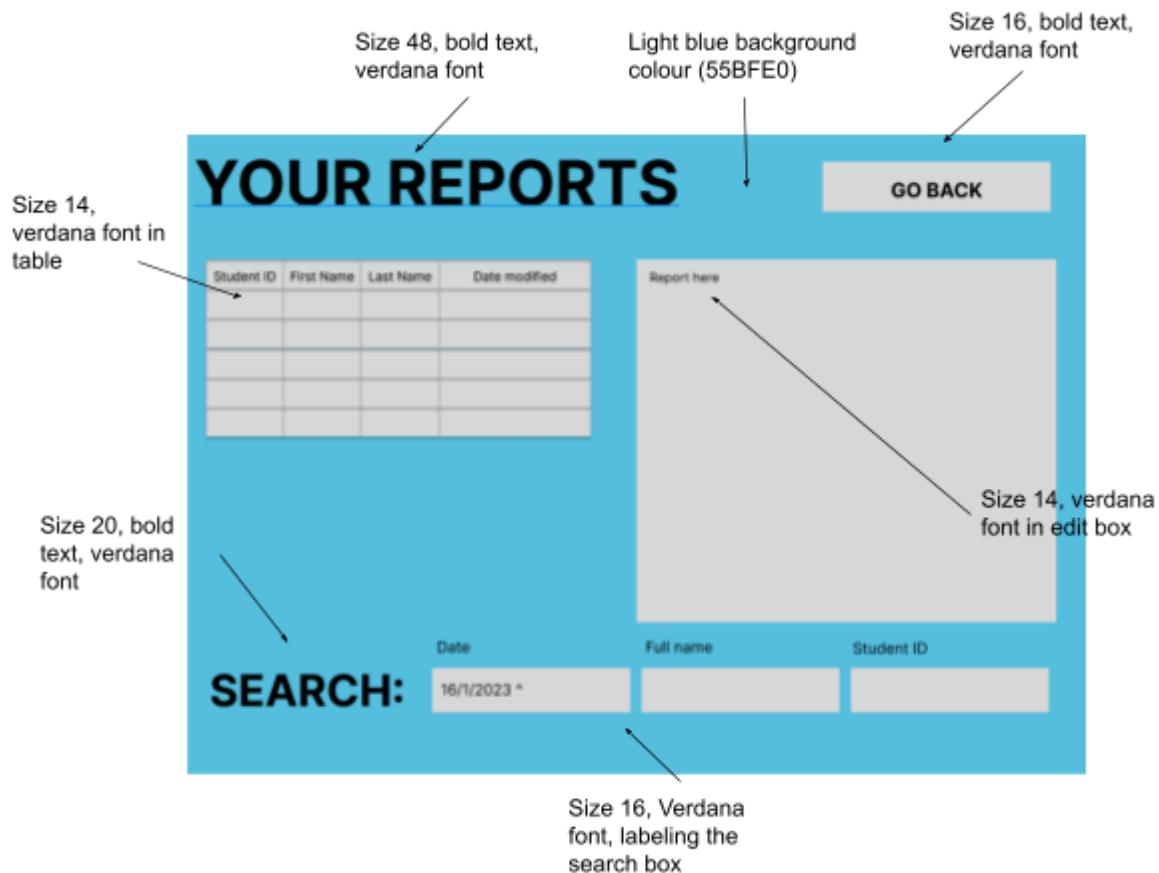
Size 14, verdana font in table

Size 20, bold text, verdana font

First name      Last name      Student ID

**SEARCH:**

Size 16, Verdana font, labeling the search box

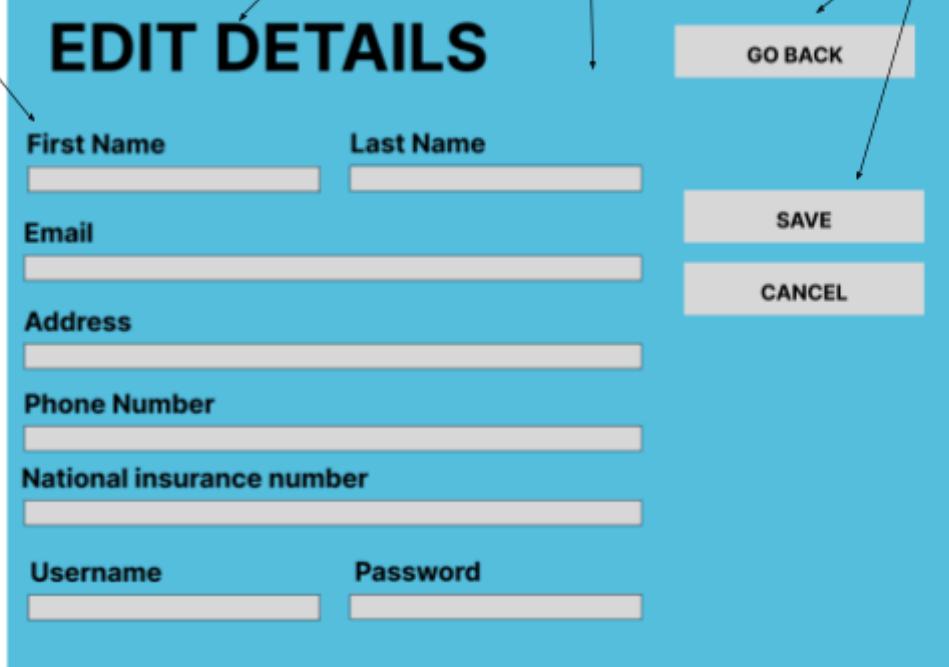
**Form name - Reports****Form name - Edit Details**

Size 16, verdana font

Size 48, bold text, verdana font

Light blue background colour (55BFE0)

Size 16, bold text, verdana font



**EDIT DETAILS**

**First Name** **Last Name**

**Email**

**Address**

**Phone Number**

**National insurance number**

**Username** **Password**

**GO BACK** **SAVE** **CANCEL**

**Form name** - Generate Report

Size 20, bold text, verdana font

Size 48, bold text, verdana font

Light blue background colour (55BFE0)

Size 16, bold text, verdana font

Size 14, verdana font displaying list of students

List of students here

Full name

**SELECT STUDENT**

**SEARCH:**

**GO BACK**

Several boxes to display student details when selected from list

Size 16, Verdana font

Student ID Year Group Form Group Candidate No.

Parent email Student Email

**CONTINUE**

**Form name** - Report Generator

Size 16, bold text, verdana font

Size 48, bold text, verdana font

Standard radio button with number to label amount

**GENERATE REPORT**

**Generate report for (NAME)**

**GO BACK**

**Include sections**

Include Introduction

Include Conclusion

Mention Grades

Mention attendance and punctuality

Mention achievement and behaviour points

Standard tickbox

Size 9, verdana font

Work Completion      Work Quality

1 2 3 4 5      1 2 3 4 5

Homework Completion      Homework Quality

1 2 3 4 5      1 2 3 4 5

Behaviour      Lesson involvement

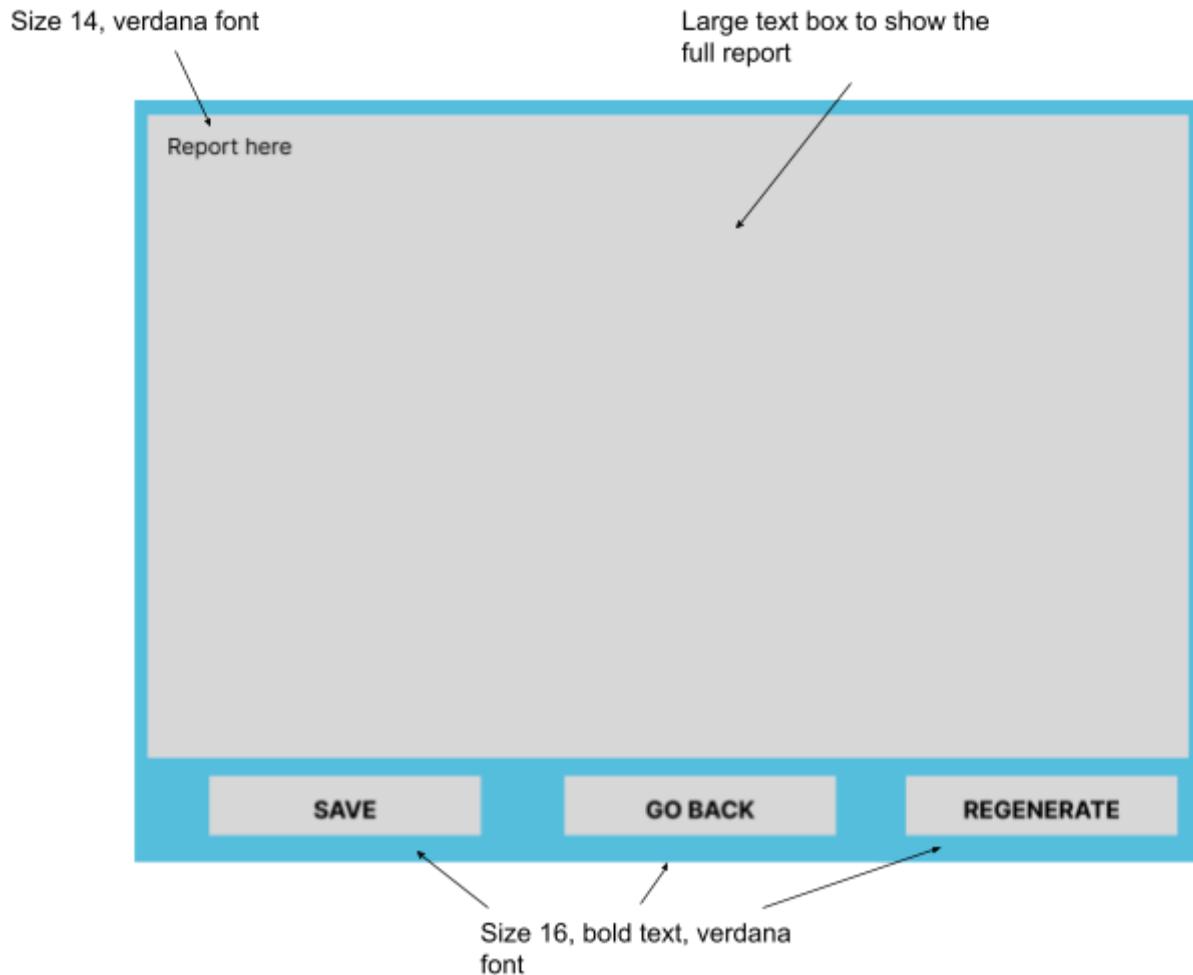
1 2 3 4 5      1 2 3 4 5

Size 16, bold text, verdana font

**GENERATE**

Light blue background colour (55BFE0)

**Form name** - Final Paragraph



## 2.3 Algorithms

The solution will contain several algorithms across each unit that carry out a specific task. Some of these algorithms can be complex and need to be broken down before implementing them. This section will outline the key algorithms that will be used, they will be displayed in pseudocode. Additionally, this section will contain the planned SQL queries, there are many SQL queries that connect the solution to the database so it is important to plan these queries before implementing to save time and reduce errors. By having these algorithms and SQL queries as reference to look back to it will speed up the development time. For each there is an explanation of the purpose and a justification for how it will contribute to the solution.

### 2.3.1 SQL Queries

**SQL - Get login fields with the username entered**

SELECT * FROM Login WHERE username = usernameVariable
---

**Purpose and Justification**

This query returns all the fields from the login table where the username field matched the one entered by the user. This is used to validate the login, the program will check if the record count is zero to see if the username exists, it will also get the password field and check if it matches the one entered. Then the staffID field is stored in a variable to be used throughout the program so it knows which staff is logged in.

**SQL - Inserting new staff details into staff table**

INSERT INTO Staff (firstName, lastName, email) VALUES (firstNameVariable, lastNameVariable, emailVariable)
--

**Purpose and Justification**

This query creates a new record in the staff table and inserts the values entered by the staff into it. This is needed for the staff to create an account and for the data to be saved when they next want to log in.

**SQL - Inserting new staff details into login table**

INSERT INTO Login VALUES (usernameVariable, passwordVariable, staffIDVariable)
--

**Purpose and Justification**

This query creates a new record in the login table and inserts the values entered by the staff that would be stored in the login table. This is needed for the staff to create an account and have their login details stored in the database for when they next want to log in.

**SQL - Get staff fields with the staffID**

SELECT * FROM Staff WHERE staffID = staffIDVariable
---

**Purpose and Justification**

This query selects all the fields from the staff table where the staffID matches the currently logged in staff's ID. This is required to display key information on the home page from the staff table.

### **SQL - Get student fields for students that only the staff teaches**

```
SELECT DISTINCT Students.studentID, firstName, lastName, yearGroup,
formGroup, studentEmail, parentEmail, candidateNumber FROM Students INNER
JOIN Reports ON Students.studentID = Reports.studentID WHERE
Reports.staffID = staffIDVariable
```

#### **Purpose and Justification**

This query returns several fields from the students table. It only returns the records which have a matching staff ID in the reports table and where the reports table staffID matches the staffID currently logged in. The result of this is returning all the students that a staff member who is logged in teaches. It won't return other students that they don't teach and have reports for. This is required to populate the student table that the staff member can view. This is so they can see their students and navigate the table looking for any details on the student they want.

### **SQL - Get performance fields for students that only the staff teaches**

```
SELECT DISTINCT Students.studentID, firstName, lastName, attendance,
punctuality, behaviourPoints, achievementPoints FROM Students, Performance,
Reports WHERE (Students.studentID = Performance.studentID) AND
(Reports.studentID = Performance.studentID) AND (Reports.staffID =
staffIDVariable)
```

#### **Purpose and Justification**

This query returns several fields from the performance and students and table. It only returns the records where the studentID field matches across tables and where the staffID matches the staffID currently logged in. The result of this is returning all the students' performance data that a staff member who is logged in teaches. This is required to populate the table that the staff member can view and navigate. They will want to check their students' performance like attendance and punctuality and this SQL will get that information.

### **SQL - Get the grades of all students**

```
SELECT Students.firstName, Students.lastName, Students.studentID,
academicYear, subject, workingAt, target, predicted, endOfYear FROM Grades
INNER JOIN Students ON Students.studentID = Grades.studentID
```

#### **Purpose and Justification**

This query returns several fields from the grades table and the first and last names

from the students table. It returns every student who has a record in the grades table, this is to allow teachers to view all the students and compare them with their students. This will populate the table that the staff members can view showing grades for all the students.

### SQL - Get the grades of all students with a specific grade filter

```
SELECT Students.firstName, Students.lastName, Students.studentID,  
academicYear, subject, workingAt, target, predicted, endOfYear FROM Grades  
INNER JOIN Students ON Students.studentID = Grades.studentID WHERE  
gradeTypeVariable = gradeVariable
```

#### Purpose and Justification

This query, like the previous, returns several fields from the grades table and the first and last names from the students table. It returns every student who has a record in the grades table and whose grade matches the grade variable that the staff can select. This is to allow staff to view all the students and compare them with their students. This will populate the table that the staff members can view showing grades for all the students. The additional purpose of this is to allow staff to filter the grades, if they only want to see students who got an A\* they will be able to, using this query.

### SQL - Update staff details in staff table

```
UPDATE Staff SET firstName = firstNameVariable, lastName = lastNameVariable,  
email = emailVariable, address = addressVariable, phoneNumber =  
phoneNumberVariable, nationalInsuranceNumber =  
nationalInsuranceNumberVariable WHERE staffID = staffIDVariable
```

#### Purpose and Justification

This query updates the staff table, it updates several fields in the record with the staff ID that matches the staffID variable. This is to allow staff to update their personal details if they ever need to. Without this query they won't be able to do this and would be stuck with their details unless they create a new account.

### SQL - Update staff details in login table

```
UPDATE Login SET username = usernameVariable, [password] =
```

passwordVariable WHERE staffID = staffIDVariable
--

**Purpose and Justification**

This query updates the login table, it updates the username and password fields in the record with the staff ID that matches the staffID variable. This is similar to the last query, instead it allows staff to update their login details as well as their personal details.

<b>SQL - Get average attendance from performance table</b>
--

| SELECT AVG(attendance) FROM Performance |

**Purpose and Justification**

This query selects the average attendance from the performance table across all students using the AVG() command. This will be used during the report generating process, the program should get the average attendance to compare with a specific student to see how well they are doing. This will also be used for other variables like punctuality, achievement points but this example is for attendance.

<b>SQL - Insert new report into reports table</b>
---

| INSERT INTO Reports (reportDocument, dateModified, studentID, staffID) VALUES (finalParagraphVariable, currentDateVariable, studentIDVariable, staffIDVariable) |

**Purpose and Justification**

This query inserts the new report into the reports table. It takes the studentID and the staffID as it needs to know which staff wrote it for which student, then inserts the report text into the database. This is needed because staff need to have a place to save their reports, so this automates this process and the staff doesn't have to worry about saving it.

## 2.3.2 Algorithms

**Form name** - Login

<b>Procedure - Validating login details</b>
---

| procedure ValidateLogin SQL Clear |

SQL ADD 'SELECT \* FROM Login WHERE Username = usernameVariable'  
 SQL Open

```
if recordCount == 0 then
  DisplayError('Incorrect Username')
elseif FieldName('password') != passwordVariable then
  DisplayError('Incorrect Username')
else
  staffID = FieldName('staffID')
  Login()
endif
endprocedure
```

### Purpose and Justification

This algorithm gets the username and password from the login table in the database where the username matches the username variable. It then checks if the passwords are equal and if they are it will call the login function and save the staffID variable. This is required to validate the login details and allow staff to login. Therefore this meets requirements 14 and 15.

**Form name** - Sign Up

### Procedure - Check name

```
global canSignUp

procedure CheckName
  array letters ['A', 'B', 'C' ... 'Z']
  if nameVariable.length == 0 then
    canSignUp = false
    DisplayError('Enter first name')
  endif

  for i = 1 to nameVariable.length
    if not nameVariable[i] in letters then
      canSignUp = false
      DisplayError('Name must only contain letters')
    endif
  next i
endprocedure
```

### Purpose and Justification

This algorithm checks the name entered when the user creates an account. It

stores an array of all the letters in the alphabet, in delphi you can use a set but here it is defined as an array. It then loops through each letter in the name variable to check if the letter is in the letters set, if it isn't the program displays an error saying name must only contain letters. Additionally, if the name variable is empty it returns an error telling the user they must enter a username. This is required because the name must be a correct standard format that can be added to the database and displayed throughout the program properly. Therefore this meets requirements 17 and 18.

### Procedure - Check email

```

global canSignUp

procedure CheckEmail
    array illegalCharacters = ['!', '#', '&', '*', '+', '/', '=', '?', '^', '_', '"', '{', '|', '}', '~']
    containsAtSymbol = false
    indexOfAtSymbol = 0
    dotAfterAtSymbol = false
    maxEmailLength = 320

    for i = 1 to emailVariable.length
        if emailVariable[i] == '@' then
            indexOfAtSymbol = i
            containsAtSymbol = true
        endif
    next i

    if not containsAtSymbol then
        canSignUp = false
    endif

    if canSignUp then
        if indexOfAtSymbol == 1 or indexOfAtSymbol == emailVariable.length then
            for i = indexOfAtSymbol + 1 to emailVariable.length
                if emailVariable[i] == '.' then
                    dotAfterAtSymbol = true
                endif

                if emailVariable[i] in illegalCharacters then
                    canSignUp = false
                endif
            next i
        endif
    endif

```

```
if emailVariable[1] == '.' then
    canSignUp = false
endif

if emailVariable.Contains(..) then
    canSignUp = false
endif
endif

if not dotAfterAtSymbol then
    canSignUp = false
else
    if emailVariable[idexOfAtSymbol + 1] == '.' or
emailVariable[emailVariable.length] == '.' then
        canSignUp = false
    endif

    if emailVariable[idexOfAtSymbol + 1] == '-' then
        canSignUp = false
    endif
endif

if emailVariable.length > maxEmailLength then
    canSignUp = false
endif

if canSignUp then
    Continue()
else
    DisplayError('Invalid email format')
endif
endprocedure
```

#### Purpose and Justification

This algorithm validates the email entered by the user during the account creation section. It thoroughly checks each section of the email and ensures a false email cannot be entered. It follows the standard email format used by RFC 5322 internet message format. It checks if the email contains an '@' symbol, it then checks if a dot is after the '@' symbol. If there is a dot right after the dot after the '@' symbol or a dot right at the end it is invalid. If there are two dots next to each other or a dot at the start it is invalid. There is also an array of illegal characters that cannot be after the '@' symbol but can be before. A max length of 320 characters is set. This is required because the proper email format must be entered into the database and staff members should provide their real emails. If for example their

email address is added to the front of the report it must be legit so parents can contact them if needed. Therefore this meets requirements 21.

## **Procedure - Check username**

```
global canSignUp

procedure CheckUsername
    array illegalCharacters = ['&', '=', '_', '-', '+', '.', '<', '>', ' ']
    
    if usernameVariable.length < 6 or usernameVariable.length > 30 then
        canSignUp = false
        DisplayError('Username must be between 6 and 30 characters')
    endif

    for i = 1 to usernameVariable.length
        if usernameVariable[i] in illegalCharacters then
            canSignUp = false
            DisplayError('Username has illegal characters')
        endif
    next i
endprocedure
```

## **Purpose and Justification**

This algorithm validates the username entered by the user in the account creation section. It contains an array of illegal characters that cannot be included in the username. It loops through the name and if any character equals the illegal character then an error message is displayed. There is also a check to see if the username is between 6 and 30 characters. It must be at least 6 to be secure enough and no longer than 30, otherwise users will enter usernames that are too long and difficult to manage, so this is a realistic length. Therefore this meets requirements 20.

## **Procedure - Check password**

```

if passwordVariable.length < 8 then
    canSignUp = false
    DisplayError('Password must be at least 8 characters')
endif

for i = 1 to passwordVariable.length
    if passwordVariable[i] in requiredCharaters then
        hasRequiredCharacter = true
    endif
next i

if not hasRequiredCharacter then
    canSignUp = false
    DisplayError('Password must contain a special character')
endif
endprocedure

```

### Purpose and Justification

This algorithm checks the password entered by the user in the account creation section. It checks to see if the password is less than 8 characters, it also checks to make sure there is a required character in the password. These validations ensure the password entered is secure enough and reduces the chance of someone hacking into an account. If it does not contain a special character an error message is displayed. If there are less than 8 characters an error message is also displayed telling the user specifically what the issue is. Therefore this meets requirements 19.

### Procedure - Check SQL and insert new sign up values

```

procedure CheckSQL
    sqlStatement = ""
    staffID = ""

    sqlStatement = 'SELECT * FROM Login WHERE Username =
usernameVariable'

    SQL Clear
    SQL Add sqlStatement
    SQL Open

    if recordCount == 0 then
        sqlStatement = 'INSERT INTO Staff (firstName, lastName, email) VALUES
(firstNameVariable, lastNameVariable, emailVariable)'

```

SQL Clear  
 SQL Add sqlStatement  
 ExecSQL

```
sqlStatement = 'SELECT * FROM Staff WHERE firstName =
firstNameVariable'
```

SQL Clear  
 SQL Add sqlStatement  
 SQL Open

```
staffID = FieldName('staffID')
```

```
sqlStatement := 'INSERT INTO Login VALUES (usernameVariable,
passwordVariable, staffIDVariable)'
```

SQL Clear  
 SQL Add sqlStatement  
 SQL Open

```
ShowMessage('Your account has been made')
else
  DisplayError('Username already exists')
endif
endprocedure
```

### Purpose and Justification

This algorithm inserts the validated details that the staff have entered during sign up into the database. It selects all fields from the login table where the username matches the one entered by the user, it then checks if a record exists, if it does then an error is displayed saying the username already exists. Otherwise the new details are inserted into the staff table, it then takes the staffID and inserts the other new details into the login table. This is required to allow the staff to create an account and sign up successfully. The validation ensures the username does not already exist because there cannot be duplicate usernames. Therefore this meets requirements 16 and 22.

**Form name** - Edit details

### Procedure - Update details

```
procedure UpdateSQL
  sqlUpdate = 'UPDATE Staff SET firstName = firstNameVariable, lastName =
lastNameVariable, email = emailVariable, address = addressVariable,
```

```
phoneNumber = phoneNumberVariable, nationalInsuranceNumber =
nationalInsuranceNumberVariable WHERE staffID = staffIDVariable'
```

```
sqlStatementLogin = 'SELECT * FROM Login WHERE Username =
usernameVariable AND staffID != staffIDVariable'
```

SQL Clear  
 SQL Add sqlUpdate  
 ExecSQL

SQL Clear  
 SQL Add sqlStatementLogin  
 SQL Open

```
if recordCount == 0 then
  sqlUpdate = 'UPDATE Login SET username = usernameVariable, [password] =
passwordVariable WHERE staffID = staffIDVariable'
```

```
SQL Clear
SQL Add sqlUpdate
ExecSQL
else
  DisplayError('Username already exists')
endif
endprocedure
```

#### Purpose and Justification

This algorithm updates the new details entered by the user in the edit details section. It updates the staff table and sets all the values to the new values the user enters. It then checks if the username already exists by running another SQL that gets all the fields from the login table where the username equals the ones entered by the user. It also must not equal the staff ID of the user since that would be the ones they just entered. If the record count is zero it continues updating the login table, otherwise it displays an error. This meets requirements 33 and 34 because it allows the user to update their details to the database if any important information has changed. It also checks if the username already exists, this ensures there are no duplicate usernames during login.

**Form name** - Report Generator

#### Procedure - Compare student attendance to average

global attendance

```
global averageAttendance
global finalParagraph

procedure CompareAttendance
    if attendance > averageAttendance then
        finalParagraph += 'This student is above average with an attendance of ' +
        attendance
    elseif attendance == averageAttendance then
        finalParagraph += 'This student is exactly on the average with an attendance
        of ' + attendance
    else
        finalParagraph += 'This student is below average with an attendance of ' +
        attendance
    endif
endprocedure
```

#### Purpose and Justification

This algorithm simply takes the student attendance stored in the database and compares it with the average attendance which will be calculated in another function using SQL AVG() that was outlined in 2.3.1. If the attendance of the student is lower than the average the final paragraph should write a suitable message, if the attendance is greater it will again write a suitable message to the final paragraph that will be displayed at the end. This is required because the report generator must take into account all the data in the database and create an accurate report. For now this looks at punctuality however the program will contain similar procedures for attendance, grades and other student information from the database.

## 2.4 Usability features

It is important for the software to have a clear user interface that is easily navigated and fast to use. It should be easy for the user to become familiar with and competent in using the user interface during the first contact with the software. The software has been designed to be consistent throughout, if you understand how to use one form it will allow you to use all forms.

One example of a usability feature is the layout of the recurring buttons, texts and forms. Each form will use a 16:9 aspect ratio and will be positioned at the desktop centre for a consistent experience. Each form will contain a 'Go Back' button positioned at the top right for quick navigation between forms, having it at the same position helps the user remember and quickly click on the button. Each form will

contain a large title at the top so the user knows what the purpose of the form is. There are two forms that will not follow these rules which are the login and sign up form, these forms will have a smaller vertical size and this is to indicate that the user has not entered the main software yet. These forms do not need to be large as they will be used to quickly log in or sign up.

Another example of a usability feature is the consistent font style and form colouring. Throughout the software the Verdana font will be used in either bold or standard styles. Each button will have Verdana size 16 and each title will have Verdana size 48. Every button will be white and every edit box grey, this helps the user distinguish between the two. The background will be a dark blue colour, this gives a clean minimal look and is not distracting for the user. The button sizes are the same throughout and the edit box sizes are mostly the same, some edit boxes are extended in length to show the user that they might have to input longer strings and also allow them to input longer strings.

A third example of a usability feature is the feedback the program gives to the user to show a change has occurred. When in the sign up or login page, hovering over the ‘Sign Up’ or ‘Login’ buttons will switch between the forms, these buttons are going to be smaller and light up red when hovered over by the mouse, this indicates that the user can click on them. When in the sign up form, if a user inputs a field that is invalid it will turn red and a special message will appear. This allows the user to see where the incorrect field they entered was and correct it. When they have completed the sign up form a confirmation box will appear to allow the user to go back and make a change. Once they have signed up another box appears to confirm their sign up, giving them confidence that the program worked as intended and they can sign up.

## 2.5 Key variables & data structures

The solution will contain several forms, each with their own unit. Each unit will contain many variables and data structures. This section will outline the key variables and data structures that are likely to be used by the program. By outlining these key variables it will reduce the time spent implementing and help give an understanding of what needs to be done for each unit.

**Form name - Login**

Variable Name	Data Type	Global/Local Variable	Description & Justification
StaffID	String	Global	This variable stores the ID of the staff member after they log in. It is required because throughout the whole program it must know the staff ID for running SQL code, so this is a global, public variable used throughout the program.
sqlStatement	String	Local	This variable stores the SQL statement for the login query. This is required because the query takes this string as input and returns the username and password to validate login.

**Form name - Sign Up**

Variable Name	Data Type	Global/Local Variable	Description & Justification
canSignUp	Boolean	Global	This variable checks whether the user can sign up or not. Initially it is set to true, then if any of the validations fail it is set to false preventing the user from signing up.

Data structure name	Data Type	Global/Local Variable	Description & Justification
illegalCharacters	Array	Local	Contains a set of characters that cannot be used in a username or first name. This is to ensure the data entered is valid by comparing it to this array.
requiredCharacters	Array	Local	Contains a set of characters that are required for the password. In order to ensure the password is secure enough it must include a required special character, by comparing it to this array this will be achieved.

**Form name - Students**

Variable Name	Data Type	Global/Local Variable	Description & Justification

sqlStatement	String	Local	This variable stores the SQL statement for the SQL query. This is required to display the students on the table for the staff to view.
inputText	String	Local	This variable stores the text inputted by the user when searching in one of the search boxes. It is required for the SQL query to locate a field.

### Form name - Student Performance

Variable Name	Data Type	Global/Local Variable	Description & Justification
sqlStatement	String	Local	This variable stores the SQL statement for the SQL query. This is required to display the students' performance on the table for the staff to view.
inputText	String	Local	This variable stores the text inputted by the user when searching in one of the search boxes. It is required for the SQL query to locate a field.

### Form name - Grades

Variable Name	Data Type	Global/Local Variable	Description & Justification
sqlStatement	String	Local	This variable stores the SQL statement for the SQL query. This is required to display the students' grades on the table for the staff to view.
inputText	String	Local	This variable stores the text inputted by the user when searching in one of the search boxes. It is required for the SQL query to locate a field.

### Form name - Reports

Variable Name	Data Type	Global/Local Variable	Description & Justification

sqlStatement	String	Local	This variable stores the SQL statement for the SQL query. This is required to display the staff's reports for their students on the table for the staff to view.
inputText	String	Local	This variable stores the text inputted by the user when searching in one of the search boxes. It is required for the SQL query to locate a field.
inputDate	String	Local	This variable stores the date inputted by the user when searching for the date they added the report. It is required for the SQL query to locate a field.

**Form name - Edit Details**

Variable Name	Data Type	Global/Local Variable	Description & Justification
sqlStatementStaff	String	Local	This variable stores the SQL statement for the SQL query. It is required for displaying the current account details and also updating the details by inserting into the Staff table
sqlStatementLogin	String	Local	This variable stores the SQL statement for the SQL query. It is required for displaying the current account details and also updating the details by inserting into the Login table
confirmationMessage	String	Local	This variable stores a string that displays the changes made to the user. This is required because the user needs to confirm they are making the right changes.
hasChangedDetails	Boolean	Local	This variable checks whether the user has changed their details, this is required because it prevents the SQL running and inserting the same data into the database.

**Form name - Generate Report**

Variable Name	Data Type	Global/Local Variable	Description & Justification
StudentID	String	Global	Stores the ID of the student the staff selects to write the report for. This is required so the program knows which student it is when inserting to the database and when creating the report.
sqlStatement	String	Local	This variable stores the SQL statement for the SQL query. This is required to display the list of students in the list box.
inputText	String	Local	This variable stores the text inputted by the user when searching for the name of the student. It is required to update the list box selection when the user inputs a name

Data structure name	Data Type	Global/Local Variable	Description & Justification
studentIDs	Array	Global	Stores a list of the student ID's returned from the query. This is required because the staff sees a list of students and the program must know which ID the staff selects, so this array can store that value.

### Form name - Report Generator

Variable Name	Data Type	Global/Local Variable	Description & Justification
firstName, lastName gender	String	Global	These variables store information on the student. These are required so the sentences that are used use the right name and pronouns automatically.
finalParagraph	String	Global	This variable is a string that contains the final paragraph of the report generated. It is required because when saving the report to the database it will use this variable, it is also displayed in a text box for the staff to see

Data structure name	Data Type	Global/Local Variable	Description & Justification
introductionSentences, conclusionSentences, workQualitySentences, homeworkQualitySentences	TSentences	Global	These are objects of type TSentences. This class has 5 variables for the different levels the staff rank the students in the reports. Each is an array of strings storing the sentences that will be randomly selected when creating the report. This custom type was required to reduce the number of variable names by having different objects each with the same variables and running functions on each object, reducing the amount of code significantly.

## 2.6 Test data during the iterative development

This section will include the planned test data that will be used during the iterative development of the solution. It is important to have a series of precise tests to carry out when implementing to ensure the code is working as expected, by planning these tests beforehand it will help speed up implementation time and make sure the solution stays on track. These tests will be specific and test every part of the solution.

Test number	Test purpose	Test data	Expected result	Test Type	Justification
1.1	Check if the sign up text changes colour when the mouse hovers over	Mouse enter event triggering	Sign up text changes to red colour.	Normal	This is to indicate to the user they can go to a sign up page.
1.2	Check if clicking the sign up button opens the sign up page	Sign up button click	Login page closes, sign up page opens.	Normal	If the user doesn't have an account they must be able to open the sign up page.
1.3	Check if valid username and	username: 'davidFrost'	Login page closes, home	Normal	The user must be able to enter the

	password takes user to home page	password: 'frostingtime123'	page opens.		main program once they enter the correct details.
1.4	Check if incorrect username displays an error	username: 'incorrectUsername'	'Incorrect Username' error message appears.	Erroneous	If the username they entered doesn't exist in the database the program should not allow them to login.
1.5	Check if incorrect password displays an error	username: 'j' password: 'incorrectPassword'	'Incorrect Password' error message appears.	Erroneous	If the password they entered doesn't exist in the database the program should not allow them to login even if their username is right.
2.1	Check if invalid first name displays an error message	first name: 'jason123!'	'First name must only contain letters' error message appears.	Erroneous	The first name should only contain letters otherwise it is invalid.
2.2	Check if invalid last name displays an error message	last name: '#@%jackson_'	'Last name must only contain letters' error message appears.	Erroneous	The last name should also only contain letters, otherwise it is invalid.
2.3	Check if empty first name displays an error	first name: ''	'Enter a first name' error message appears.	No data	The user might forget to fill in a field, this is an additional check to ensure they enter a first name and to inform the user they can't leave it empty.
2.4	Check if empty last name displays an error	last name: ''	'Enter a last name' error message appears.	No data	The user might forget to fill in a field, this is an additional check to ensure they enter a last name and to inform the user they can't leave it empty.
2.5	Check if email in the incorrect format displays an error	email: 'jason@email'	'Invalid email format' error message appears.	Erroneous	The email must be in the correct format, the program should check the email thoroughly to

					ensure it is legit and the user's real email address. If the email is used in the program at some point it must be valid.
2.6	Check if username is more than 30 characters	username: 'thislongusernamaiswaytoolongandnobodywoulduseit'	'Username between 6 and 30 characters' error message appears.	Invalid	The username they enter must be a realistic length and manageable so it can be stored on the database without causing problems. This also prevents hackers from injecting SQL with a username that is too long.
2.7	Check if username is exactly 6 characters	username: 'Daniel'	N/A	Boundary	To ensure the program is working as intended, testing at the boundary is important, it needs to be exact and it should not confuse the user what the range is. Since the username can be 6 characters, it should be allowed and there should be no error message.
2.8	Check if username is exactly 30 characters	'DanielScottAtTrentlockSchool1'	N/A	Boundary	To ensure the program is working as intended, testing at the boundary is important, it needs to be exact and it should not confuse the user what the range is. Since the username can be 30 characters, it should be allowed and there should be no error message.
2.9	Check if username has illegal characters	'<Daniel>&&'	'Username has illegal characters'	Erroneous	The username must be a standard format and not

			error message appears.		contain any special characters.
2.10	Check if password has less than 8 characters	'123'	'Password must be at least 8 characters' error message appears.	Invalid	The password must be at least 8 characters because it is safer and reduces the chance of a user getting their account stolen or hacked.
2.11	Check if password has exactly 8 characters	'passwor!'	N/A	Boundary	To ensure the program is working as intended, testing at the boundary is important, it needs to be exact and it should not confuse the user what the range is. The password must be at least 8 characters so we must test if the language is correct by testing exactly 8 characters, this is within 8 characters so the program should not give an error message.
2.12	Check if password has more than 8 characters	'thelonelysoldier123\$'	N/A	Normal	The password must be at least 8 characters, if they have entered a password more than 8 characters this is normal data and should be allowed so the program should not display an error message.
2.13	Check if password does not contain a special character	'thelonelysoldier123'	'Password must contain a special character' error message appears.	Erroneous	The password must contain a special character because it is safer and reduces the chance of a user getting their account stolen or hacked.

2.14	Check if clicking the sign up button shows confirmation message	Sign up button click	Confirmation box should open.	Normal	In case the user has made a mistake or wants to go back and change a detail they entered, the program will ask them to confirm. This gives them time to ensure everything they entered was correct and confirm to the program they want to continue.
2.15	Check if program displays error if username already exists	'qwert'	'Username already exists' error message appears.	Normal	The program requires a username and password to be entered when logging in, therefore these fields must be unique so there is no duplication in the database and so there can be unique users, if the username already exists the program should display an error message and allow the user to re-enter their username.
2.16	Check if clicking yes in the confirmation box returns to login page	'Yes' button click	Sign up page closes, login page opens.	Normal	Once the user has confirmed they must be taken back to the login page so they can use their newly created account to login right away.
3.1	Check if sign out buttons signs the user out	Sign out button click	Home page closes, login page opens.	Normal	The user must be able to sign out whenever they want, this test ensures the button works as intended and correctly goes back to the login page.
3.2	Check if quit	Quit button	Program	Normal	If the user is done

	button closes the program	click	closes.		using the program they should be able to quit the application, this test ensures the quit functionality works as intended.
3.3	Check, staff ID, email and full name display on home page	username: 'davidfrost' password: 'davidfrost!'	'Name - David frost' 'Email - davidFrost@gmail.com' 'Staff ID - 2'	Normal	Test data will be added to the database, a user by the name of david frost will be added with a staff ID of 2 to test this section correctly.
3.4	Check if 'students' button opens student page	Students button click	Home page closes, students page opens	Normal	Users must be able to navigate other forms from the home page by clicking the button.
3.5	Check if 'student stats' button opens student performance page	Student stats button click	Home page closes, student performance page opens	Normal	Users must be able to navigate other forms from the home page by clicking the button.
3.6	Check if 'grades' button opens grades page	grades button click	Home page closes, grades page opens	Normal	Users must be able to navigate other forms from the home page by clicking the button.
3.7	Check if 'reports' button opens reports page'	reports button click	Home page closes, reports page opens	Normal	Users must be able to navigate other forms from the home page by clicking the button.
3.8	Check if 'edit details' button opens edit details page'	edit details button click=	Home page closes, edit details page opens	Normal	Users must be able to navigate other forms from the home page by clicking the button.
3.9	Check if 'generate report' button opens generate report page	generate report button click	Home page closes, generate report page opens	Normal	Users must be able to navigate other forms from the home page by clicking the button.
4.1	Check if go back button works on	go back button click	Students page closes, home	Normal	User must be able to go back to the

	students page		page opens		home page from any form
4.2	Check if student table displays correctly when opening the form, showing staffs students	staff ID: 2	Table should be populated with the students that staff with staff ID 2 search	Normal	This test is to ensure the table is correctly displayed when the staff open the page.
4.3	Check if first name search box correctly updates the table	first name: 'Jerry'	Table selection updates to locate search input	Normal	This test is to ensure the search box correctly searches for the first name, this is required so the teacher can search for the student name
4.4	Check if last name text box correctly updates the table	last name: 'Leon'	Table selection updates to locate search input	Normal	This test is to ensure the search box correctly searches for the last name, this is required so the teacher can search for the student name
4.5	Check if student ID search box correctly updates the table	Student ID: 14	Table selection updates to locate search input	Normal	This test is to ensure the search box correctly searches for the student ID number, this is required so the teacher can search for the ID quickly
4.6	Check if clicking column header sorts the column from ascending to descending or descending to ascending	N/A	Data in column should be resorted from ascending or descending	Normal	This test ensures the user can search the table by sorting for ascending to descending for any column they want
5.1	Check if go back button works on student performance page	go back button click	Student performance page closes, home page opens	Normal	This test is required to check if the user can go back to the home page and not get stuck on this page

5.2	Check if performance table displays correctly when opening the form, showing performance of students	staff ID: 2	Table should be populated with the student performance that staff with staff ID 2 search	Normal	This test is to ensure the table is correctly displayed when the staff open the page.
5.3	Check if first name search box correctly updates the table	first name: 'Abraham'	Table selection updates to locate search input	Normal	This test is to ensure the search box correctly searches for the first name, this is required so the teacher can search for the student name
5.4	Check if last name text box correctly updates the table	last name: 'Mckenzie'	Table selection updates to locate search input	Normal	This test is to ensure the search box correctly searches for the last name, this is required so the teacher can search for the student name
5.5	Check if student ID search box correctly updates the table	Student ID: 20	Table selection updates to locate search input	Normal	This test is to ensure the search box correctly searches for the student ID number, this is required so the teacher can search for the ID quickly
5.6	Check if clicking column header sorts the column from ascending to descending or descending to ascending	N/A	Data in column should be resorted from ascending to descending	Normal	This test ensures the user can search the table by sorting for ascending to descending for any column they want
6.1	Check if go back button works on grades page	go back button click	Grades page closes, home page opens	Normal	This test is required to check if the user can go back to the home page and not get stuck on this page

6.2	Check if grades table displays correctly when opening the form	N/A	Table should be populated with the grade information for all students	Normal	This test is to ensure the table is correctly displayed when the staff open the page.
6.3	Check if first name search box correctly updates the table	first name: 'Barret'	Table selection updates to locate search input	Normal	This test is to ensure the search box correctly searches for the first name, this is required so the teacher can search for the student name
6.4	Check if last name text box correctly updates the table	last name: 'Green'	Table selection updates to locate search input	Normal	This test is to ensure the search box correctly searches for the last name, this is required so the teacher can search for the student name
6.5	Check if student ID search box correctly updates the table	Student ID: 3	Table selection updates to locate search input	Normal	This test is to ensure the search box correctly searches for the student ID number, this is required so the teacher can search for the ID quickly
6.6	Check if clicking column header sorts the column from ascending to descending or descending to ascending	N/A	Data in column should be resorted from ascending or descending	Normal	This test ensures the user can search the table by sorting for ascending to descending for any column they want
6.7	Check if 'working at' drop down box correctly updates table	target grade: 'B'	Table should be sorted and should now only show students with working at grade B	Normal	This test is required because the filter grades option must work, this will allow staff to quickly find the information they need
7.1	Check if go back button works on reports page	go back button click	Reports page closes, home page opens	Normal	This test is required to check if the user can go back to

					the home page and not get stuck on this page
7.2	Check if reports table displays correctly when opening the form	N/A	Table should be populated with the reports table from the database	Normal	This test is to ensure the table is correctly displayed when the staff open the page.
7.3	Check if full name search box correctly updates the table	first name: 'Kassidy'	Table selection updates to locate search input	Normal	This test is to ensure the search box correctly searches for the full name, this is required so the teacher can search for the student name
7.4	Check if student ID search box correctly updates the table	Student ID: 11	Table selection updates to locate search input	Normal	This test is to ensure the search box correctly searches for the student ID number, this is required so the teacher can search for the ID quickly
7.5	Check if date picker search box correctly updates the table	date: 21/3/2023	Table selection updates to locate search input	Normal	This test ensures the staff can search for a date, this allows them to search much quicker
7.6	Check if reports are displayed in the rich edit text box correctly	N/A	Rich edit box should be populated with the report document from the database	Normal	This test is to ensure the reports are correctly displayed on the right, this is required because the staff must be able to view their reports and edit them.
7.7	Check if clicking column header sorts the column from ascending to descending or descending to ascending	N/A	Data in column should be resorted from ascending or descending	Normal	This test ensures the user can search the table by sorting for ascending to descending for any column they want

8.1	Check if go back button works on edit details page	go back button click	Edit details page closes, home page opens	Normal	This test is required to check if the user can go back to the home page and not get stuck on this page
8.2	Check if cancel button reverts the edit boxes to the current details	Cancel button click	Edit boxes should revert to previous data they were displaying	Normal	This test is required because the staff must be able to cancel their changes if they input something they want to revert or if its a mistake
8.3	Check if program displays error if changes have not been made	Save button click	'You have not changed your details, cannot save' error message appears.	Normal	This test is required because the staff should not be able to save to the database if the data is the same, so this ensures that cannot happen.
8.4	Check if confirmation box appears when user clicks save	new first name: 'test' new last name: 'test2'	Confirmation box appears, it outlines the changes that have been made which should be what the test data is	Normal	This test is required to ensure the confirmation box functionality is working correctly.
9.1	Check if go back button works on generate report page	go back button click	generate report page closes, home page opens	Normal	This test is required to check if the user can go back to the home page and not get stuck on this page
9.2	Check if continue button opens report generator page	continue button click	generate report page closes, report generator page opens	Normal	This test ensures the user can go to the next page after clicking continue
9.3	Check if continue button opens report generator page when there is no student selected	continue button click  selected student: null	Next page should not open because a student has not been selected or is not in the list	Normal	This test is required to ensure the staff cannot create a report when they have not selected a student, otherwise this would cause errors

9.4	Check if search box correctly updates list box selection	search input: 'Peter'	List box selection should update and be selected on the name searched which with this test is 'Peter'	Normal	This test is required because the staff must be able to quickly search for their student in a large list of students
9.5	Check if display boxes update to display selected student	search input: 'Amari'	display boxes should display information for Amari: student ID: 15, year group: 8	Normal	This test ensures the display boxes work as intended and are displayed when a student is selected.
10.1	Check if go back button works on report generator page	go back button click	report generator page closes, generate report page opens	Normal	This test is required to check if the user can go back to the previous and not get stuck on this page
10.2	Check if student name is displayed at the top of the page	student name selected: 'Peter Lara'	'Generate report for Peter Lara' should be displayed at top of form	Normal	This test is required because the staff must be able to see the name of the student they are writing the report for, if this doesn't work the label would display something default which is not intended
10.3	Check if error message is displayed when radio groups have not been filled in	radio groups current index: -1	'You must fill in each section' error message should appear	Normal	This test is required to ensure the program does not create the report when no inputs have been selected. By default the radio buttons are at a value of -1 so it would cause errors in the program

## 2.7 Further data to be used in the post development stage

This section will test the system as a whole from start to finish. A number of realistic scenarios will be tested like a staff member signing up and updating their personal details or a staff member logging in, generating a report and saving it to the database. This will be used once implementation is complete to further test the system as a completed solution. As well as system testing there will be user testing, this is the process through which the interface and functionality of the system are tested by the end-user who performs specific tasks in realistic conditions. User testing will be achieved with a questionnaire for the staff Mr. Wadud to complete.

## 2.7.1 System testing

Test number	Test purpose and justification	Test data	Expected result
1.	Testing the process of creating an account, logging in and editing account details.	first name: 'Imran' last name: 'Wadud' email: <a href="mailto:lwadud@trentlock.com">lwadud@trentlock.com</a> username: 'imranwadud' password: 'imranwadud!'  address: 'SW15 9QS'  phone number: 071765422214	Staff launches program and is displayed the login page  The staff clicks sign up and enters their details into the text boxes. Customer clicks sign up and a confirmation message appears, then returning them to the login page  Staff logs in with their details  Staff clicks on edit details page, then inputs the details they want to add or update  Staff clicks save and details are saved to database
2.	Testing the process of a staff member logging in, viewing their students in the students page and searching for student name, viewing the student performance in the student performance page and searching for student ID, viewing the grades in the grades page and filtering grades.	username: 'imranwadud' password: 'imranwadud!'  student name search: 'Peter'  student ID search: 10  workingAtGrade: 'B'	User launches program and is displayed with the login page  Staff enters their login details and clicks login  Staff clicks students button and is taken to students page  Staff searches for student name and table is updated  Staff clicks go back and is taken back to home page  Staff clicks student performance button and is taken to student

			<p>performance page</p> <p>Staff search for student ID and the table is updated</p> <p>Staff clicks go back and is taken back to home page</p> <p>Staff clicks grades button and is taken to grades page, they click the filter drop down box and select a filter which updates the table</p>
3.	Testing the process of a staff member logging in, going to the reports page, searching for a student name, then editing the report they have for that student.	username: 'imranwadud' password: 'imranwadud!'  student name search: 'Jamarion'	<p>Staff launches program and is displayed with the login page</p> <p>Staff logs in with their login details and is taken to home page</p> <p>Staff clicks reports button and is taken to reports page</p> <p>Staff searches for report by date or by student name and table is updated</p> <p>Staff edits the report text box and it is saved to the database. The date modified should also be updated to today</p>
4.	Testing the process of a staff member logging, generating a report for their student and saving it to the database.	username: 'imranwadud' password: 'imranwadud!'  student name search: 'Barret'  Work completion: 4 Work quality: 4 Homework completion: 5 Homework quality: 4 Behaviour: 4 Lesson involvement: 5  Introduction: unticked Conclusion: unticked Grades: unticked Attendance & Punctuality: ticked	<p>Staff launches program and is displayed with login page</p> <p>Staff enter their login details and is taken to the home page</p> <p>Staff click generate report button and are taken to generate report page</p> <p>Staff search student name or scroll through list of students to select a student</p> <p>Staff click continue and the report generator page opens.</p> <p>Staff input values from 1-5 from radio groups and tick check boxes to pick if the report includes key sections</p> <p>Staff click the generate button and</p>

		<p>behaviour and achievement points: ticked</p>	<p>the final paragraph page opens. Report is generated and displayed in a large text box.</p> <p>Staff click regenerate button to regenerate the report, updates the report in the text box</p> <p>Staff edit text box to make final changes</p> <p>Staff clicks save button and report is saved to the database</p>
--	--	---	--

## 2.7.2 User testing

Questions that will be asked:

1. Does the login screen open when the program is launched?
2. Does the sign up button open the sign up page?
3. How easy is it to sign up?
4. Is the UI clear and easy to understand?
5. Does the program inform you if you enter an incorrect field?
6. Does the program save your details after signing up?
7. Can you successfully log in?
8. Do you understand where each button takes you and what the purpose of each form is?
9. How easy is it to navigate the system from 1 to 10?
10. Can you easily edit your details?
11. Does the students table contain all the required information?
12. Are the tables easy to understand and use?

12. Can you clearly see your reports in the reports page?
13. How easy is it to search for a student or report?
14. How easy is it to select a student and generate a report for them?
15. How easy is it to save the report to the database?
16. How easy is it to create a new report for a new student or regenerate your current report?
17. Have you encountered any bugs or errors?

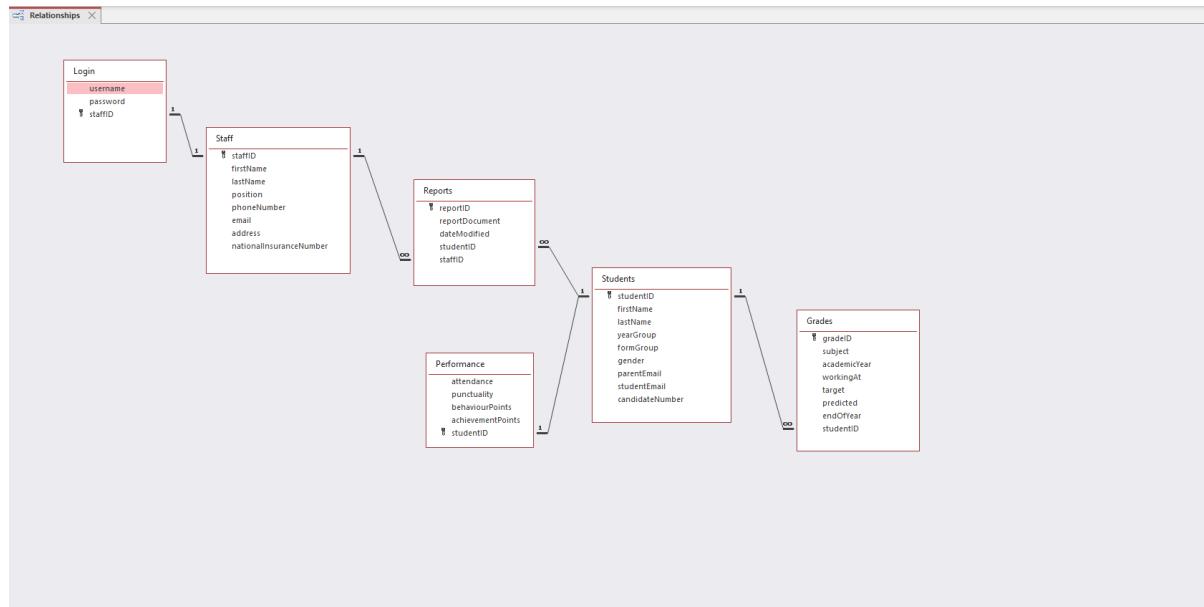
## 3. Developing the coded solution

This section will display the iterative development of the coded solution. It will contain each stage of the iterative development process by using prototypes. Each prototype version will aim to meet a set of requirements and improve upon the last prototype. The tests required for each prototype will also be completed with results. The accomplishments of each prototype and a summary of the project at that stage will be explained throughout to keep track of the progress being made. Each prototype will contain a GUI and annotated code. All variables and data structures including any key validation will be stated and explained.

### 3.1 Stage 1 Database

#### 3.1.1 Prototype 1

##### Database screenshot



## Database accomplishments

The school database has been created. This database is fully normalised using 3rd normal form. It is a relational database containing 6 tables, each connected with primary and foreign keys. This will be accessible to staff members through abstraction when using the program.

## Requirements met

This meets requirements 4 and 24.

Number	Feature	Explanation and justification
4.	Microsoft Access 2000 or later installed.	The solution will use a school database, this database is crucial for the solution to work and will contain important information on the staff and the students, without it the solution cannot function. The database will be in Microsoft Access 2000 version so this must be installed on the systems. The stakeholders already own Microsoft products including Access so this comes as no additional cost to the school.

24.	All databases must be in 3rd normal form.	This is a basic requirement for any database because it prevents any data duplication and errors when accessing using SQL.
-----	---	--

### Testing

No testing needs to be carried out for this prototype.

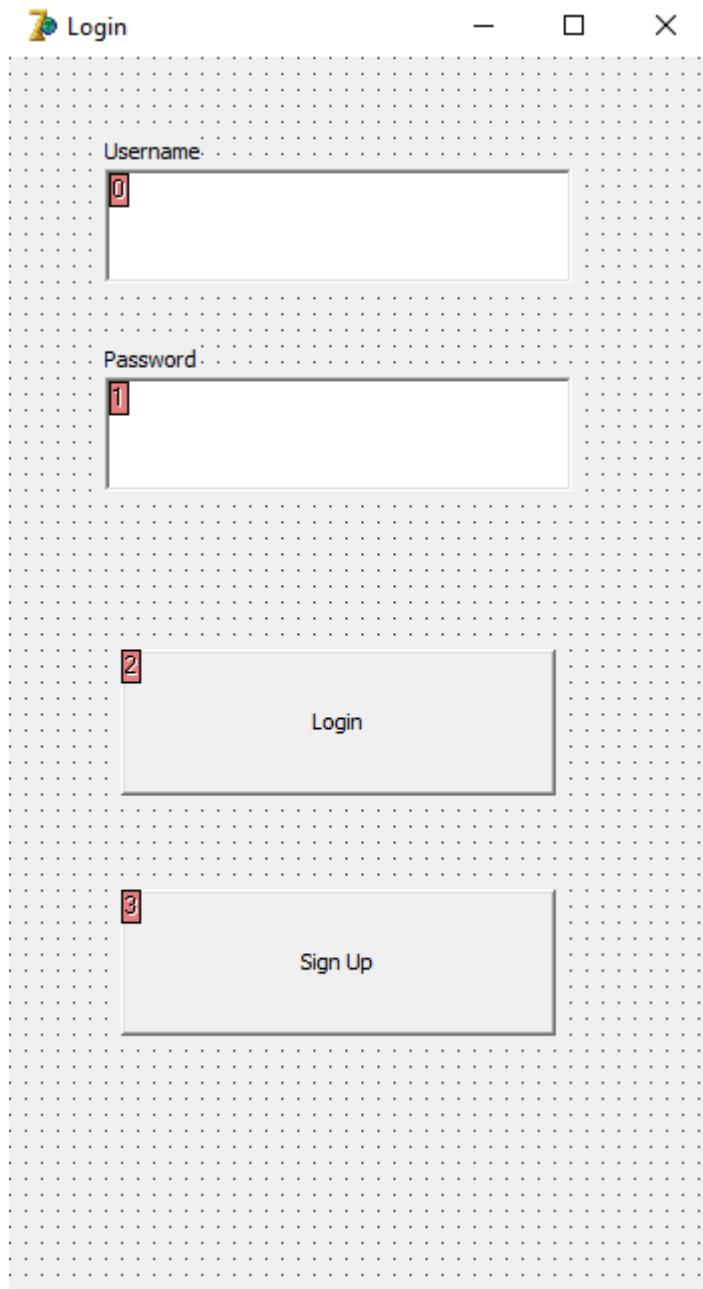
### Summary

The database has now been created. There is no actual program at this stage, however now that the database has been created the program can be implemented and will be able to connect with the database. The database is already at a high standard and a completed stage so this prototype will be the only prototype required.

## 3.2 Stage 2 login form

### 3.2.1 Prototype 1

#### Form screenshot



## Form accomplishments

The login form has been created, it contains a labelled username edit box and a labelled password edit box for the user to enter their username and password. It also contains two buttons, a login button and a sign up button. Currently the form is not linked to the database and the buttons do not do anything when clicked. The basic design has now been laid out.

## Code

**unitLogin**

```
unit unitLogin;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls;

type
  TfrmLogin = class(TForm)
    // username and password variables have been defined, login and sign up button
    // variables have been defined

    IbledtUsername: TLabledEdit;
    IbledtPassword: TLabledEdit;
    btnLogin: TButton;
    btnSignUp: TButton;

    private
      { Private declarations }
    public
      { Public declarations }
    end;

var
  frmLogin: TfrmLogin;

implementation

{$R *.dfm}

end.
```

**Code accomplishments**

With each form there is a unit where the code is written, here is the completed unit for the first prototype. This unit contains the default unit layout delphi gives you when creating the form including the frmLogin variable that other units can reference to access variables in this unit. There is no implementation for this prototype except for two labelled edits and two buttons defined at the class level.

## Requirements met

This meets requirement 6.

Number	Feature	Explanation and justification
6.	There must be a login and sign up form with a small vertical window size.	The user should be greeted with a login page and have the option to sign up which will take them to a sign up page. These forms should be smaller than the main forms as they don't need to be large since they aren't displaying lots of information. It is also indicating to the user they have not entered the main program yet.

## Testing

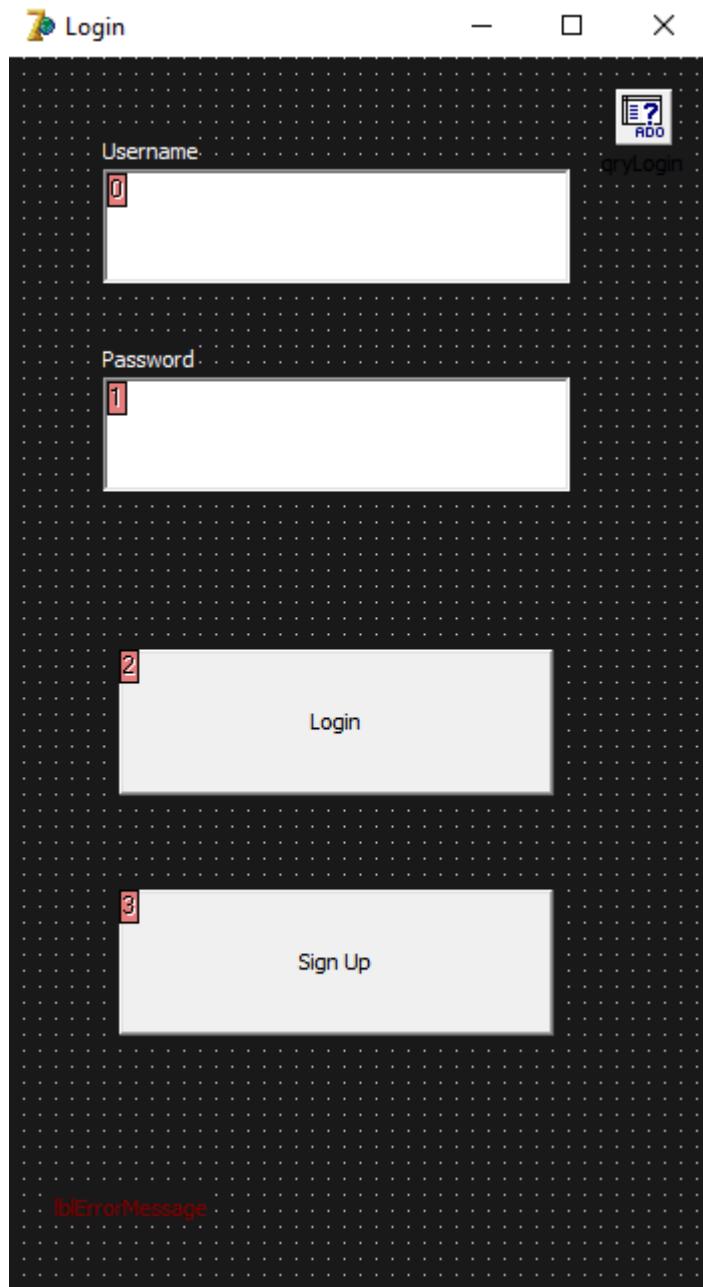
No testing to be carried out.

## Summary

The program can be launched, the user will be shown the login page, they can input their username and password in the edit boxes, they can choose to click on sign up or click login once they have entered their username and password.

## 3.2.2 Prototype 2

### Form screenshot



## Form accomplishments

The login form remains mostly the same for the second prototype, some design changes have been made, a black background has been added to replace the default light background colour. This improves the look and fits the requirements. There is also a red error message at the bottom left of the form for when the program needs to display an error. This is set to invisible when the program starts and then turns visible when displaying an error. Finally an ado query has been placed at the top right of the form, this is not visible when the program runs but

needs to be an object within the form to connect to the database so it can be used in the code.

## Code

### unitLogin

```
unit unitLogin;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, DB, ADODB, unitHomePage, unitSignUp;

type
  TfrmLogin = class(TForm)

// username and password variables defined, login and sign up variables defined.
login query and error message label defined

  IbledtUsername: TLabledEdit;
  IbledtPassword: TLabledEdit;
  btnLogin: TButton;
  btnSignUp: TButton;
  qryLogin: TADOQuery;
  IblErrorMessage: TLabel;

// Two events triggered when the user presses login or sign up button

  procedure btnLoginClick(Sender: TObject);
  procedure btnSignUpClick(Sender: TObject);

// public variable, this is set to the staffID found in the database once the user logs
in, it is public so other units can access the variable and always know the staffID

public
  StaffID : string;

// 3 private procedures, GetUsernamePassword() is called first and then calls
Login() or displayError() depending on the result, this keeps the code clean and
easy to make changes as each procedure is independent and has a single
responsibility
```

```
private
  procedure DisplayError(errorMessage : string);
  procedure GetUsernamePassword;
  procedure Login;
end;

var
  frmLogin: TfrmLogin;

implementation

{$R *.dfm}

// When login button is clicked it calls the GetUsernamePassword() procedure to start the login validation

procedure TfrmLogin.btnLoginClick(Sender: TObject);
begin
  GetUsernamePassword;
end;

// Hides the current form and opens the sign up form

procedure TfrmLogin.btnSignUpClick(Sender: TObject);
begin
  Hide;
  frmSignUp.Show;
end;

// This procedure has a parameter called errorMessage, this allows for the error message to change depending on the error and saves code. It sets the label to the errorMessage and sets it to visible

procedure TfrmLogin.DisplayError(errorMessage : string);
begin
  lblErrorMessage.Caption := errorMessage;
  lblErrorMessage.Visible := true;
end;

// local variable called sqlStatement stores the sql statement that will be run, it gets all fields from the login table where the username is the one entered in the username box

procedure TfrmLogin.GetUsernamePassword;
var
```

```
sqlStatement : string;
begin
    sqlStatement := 'SELECT * FROM Login WHERE Username = ' +
QuotedStr(lbledtUsername.Text);

// Opens the query login with the sqlStatement

with qryLogin do begin
    Close;
    SQL.Clear;
    SQL.Add(sqlStatement);
    Open;

// If there is no record then the username must be incorrect so it will call the display error procedure, otherwise if the password does not match it calls the display error procedure with a different error message, finally if it is all valid, the login procedure is called and the staffID variable is set to the staffID from the login table

if RecordCount = 0 then
    DisplayError('Incorrect Username')
else if FieldByName('password').AsString <> lbledtPassword.Text then
    DisplayError('Incorrect Password')
else begin
    StaffID := FieldByName('staffID').AsString;
    Login;
end;
end;
end;

// Closes login page and opens home page

procedure TfrmLogin.Login;
begin
    Hide;
    frmHomePage.Show;
end;

end.
```

## Code accomplishments

The code has been implemented a lot further since the first prototype where it was mostly the default script. Three new variables have been added at the class level:

qryLogin, lblErrorMessage and staffID. qryLogin connects to the database and selects values from the login table. lblErrorMessage is a label in the form that displays an error message. This unit and all other units will make use of object oriented programming, it has public and private variables to control the access level of the variables, any variable or function that needs to be defined will be defined privately and a few will be defined publicly so that other classes can read and write the variable. staffID is a public variable so other units will be able to access the staffID of the staff who has just logged in. This form contains two new event functions: btnLoginClick, btnSignUpClick. These are called when the user presses the login or sign up button. Three procedures have been added: DisplayError(), GetUsernamePassword() and Login(). Each has been implemented fully. DisplayError() takes in an error message string and turns the errorMessage label visible and sets it to the string argument. GetUsernamePassword() opens the qryLogin and gets the username and password, compares them to the ones the user inputted and then shows an error or logs in. Login() hides this page and opens the login page.

### Requirements met

This meets requirements 11, 14 and 15.

Number	Feature	Explanation and justification
11.	Each form should have a dark blue background with contrasting white text.	The software should be easy to look at and have a clear layout, by having a dark blue background it creates a good look without being distracting.
14.	Program must have login functionality, staff should be able to login with their own account.	The school wants to store the data of their staff and provide privacy and security to the staff. An account is the best method of doing this, therefore it is best that they have a way to login with their user details.
15.	Program must display an error message to the user if their username or password is incorrect.	This validation is used to make sure the staff know their login details and ensure nobody else can access their account.

## Key Validation

Login validation has been implemented, the username and password is taken in and the program selects all fields where the login usernames matches the one in the database, it then checks if the password is equal to the one in the database and then the user can log in.

## Testing

Test numbers 1.2, 1.3, 1.4, 1.5 will be carried out.

Test number	Test purpose	Test data	Expected result	Result
1.2	Check if clicking the sign up button opens the sign up page	Sign up button click	Login page closes, sign up page opens.	Success
1.3	Check if valid username and password takes user to home page	username: 'davidFrost' password: 'frostingtime123'	Login page closes, home page opens.	Success
1.4	Check if incorrect username displays an error	username: 'incorrectUsername'	'Incorrect Username' error message appears.	Success
1.5	Check if incorrect password displays an error	username: 'j' password: 'incorrectPassword'	'Incorrect Password' error message appears.	Success

### 1.2 Result

 frmSignUp**1.3 Result** frmHomePage**1.4 Result**

Login X

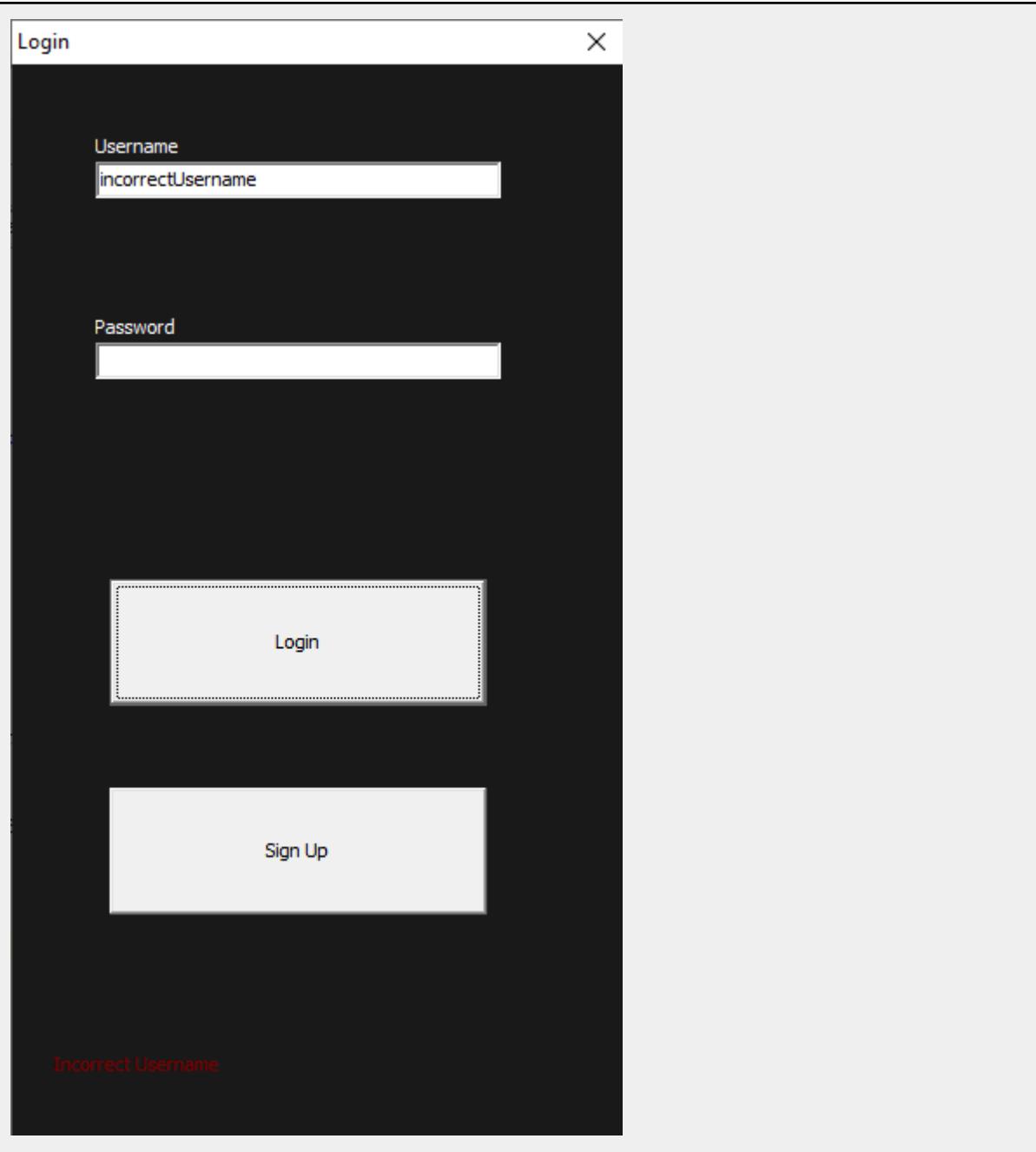
Username

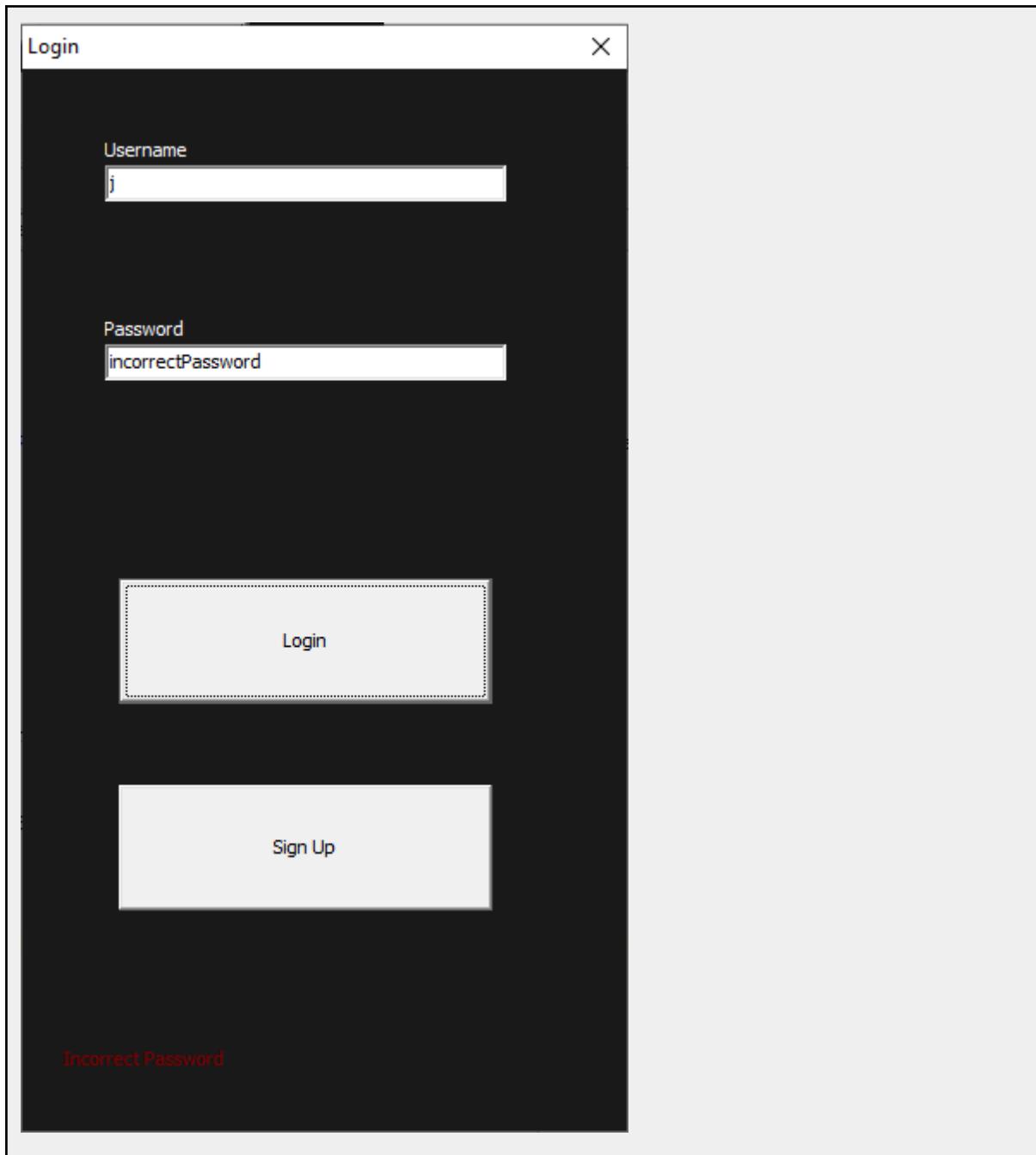
Password

[Sign Up](#)

Incorrect Username

**1.5 Result**



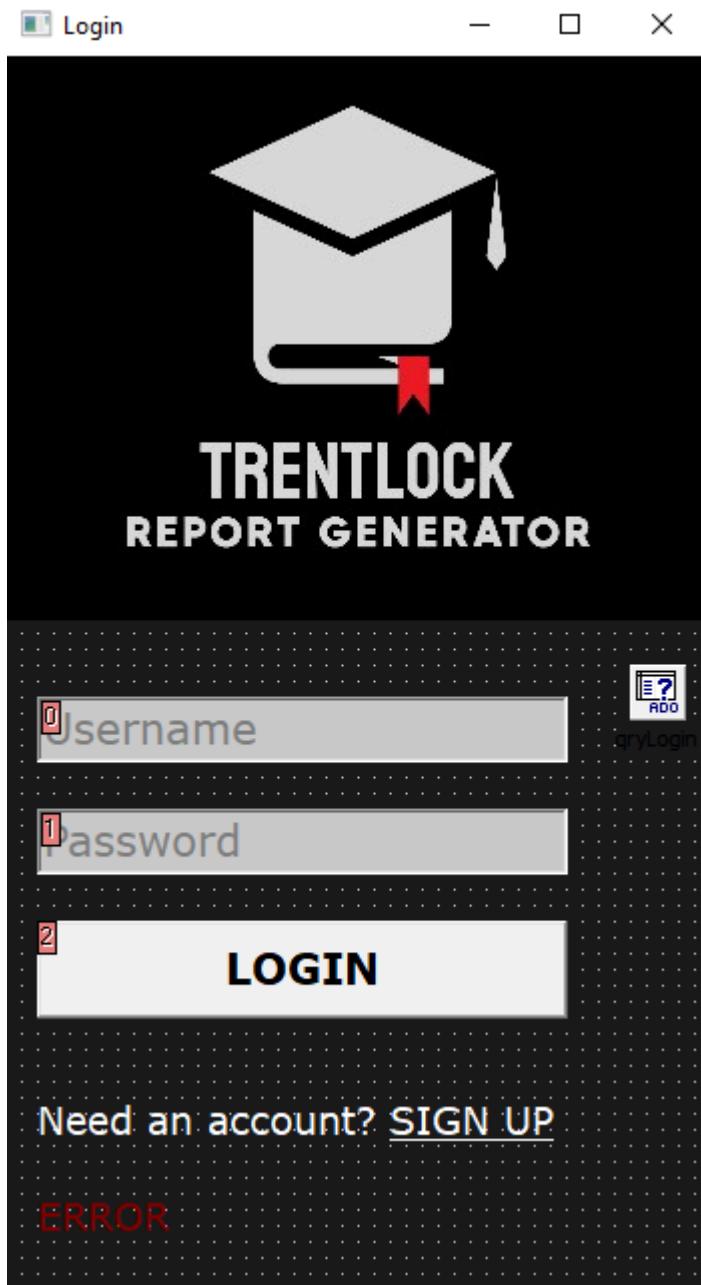


## Summary

The user now launches the program, the login page is displayed to them, they can input their username and password and the program will validate their login details, if they are correct the home page will open, if they are incorrect an error message appears. The user can also click the sign up page to create an account.

### 3.2.3 Prototype 3

#### Form screenshot



### Form accomplishments

The form has been substantially improved visually, text is now bigger and clearer using verdana font as per the requirements. The username and password boxes have been improved, they use regular edit boxes instead of labelled and the label exists within the edit box, this saves space and is easier to look at. The login button has improved dimensions and a bigger text using verdana and bold font. The school logo has been added along with a title, this brings the design closer to the interface design planned earlier on. There is now text instead of a sign up button, this is more appealing to the user and more clearer on what they can do.

**Code****unitLogin**

```
unit unitLogin; // testing annotation

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, jpeg, ExtCtrls, StdCtrls, DB, ADODB, unitHomePage, unitSignUp;

type
  TfrmLogin = class(TForm)

// Logo has been added and it is an image variable

  imgLogo: TImage;
  edtUsername: TEdit;
  edtPassword: TEdit;
  btnLogin: TButton;
  lblSignUp: TLabel;
  lblSignUpButton: TLabel;
  lblErrorMessage: TLabel;
  qryLogin: TADOQuery;

  procedure edtUsernameKeyPress(Sender: TObject; var Key: Char);
  procedure edtPasswordKeyPress(Sender: TObject; var Key: Char);
  procedure edtUsernameClick(Sender: TObject);
  procedure edtPasswordClick(Sender: TObject);
  procedure lblSignUpButtonClick(Sender: TObject);
  procedure lblSignUpButtonMouseEnter(Sender: TObject);
  procedure lblSignUpButtonMouseLeave(Sender: TObject);
  procedure btnLoginClick(Sender: TObject);
  procedure FormShow(Sender: TObject);

public
  StaffID : string;

private
  usernamePromptHidden : boolean;
  passwordPromptHidden : Boolean;

  procedure HideTextPrompt(textEdit : TEdit; var hidden : boolean);
```

```
procedure DisplayError(errorMessage : string);
procedure GetUsernamePassword;
procedure Login;
end;

var
frmLogin: TfrmLogin;

implementation

{$R *.dfm}

// This event is triggered when the form is shown, this allows us to set some default values or reset any values before the user sees anything. It sets the error message to not be visible and resets the username and password fields to their prompt text so the user can again see what the boxes are there for.

procedure TfrmLogin.FormShow(Sender: TObject);
begin
  lblErrorMessage.Visible := false;

  usernamePromptHidden := false;
  passwordPromptHidden := false;

  edtUsername.Text := 'Username';
  edtPassword.Text := 'Password';
  edtUsername.Font.Color := clGray;
  edtPassword.Font.Color := clGray;
  edtPassword.PasswordChar := #0;
end;

// If the user clicks or types into the username or password edit boxes, the prompt text must hide allowing them to enter their username and password, these procedures that are triggered when that happens calls a function to hide the text prompt

procedure TfrmLogin.edtUsernameKeyPress(Sender: TObject; var Key: Char);
begin
  HideTextPrompt(TEdit(Sender), usernamePromptHidden);
end;

procedure TfrmLogin.edtPasswordKeyPress(Sender: TObject; var Key: Char);
begin
  HideTextPrompt(TEdit(Sender), passwordPromptHidden);
  TEdit(Sender).PasswordChar := '*';

```

```
end;

procedure TfrmLogin.edtUsernameClick(Sender: TObject);
begin
  HideTextPrompt(TEdit(Sender), usernamePromptHidden);
end;

procedure TfrmLogin.edtPasswordClick(Sender: TObject);
begin
  HideTextPrompt(TEdit(Sender), passwordPromptHidden);
  TEdit(Sender).PasswordChar := '*';
end;

procedure TfrmLogin.lblSignUpButtonClick(Sender: TObject);
begin
  Hide;
  frmSignUp.Show;
end;

// The sign up text changes colour when the mouse hovers over, this is to indicate to the user the text is interactable and they can click it. On the mouse enter event the colour changes to red and then back to white on leave

procedure TfrmLogin.lblSignUpButtonMouseEnter(Sender: TObject);
begin
  TLabel(Sender).Font.Color := clMaroon;
end;

procedure TfrmLogin.lblSignUpButtonMouseLeave(Sender: TObject);
begin
  TLabel(Sender).Font.Color := clWhite;
end;

procedure TfrmLogin.btnLoginClick(Sender: TObject);
begin
  GetUsernamePassword;
end;

// This procedure takes in a text edit object and a boolean to determine if the text on the object has been hidden yet. It hides the text prompt by clearing the string and setting the colour back to black instead of the default grey as shown in the form screenshot

procedure TFrmLogin.HideTextPrompt(textEdit : TEdit; var hidden : Boolean);
begin
```

```
if not hidden then begin
    textEdit.Text := '';
    textEdit.Font.Color := clBlack;
    hidden := true;
end;
end;

procedure TfrmLogin.DisplayError(errorMessage : string);
begin
    lblErrorMessage.Caption := errorMessage;
    lblErrorMessage.Visible := true;
end;

procedure TfrmLogin.GetUsernamePassword;
var
    sqlStatement : string;
begin
    sqlStatement := 'SELECT * FROM Login WHERE Username = ' +
    QuotedStr(edtUsername.Text);

    with qryLogin do begin
        Close;
        SQL.Clear;
        SQL.Add(sqlStatement);
        Open;

        if RecordCount = 0 then
            DisplayError('Incorrect Username')
        else if FieldByName('password').AsString <> edtPassword.Text then
            DisplayError('Incorrect Password')
        else begin
            StaffID := FieldByName('staffID').AsString;
            Login;
        end;
    end;
end;

procedure TfrmLogin.Login;
begin
    Hide;
    frmHomePage.Show;
end;

end.
```

## Code accomplishments

The code has been updated to allow for new UI changes, by default delphi does not support many standard UI formats so these had to be coded in manually. Now the username and password text boxes have the label within the text box in a grey colour, this remains until the user clicks on it and it will hide, this hide effect is by resetting the string to empty and changing the colour back to black to give the illusion it is a text prompt that goes away when the user starts typing. This is what the HideTextPrompt() procedure is used for. There is also a form show event that is triggered when the form is shown, this allows for more useful changes like making the error message invisible initially, this improves the look of the form. Whenever the form is opened again the code will be run there. The key validation and login functionality remains the same since prototype 2.

## Requirements met

This meets requirements 8 and 12.

Number	Feature	Explanation and justification
8.	The login form must contain the school logo and name of the application.	This is a visual change that would improve the look of the program, by showing the school logo and name of the solution the user knows what they are about to use and it makes the program look more complete and trustworthy.
12.	Every title, button and edit box should be clearly labelled.	Once again the software should be easy to use, the user should understand what everything does right away, this requirement will help with this.

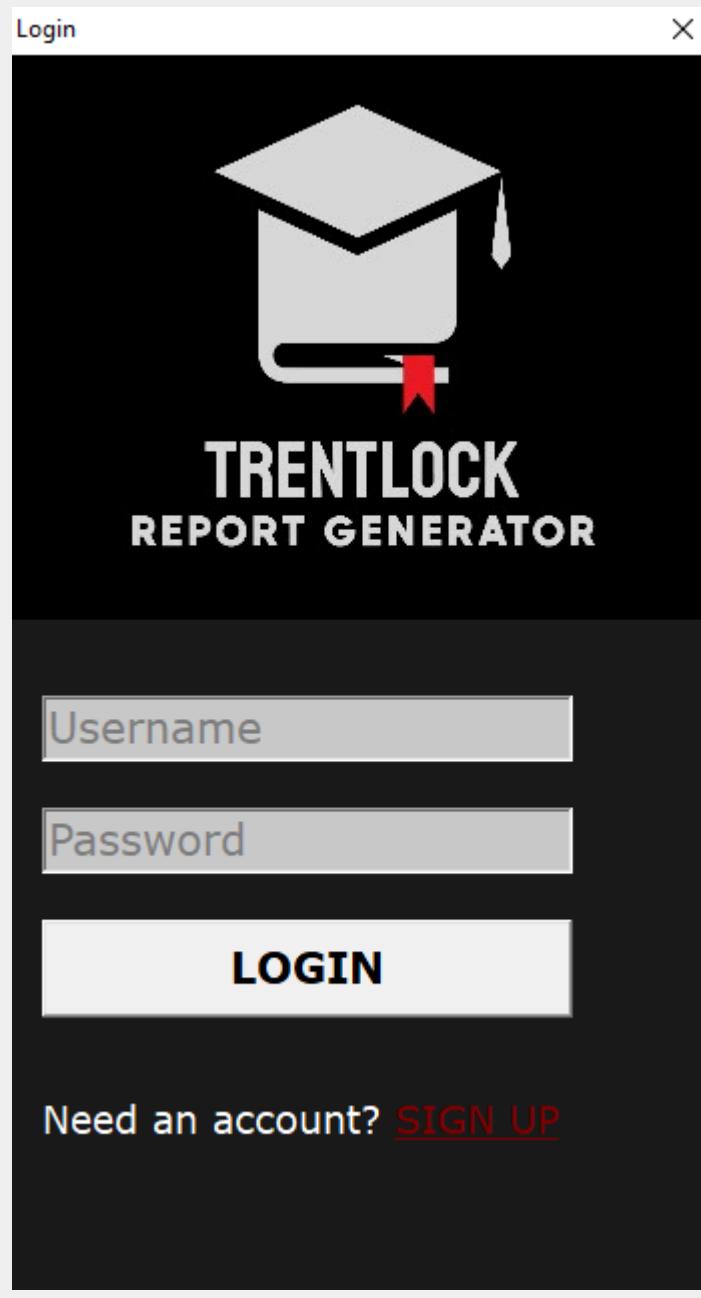
## Testing

Test number 1.1 will be carried out.

Test number	Test purpose	Test data	Expected result	Result
1.1	Check if the sign up text	Mouse enter	Sign up text changes	Success

	changes colour when the mouse hovers over	event triggering	to red colour.	
--	---	------------------	----------------	--

### 1.1 Result



### Summary

Same as prototype two, the user can launch the program, login or go to the sign up page. The experience has been improved a lot with the improved UI which makes it

clearer and easier to use. A large amount of code has been added to support the UI changes. This form is now to a high standard so this will be the last prototype and the final version of the login form.

## 3.3 Stage 3 sign up form

### 3.3.1 Prototype 1

#### Form screenshot

The screenshot shows a 'Sign Up' form window with a dotted background. The form contains five input fields labeled 0 through 4, each with a red square icon in the top-left corner. Below the inputs is a 'Sign Up' button with a red square icon. At the bottom is a 'Login' button with a red square icon. The window has standard minimize, maximize, and close buttons at the top right.

Field	Value
First Name	0
Last Name	1
Email	2
Username	4
Password	3

## Form accomplishments

The sign up form has been created. It contains 5 edit boxes for first name, last name, email, username and password. It also contains two buttons named login and sign that the user can click. The buttons have been placed below the edit boxes to indicate to the user that they must fill in the boxes before pressing sign up or login.

## Code

### unitSignUp

```
unit unitSignUp;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, StdCtrls, ExtCtrls, StrUtils, DB, ADODB;

type
  TfrmSignUp = class(TForm)

// Class variables have been defined here including the buttons added to the form
and the edit boxes

  btnLogin: TButton;
  btnSignUp: TButton;
  IbledtFirstName: TLabeledEdit;
  IbledtLastName: TLabeledEdit;
  IbledtEmail: TLabeledEdit;
  IbledtUsername: TLabeledEdit;
  IbledtPassword: TLabeledEdit;

// Two event procedures for when the user clicks login or sign up

  procedure btnLoginClick(Sender: TObject);
  procedure btnSignUpClick(Sender: TObject);

// Here are several procedures that have been defined but not implemented yet for
this prototype, these are for checking each field the user inputs that will go into
the database to ensure valid data
```

```
private
  canSignUp : Boolean;

procedure CheckFirstName;
procedure CheckLastName;
procedure CheckEmail;
procedure CheckUsername;
procedure CheckPassword;
procedure CheckSQL;

// This procedure takes in a string and displays an error message, however it has not been implemented yet
procedure DisplayError(errorMessage : string);
end;

var
  frmSignUp: TfrmSignUp;

implementation

uses unitLogin;

{$R *.dfm}

// Hides the current form and goes back to the login page

procedure TfrmSignUp.btnLoginClick(Sender: TObject);
begin
  Hide;
  frmLogin.Show;
end;

procedure TfrmSignUp.btnSignUpClick(Sender: TObject);
begin
  canSignUp := true;

// When the user clicks sign up, it runs the Check first name function, this will then validate the value in the first name edit box, if it is invalid it will set the canSignUp to false, initially it is set to true because the validation has not been implemented yet

  CheckFirstName;

// After validation, if canSignUp variable is still true the user can login, so the
```

*CheckSQL function is run, which will save the details to the database, for now no implementation of that has been added*

```
if canSignUp then begin
```

```
    if MessageDlg('Confirm Sign Up - Are you sure these details are correct?',
```

*mtConfirmation, [mbYes, mbNo], 0) = mrYes then begin // Shows confirmation box with options of yes or no*

```
        CheckSQL;
```

```
        end;
```

```
    end;
```

```
end;
```

*// Procedures that will be added in prototype 2 for validating login details, each field will have its own procedure to validate that field*

```
procedure TfrmSignUp.CheckFirstName;
```

```
end;
```

```
procedure TfrmSignUp.CheckLastName;
```

```
end;
```

```
procedure TfrmSignUp.CheckEmail;
```

```
end;
```

```
procedure TfrmSignUp.CheckUsername;
```

```
end;
```

```
procedure TfrmSignUp.CheckPassword;
```

```
end;
```

```
procedure TfrmSignUp.DisplayError(errorMessage : string);
```

```
begin
```

```
end;
```

```
procedure TfrmSignUp.CheckSQL;
```

```
end;
```

```
end.
```

## Requirements met

This has met requirements 6, 16, and 17

Number	Feature	Explanation and justification
--------	---------	-------------------------------

6.	There must be a login and sign up form with a small vertical window size.	The user should be greeted with a login page and have the option to sign up which will take them to a sign up page. These forms should be smaller than the main forms as they don't need to be large since they aren't displaying lots of information. It is also indicating to the user they have not entered the main program yet.
16.	Program must allow users to sign up with a sign up page	If a staff member does not have an account but wishes to use the solution they need to be able to sign up.
17.	Program should require the user to enter a username, password, full name and email address before signing up.	The user needs to enter their details so they can be saved in the database and displayed. If they enter their username and password they won't have a default username or password which could be unsafe.

## Testing

No testing is required for this prototype.

### 3.3.2 Prototype 2

#### Form screenshot

The screenshot shows a Windows application window titled "Sign Up". The window has a dark grey header bar with the title and standard window controls (minimize, maximize, close). The main area contains a registration form with the following fields:

- First Name: A text input field containing "0".
- Last Name: A text input field containing "1".
- Email: A text input field containing "2".
- Username: A text input field containing "4".
- Password: A text input field containing "3".
- A checkbox labeled "7" followed by the text "I confirm I work at Trentlock".

Below the form are two large rectangular buttons:

- A white button labeled "5" containing the text "Sign Up".
- A white button labeled "6" containing the text "Login".

At the bottom left, there is a small red label "lblErrorMessage".

## Form accomplishments

The second prototype has been made, this includes some visual improvements to the form design. The background has been changed to match the requirements and improve the look. A confirmation box has been added, this requires the user to click the box before signing up to ensure they work at the school. An error message has been added to the bottom right of the form, this will be enabled when there is an error so the user can see the error message.

## Code

**unitSignUp**

unit unitSignUp;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, Buttons, StdCtrls, ExtCtrls, StrUtils, DB, ADODB;

type

TfrmSignUp = class(TForm)

  btnLogin: TButton;  
  btnSignUp: TButton;  
  IbledtFirstName: TLabeledEdit;  
  IbledtLastName: TLabeledEdit;  
  IbledtEmail: TLabeledEdit;  
  IbledtUsername: TLabeledEdit;  
  IbledtPassword: TLabeledEdit;  
  chkConfirm: TCheckBox;  
  lblErrorMessage: TLabel;

*// Check box and error message variables added*

  procedure FormShow(Sender: TObject);  
  procedure btnLoginClick(Sender: TObject);  
  procedure btnSignUpClick(Sender: TObject);  
  procedure IbledtAllClick(Sender: TObject);

private

  canSignUp : Boolean;

  procedure CheckFirstName;  
  procedure CheckLastName;  
  procedure CheckEmail;  
  procedure CheckUsername;  
  procedure CheckPassword;  
  procedure CheckSQL;

  procedure DisplayError(errorMessage : string);  
end;

var

  frmSignUp: TfrmSignUp;

```
implementation

uses unitLogin;

{$R *.dfm}

// When the form is show it sets the error message label it invisible so the user cannot see it

procedure TfrmSignUp.FormShow(Sender: TObject);
begin
  IblErrorMessage.Visible := false;
end;

// This event procedure resets the colour of the labelled edit box when clicking on it, when there is an error the text in the edit box highlights (code shown below). This allows the user to see which field they got incorrect. Once they click on the edit box the test will go to black

procedure TfrmSignUp.IbledtAllClick(Sender: TObject);
begin
  TLabeledEdit(Sender).Font.Color := clBlack;
end;

procedure TfrmSignUp.btnLogin(Sender: TObject);
begin
  Hide;
  frmLogin.Show;
end;

procedure TfrmSignUp.btnSignUpClick(Sender: TObject);
begin
  canSignUp := true;

  CheckFirstName;

  if canSignUp then begin
    if chkConfirm.Checked then begin
      if MessageDlg('Confirm Sign Up - Are you sure these details are correct?', mtConfirmation, [mbYes, mbNo], 0) = mrYes then begin
        CheckSQL;
      end;
    end
    else begin
  
```

```

        DisplayError('Fill in all fields'); // Using display error function that has now been
        implemented, it passes this string as an argument, so the error label will show this
        message
    end;
end;

end;

// Checking the first name validation has now been implemented

procedure TfrmSignUp.CheckFirstName;
const
    letters = ['A'..'Z', 'a'..'z']; // pascal set containing all letters from A to Z in
lowercase and uppercase
var
    i : integer;
    errorMessage : string; // Uses a variable so the error message can change
depending on the error
begin
    if Length(lbledtFirstName.Text) = 0 then begin // If user has left the text box blank
show an error and canSignUp should be set to false
        canSignUp := false;
        errorMessage := 'Enter a first name';
    end;

    // Loops through first name edit box, for each character at index i, if the character
is not in the set then they can't sign up because the character is not a letter in the
alphabet
    for i := 1 to Length(lbledtFirstName.Text) do begin
        if not (lbledtFirstName.Text[i] in letters) then begin
            canSignUp := false;
            errorMessage := 'First name must only contain letters';
        end;
    end;

    // If the canSignUp variable is still true then the first name is valid, so it moves on
to the next procedure which is login
    if canSignUp then
        CheckLastName
    else begin // Otherwise display the error message for the first name being wrong
and set the text colour to red so the user can see it visually change
        DisplayError(errorMessage);
        lbledtFirstName.Font.Color := clRed;
    end;
end;

```

// Checking last name validation has now been implemented, logic is the same as first name, it checks if the edit box is empty and then if the edit box does not contain any letters

```
procedure TfrmSignUp.CheckLastName;
const
  letters = ['A'..'Z', 'a'..'z'];
var
  i : integer;
  errorMessage : string;
begin
  if Length(lbledtLastName.Text) = 0 then begin
    canSignUp := false;
    errorMessage := 'Enter a last name';
  end;

  for i := 1 to Length(lbledtLastName.Text) do begin
    if not (lbledtLastName.Text[i] in letters) then begin
      canSignUp := false;
      errorMessage := 'Last name must only contain letters';
    end;
  end;
end;
```

// If they can sign up still, move on to checking email, otherwise display error procedure is called

```
if canSignUp then
  CheckEmail
else begin
  DisplayError(errorMessage);
  lbledtLastName.Font.Color := clRed;
end;
end;
```

// Checking email has now been implemented

```
procedure TfrmSignUp.CheckEmail;
const
  illegalCharacters = ['!', '#', '$', '&', '*', '+', '/', '=', '?', '^', '_', '^', '{', '|', '}', '~'];
  maxEmailLength = 320;
```

// Two variables defined as constants because they won't change, an array of illegal characters that cannot be used and the max length of an email address which is used online

```
var
```

```
i, j: integer;
containsCommercialAt : boolean;
indexOfCommercialAt : integer;
dotAfterCommercialAt : boolean;
begin
  for i := 1 to Length(lbledtEmail.Text) do begin // If the email contains an @ symbol
    then set the index of the @ symbol and set boolean to true
    if lbledtEmail.Text[i] = '@' then begin
      indexOfCommercialAt := i;
      containsCommercialAt := true;
    end
  end;

// If there is no @ symbol can sign up is false

if not containsCommercialAt then begin
  canSignUp := false;
end;

// If the position of the @ symbol in the string is at the start of end, then this is
invalid and can sign up is set to false

if canSignUp then begin
  if (indexOfCommercialAt = 1) or (indexOfCommercialAt =
Length(lbledtEmail.Text)) then begin
    canSignUp := false;
  end
  else begin // Otherwise loop through array from the @ symbol to the rest, to
check if there is a dot after the @ symbol
    for j := indexOfCommercialAt + 1 to Length(lbledtEmail.Text) do begin
      if lbledtEmail.Text[j] = '.' then begin
        dotAfterCommercialAt := true;
      end;
    end;

// Whilst looping through from the @ symbol to the end it can now check for the
illegal characters that cannot be in the domain name section

    if lbledtEmail.Text[j] in illegalCharacters then begin
      canSignUp := false;
    end;
  end;
end;

// If the first character is a dot they cant sign up
if lbledtEmail.Text[1] = '.' then begin
```

```
canSignUp := false;
end;

// If there are two dots next to each other they cannot sign up

if AnsiContainsText(lbledtEmail.Text, '..') then begin
  canSignUp := false;
end;
end;

// Now that is has checked if there is a dot after the sign up it can set sign up to false if there isn't
if not dotAfterCommercialAt then begin
  canSignUp := false;
end
else begin // If there is a dot right after the @, or at the end of the email this is invalid
  if (lbledtEmail.Text[indexOfCommercialAt + 1] = '.') or
  (lbledtEmail.Text[Length(lbledtEmail.Text)] = '.') then begin
    canSignUp := false;
  end;

// If there is a dash after the @ its invalid
if (lbledtEmail.Text[indexOfCommercialAt + 1] = '-') then begin
  canSignUp := false;
end;
end;

// Checks length
if Length(lbledtEmail.Text) > maxEmailLength then begin
  canSignUp := false;
end;

// If can sign up is still true, then check the username field, otherwise display error message

if canSignUp then begin
  CheckUsername;
end
else begin
  DisplayError('Invalid email format');
  lbledtEmail.Font.Color := clRed;
end;
end;
```

```
// Check username procedure has now been implemented
procedure TfrmSignUp.CheckUsername;
const
  illegalCharacters = ['&', '=', '_', '^', '+', ',', '<', '>', '''];
// Array of characters that cannot be used
var
  errorMessage : string;
  i : integer;
begin
  // If length is less than 6 or more than 30 they cannot sign up
  if (Length(lbledtUsername.Text) < 6) or (Length(lbledtUsername.Text) > 30) then
  begin
    canSignUp := false;
    errorMessage := 'Username between 6 and 30 characters';
  end;

  // Loops through username edit, for each character if it is in the illegal characters
  // then they can't sign up
  for i := 1 to Length(lbledtUsername.Text) do begin
    if lbledtUsername.Text[i] in illegalCharacters then begin
      canSignUp := false;
      errorMessage := 'Username has illegal characters';
    end;
  end;

  // If can sign up is still true program runs check password procedure, otherwise
  // display error
  if canSignUp then begin
    CheckPassword;
  end
  else begin
    DisplayError(errorMessage);
    lbledtUsername.Font.Color := clRed;
  end;
end;

// Check password procedure has now been implemented
procedure TfrmSignUp.CheckPassword;
const
  requiredCharacters = ['!', '#', '$', '%', '&', '(', ')', '*', '+', ',', '^', '-', '.', '/', ';', '>', '<', '=',
  '@', '[', ']', '\', '^', '_']; // Array of required characters to make password secure
var
  i : integer;
  hasRequiredCharacter : boolean;
  errorMessage : string;
```

```
begin
  if Length(lbledtPassword.Text) < 8 then begin // Password must be at least 8
    characters
      canSignUp := false;
      errorMessage := 'Password must be at least 8 characters';
      end;

  // Loops through each character, if it is in the required characters set this boolean
  // to true
  for i := 1 to Length(lbledtPassword.Text) do begin
    if lbledtPassword.Text[i] in requiredCharacters then begin
      hasRequiredCharacter := true;
      end;
    end;

  // If there are no required characters, they can't sign up and appropriate error
  // message is set
  if not hasRequiredCharacter then begin
    canSignUp := false;
    errorMessage := 'Password must contain a special character';
    end;

  if not canSignUp then begin
    DisplayError(errorMessage);
    lbledtPassword.Font.Color := clRed;
    end;
  end;

// Display error procedure has now been implemented, it sets the label it visible
// and sets its caption to the error message parameter
procedure TfrmSignUp.DisplayError(errorMessage : string);
begin
  lblErrorMessage.Caption := errorMessage;
  lblErrorMessage.Visible := true;
end;

// This procedure has not been implemented and validating the details with the
// database will be complete in the 3rd prototype
procedure TfrmSignUp.CheckSQL;
end;

end.
```

## Requirements met

This has met requirements 11, 18, 19, 20 and 21.

Number	Feature	Explanation and justification
11.	Each form should have a dark blue background with contrasting white text.	The software should be easy to look at and have a clear layout, by having a dark blue background it creates a good look without being distracting. During the interview with Mr. Nicolston this was outlined as a key requirement.
18.	Program must display an error message if a text box is left empty or the value entered is invalid.	There will be a large amount of validation in the sign up page, if a user has inputted invalid data, they must know which field was invalid instead of having to guess.
19.	The password entered must be at least 8 characters and require a special character.	To ensure security the program will validate the password making sure it's long enough and contains a special character. This means it's harder to hack the user's account.
20.	The username must be between 6 and 30 characters and contain no illegal characters.	The username must also follow a standard format, there will be a list of illegal characters it cannot have and it must have a realistic feasible length.
21.	The email address should be the correct format.	The email address must be the correct format otherwise a user can put anything they want and this will cause errors down the line. If an email system is implemented the email must be valid.

## Testing

Test number	Test purpose	Test data	Expected result	Result
2.1	Check if invalid first name	first name:	'First name must only	Success

	displays an error message	'jason123!'	contain letters' error message appears.	
2.2	Check if invalid last name displays an error message	last name: '#@%jackson_'	'Last name must only contain letters' error message appears.	Success
2.3	Check if empty first name displays an error	first name: ''	'Enter a first name' error message appears.	Success
2.4	Check if empty last name displays an error	last name: ''	'Enter a last name' error message appears.	Success
2.5	Check if email in the incorrect format displays an error	email: '.jason@email'	'Invalid email format' error message appears.	Success
2.6	Check if username is more than 30 characters	username: 'thislongusernameiswaytoolonganobodywoulduseit'	'Username between 6 and 30 characters' error message appears.	Success
2.7	Check if username is exactly 6 characters	username: 'Daniel'	N/A	Success
2.8	Check if username is exactly 30 characters	'DanielScottAtTrentlockSchool1'	N/A	Success
2.9	Check if username has illegal characters	'<Daniel>&&&'	'Username has illegal characters' error message appears.	Success
2.10	Check if password has less than 8 characters	'123'	'Password must be at least 8 characters' error message appears.	Fail
2.11	Check if password has exactly 8 characters	'password!'	N/A	Success
2.12	Check if password has more than 8 characters	'thelonelysoldier123\$'	N/A	Success
2.13	Check if password does not contain a special character	'thelonelysoldier123'	'Password must contain a special character' error message appears.	Success
2.14	Check if clicking the sign up button shows confirmation message	Sign up button click	Confirmation box should open.	Success

## 2.1 Result

Sign Up X

First Name

Last Name

Email

Username

Password

I confirm I work at Trentlock

First name must only contain letters

## 2.2 Result

Sign Up X

First Name

Last Name

Email

Username

Password

I confirm I work at Trentlock

Sign Up

Login

Last name must only contain letters

### 2.3 Result

Sign Up X

First Name

Last Name

Email

Username

Password

I confirm I work at Trentlock

Enter a first name

**2.4 Result**

Sign Up X

First Name

Last Name

Email

Username

Password

I confirm I work at Trentlock

Sign Up

Login

Enter a last name

**2.5 Results**

Sign Up X

First Name

Last Name

Email

Username

Password

I confirm I work at Trentlock

[Sign Up](#)

[Login](#)

Invalid email format

**2.6 Results**

Sign Up X

First Name

Last Name

Email

Username

Password

I confirm I work at Trentlock

Username between 6 and 30 characters

**2.7 Results**

Sign Up X

First Name

Last Name

Email

Username

Password

I confirm I work at Trentlock

lblErrorMessage

**2.8 Results**

Sign Up X

First Name

Last Name

Email

Username

Password

I confirm I work at Trentlock

lblErrorMessage

**2.9 Results**

Sign Up X

First Name

Last Name

Email

Username

Password

I confirm I work at Trentlock

Username has illegal characters

## 2.10 Previous Code

```
if Length(lbledtPassword.Text) > 8 then begin
    canSignUp := false;
    errorMessage := 'Password must be at least 8 characters';
end;
```

## 2.10 Amended Code

```
if Length(lbledtPassword.Text) < 8 then begin
    canSignUp := false;
    errorMessage := 'Password must be at least 8 characters';
```

end

## 2.10 Results

Sign Up X

First Name

Last Name

Email

Username

Password

I confirm I work at Trentlock

Sign Up

Login

Password must be at least 8 characters

## 2.11 Results

Sign Up X

First Name

Last Name

Email

Username

Password

I confirm I work at Trentlock

lblErrorMessage

**2.12 Results**

Sign Up X

First Name

Last Name

Email

Username

Password

I confirm I work at Trentlock

lblErrorMessaage

**2.13 Results**

Sign Up X

First Name

Last Name

Email

Username

Password

I confirm I work at Trentlock

Password must contain a special character

Forgot password? | Create account

**2.14 Results**

The screenshot displays a user interface for sign-up and confirmation:

- Sign Up:** A modal window titled "Sign Up" containing fields for First Name (test), Last Name (test), Email (test@test.com), and Username (testing).
- Confirm:** A modal window titled "Confirm" asking "Confirm Sign Up - Are you sure these details are correct?" with "Yes" and "No" buttons.
- Feedback:** A message at the bottom stating "Password must contain a special character" in red text.

### 3.3.3 Prototype 3

#### Form screenshot

The screenshot shows a Windows application window titled "Sign Up". The form contains five input fields labeled "First Name", "Last Name", "Email", "Username", and "Password", each with a red number placeholder (1 through 5) in the top-left corner. To the right of each input field is a small blue icon with a question mark and the text "ADO". Below the input fields is a checkbox labeled "I CONFIRM I WORK AT TRENTLOCK". At the bottom of the form is a large, bold "SIGN UP" button with a red number placeholder "0" in its top-left corner. The background of the form is dark grey, and the overall design is clean and modern.

## Form accomplishments

The third prototype has now been completed, this contains only visual improvements and no functionality improvements. The edit boxes have been resized for a more readable look. The fonts have been changed and resized to use verdana font. The login button has been replaced with a text and an underlined text button to visually indicate to the user they can click it. The adoQuery has also been added to the form, this object in the form will be used in code to link to the database.

## Code

**unitSignUp**

```
unit unitSignUp;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, StdCtrls, ExtCtrls, StrUtils, DB, ADODB;

type
  TfrmSignUp = class(TForm)

    lblLoginButton: TLabel;
    btnSignUp: TButton;
    lbledtFirstName: TLabeledEdit;
    lblLogin: TLabel;
    lbledtLastName: TLabeledEdit;
    lbledtEmail: TLabeledEdit;
    lbledtUsername: TLabeledEdit;
    lbledtPassword: TLabeledEdit;
    chkConfirm: TCheckBox;
    lblErrorMessage: TLabel;
    qrySignUp: TADOQuery; // Query variable added

    procedure FormShow(Sender: TObject);
    procedure lblLoginButtonMouseEnter(Sender: TObject);
    procedure lblLoginButtonMouseLeave(Sender: TObject);
    procedure lblLoginButtonClick(Sender: TObject);
    procedure btnSignUpClick(Sender: TObject);
    procedure lbledtAllClick(Sender: TObject);

private
  canSignUp : Boolean;

  procedure CheckFirstName;
  procedure CheckLastName;
  procedure CheckEmail;
  procedure CheckUsername;
  procedure CheckPassword;
  procedure CheckSQL;

  procedure DisplayError(errorMessage : string);
end;
```

```
var
  frmSignUp: TfrmSignUp;

implementation

uses unitLogin;

{$R *.dfm}

procedure TfrmSignUp.FormShow(Sender: TObject);
begin
  IblErrorMessage.Visible := false;
end;

// When hovering over the login label, it changes the colour of the label and when leaving it changes it back to white

procedure TfrmSignUp.IblLoginButtonMouseEnter(Sender: TObject);
begin
  TLabel(Sender).Font.Color := clMaroon;
end;

procedure TfrmSignUp.IblLoginButtonMouseLeave(Sender: TObject);
begin
  TLabel(Sender).Font.Color := clWhite;
end;

procedure TfrmSignUp.IbledtAllClick(Sender: TObject);
begin
  TLabel(Sender).Font.Color := clBlack;
end;

procedure TfrmSignUp.IblLoginButtonClick(Sender: TObject);
begin
  Hide;
  frmLogin.Show;
end;

procedure TfrmSignUp.btnSignUpClick(Sender: TObject);
begin
  canSignUp := true;

  CheckFirstName;
```

```
if canSignUp then begin
    if chkConfirm.Checked then begin
        if MessageDlg('Confirm Sign Up - Are you sure these details are correct?', mtConfirmation, [mbYes, mbNo], 0) = mrYes then begin
            CheckSQL;
        end;
    end
    else begin
        DisplayError('Fill in all fields');
    end;
end;

procedure TfrmSignUp.CheckFirstName;
const
    letters = ['A'..'Z', 'a'..'z'];
var
    i : integer;
    errorMessage : string;
begin
    if Length(lbledtFirstName.Text) = 0 then begin
        canSignUp := false;
        errorMessage := 'Enter a first name';
    end;

    for i := 1 to Length(lbledtFirstName.Text) do begin
        if not (lbledtFirstName.Text[i] in letters) then begin
            canSignUp := false;
            errorMessage := 'First name must only contain letters';
        end;
    end;

    if canSignUp then
        CheckLastName
    else begin
        DisplayError(errorMessage);
        lbledtFirstName.Font.Color := clRed;
    end;
end;

procedure TfrmSignUp.CheckLastName;
const
    letters = ['A'..'Z', 'a'..'z'];
```

```
var
  i : integer;
  errorMessage : string;
begin
  if Length(lbledtLastName.Text) = 0 then begin
    canSignUp := false;
    errorMessage := 'Enter a last name';
  end;

  for i := 1 to Length(lbledtLastName.Text) do begin
    if not (lbledtLastName.Text[i] in letters) then begin
      canSignUp := false;
      errorMessage := 'Last name must only contain letters';
    end;
  end;

  if canSignUp then
    CheckEmail
  else begin
    DisplayError(errorMessage);
    lbledtLastName.Font.Color := clRed;
  end;
end;

procedure TfrmSignUp.CheckEmail;
const
  illegalCharacters = ['!', '#', '$', '&', '*', '+', '/', '=', '?', '^', '_', '(', ')', '{', '|', '}', '~'];
  maxEmailLength = 320;
var
  i, j: integer;
  containsCommercialAt : boolean;
  indexOfCommercialAt : integer;
  dotAfterCommercialAt : boolean;
begin
  for i := 1 to Length(lbledtEmail.Text) do begin
    if lbledtEmail.Text[i] = '@' then begin
      indexOfCommercialAt := i;
      containsCommercialAt := true;
    end;
  end;

  if not containsCommercialAt then begin
    canSignUp := false;
  end;
end;
```

```
if canSignUp then begin
    if (indexOfCommercialAt = 1) or (indexOfCommercialAt =
Length(lbledtEmail.Text)) then begin
        canSignUp := false;
    end
    else begin
        for j := indexOfCommercialAt + 1 to Length(lbledtEmail.Text) do begin
            if lbledtEmail.Text[j] = '.' then begin
                dotAfterCommercialAt := true;
            end;

            if lbledtEmail.Text[j] in illegalCharacters then begin
                canSignUp := false;
            end;
        end;
    end;

    if lbledtEmail.Text[1] = '.' then begin
        canSignUp := false;
    end;

    if AnsiContainsText(lbledtEmail.Text, '..') then begin
        canSignUp := false;
    end;
end;

if not dotAfterCommercialAt then begin
    canSignUp := false;
end
else begin
    if (lbledtEmail.Text[indexOfCommercialAt + 1] = '.') or
(lbledtEmail.Text[Length(lbledtEmail.Text)] = '.') then begin
        canSignUp := false;
    end;

    if (lbledtEmail.Text[indexOfCommercialAt + 1] = '-') then begin
        canSignUp := false;
    end;
end;

if Length(lbledtEmail.Text) > maxEmailLength then begin
    canSignUp := false;
end;

if canSignUp then begin
```

```

    CheckUsername;
end
else begin
  DisplayError('Invalid email format');
  lbledtEmail.Font.Color := clRed;
end;
end;

procedure TfrmSignUp.CheckUsername;
const
  illegalCharacters = ['&', '=', '_', '^', '+', ',', '<', '>', '''];
var
  errorMessage : string;
  i : integer;
begin
  if (Length(lbledtUsername.Text) < 6) or (Length(lbledtUsername.Text) > 30) then
begin
  canSignUp := false;
  errorMessage := 'Username between 6 and 30 characters';
end;

for i := 1 to Length(lbledtUsername.Text) do begin
  if lbledtUsername.Text[i] in illegalCharacters then begin
    canSignUp := false;
    errorMessage := 'Username has illegal characters';
  end;
end;

if canSignUp then begin
  CheckPassword;
end
else begin
  DisplayError(errorMessage);
  lbledtUsername.Font.Color := clRed;
end;
end;

procedure TfrmSignUp.CheckPassword;
const
  requiredCharacters = ['!', '#', '$', '%', '&', '(', ')', '*', '+', ',', '^', '-', '.', '/', ':', ';', '>', '<', '=', '@', '[', ']', '\', '^', '_'];
var
  i : integer;
  hasRequiredCharacter : boolean;
  errorMessage : string;

```

```
begin
if Length(lbledtPassword.Text) < 8 then begin
    canSignUp := false;
    errorMessage := 'Password must be at least 8 characters';
end;

for i := 1 to Length(lbledtPassword.Text) do begin
    if lbledtPassword.Text[i] in requiredCharacters then begin
        hasRequiredCharacter := true;
    end;
end;

if not hasRequiredCharacter then begin
    canSignUp := false;
    errorMessage := 'Password must contain a special character';
end;

if not canSignUp then begin
    DisplayError(errorMessage);
    lbledtPassword.Font.Color := clRed;
end;
end;

procedure TfrmSignUp.DisplayError(errorMessage : string);
begin
    lblErrorMessage.Caption := errorMessage;
    lblErrorMessage.Visible := true;
end;

// Check SQL procedure has now been added, this inserts the new values into the database

procedure TfrmSignUp.CheckSQL;
var
    sqlStatement : string;
    staffID : string;
begin
// string variable storing the sql statement, this is then added to the query
    sqlStatement := 'SELECT * FROM Login WHERE Username = ' +
    QuotedStr(lbledtUsername.Text);

// Closes and clears query to ensure it's empty before using
with qrySignUp do begin
    Close;
    SQL.Clear;
```

```
SQL.Add(sqlStatement);
Open;

// If there are no fields with the username that was inputted by the user it is safe to
// insert the new values, otherwise display an error saying the username already
// exists

if RecordCount = 0 then begin
    sqlStatement := 'INSERT INTO Staff (firstName, lastName, email) VALUES (' +
    QuotedStr(lbledtFirstName.Text) + ', ' + QuotedStr(lbledtLastName.Text) + ', ' +
    QuotedStr(lbledtEmail.Text) + ')';

// Changes the sqlStatement variable for inserting the values into the database
// quoted str is used because you can't use quotation marks within a string so this
// function does this for you

with qrySignUp do begin
    Close;
    SQL.Clear;
    SQL.Add(sqlStatement);
    ExecSQL; // Sql is executed
end;

// Now that data has been inserted into the staff table it gets the new staffID that
// was automatically numbered, it then inserts into the login table the username,
// password and staffID

sqlStatement := 'SELECT * FROM Staff WHERE firstName = ' +
QuotedStr(lbledtFirstName.Text);

with qrySignUp do begin
    Close;
    SQL.Clear;
    SQL.Add(sqlStatement);
    Open;

    staffID := FieldByName('staffID').AsString;
end;

sqlStatement := 'INSERT INTO Login VALUES (' +
QuotedStr(lbledtUsername.Text) + ', ' + QuotedStr(lbledtPassword.Text) + ', ' +
staffID + ')';

with qrySignUp do begin
    Close;
    SQL.Clear;
    SQL.Add(sqlStatement);
    ExecSQL; // Execute sql for inserting data into login table
```

```

end;
// Once data has been saved to database, uses show message function to tell the user it has been successful
ShowMessage('Congratulations, you''re account has been created');
Hide;
frmLogin.Show;
end
else begin
  DisplayError('Username already exists');
end;
end;
end;

end.

```

## Requirements met

This has met requirements 12, 13, 22 and 23.

Number	Feature	Explanation and justification
12.	Every title, button and edit box should be clearly labelled.	Once again the software should be easy to use, the user should understand what everything does right away, this requirement will help with this.
13.	Verdana font should be used throughout each form. Verdana size 16 bold should be used for each button and size 16 regular for each edit box. Each title should be size 48 bold.	The verdana font is a clean professional look, it will improve the UI significantly if this font is used throughout the program with consistent sizing and bold or regular formats.
22.	Program must tell the user if the username already exists.	The login page will require a username, this means there cannot be duplicate usernames. Once validation is complete the program must display an error message saying the username already exists.
23.	Program must save the sign up details to the database once	If the sign is successful, the details they entered should be saved into

	sign up is successful and return the user to the login page.	the database and they should be greeted with the login page again so they can quickly log in right away.
--	--	--

## Testing

Test number	Test purpose	Test data	Expected result	Result
2.15	Check if program displays error if username already exists	'qwerty'	'Username already exists' error message appears.	Success
2.16	Check if clicking yes in the confirmation box returns to login page	'Yes' button click	Sign up page closes, login page opens.	Success

### 2.15 Result

Sign Up X

*First Name*  
testing

*Last Name*  
testing

*Email*  
test@test.com

*Username*  
qwerty

*Password*  
\*\*\*\*\*

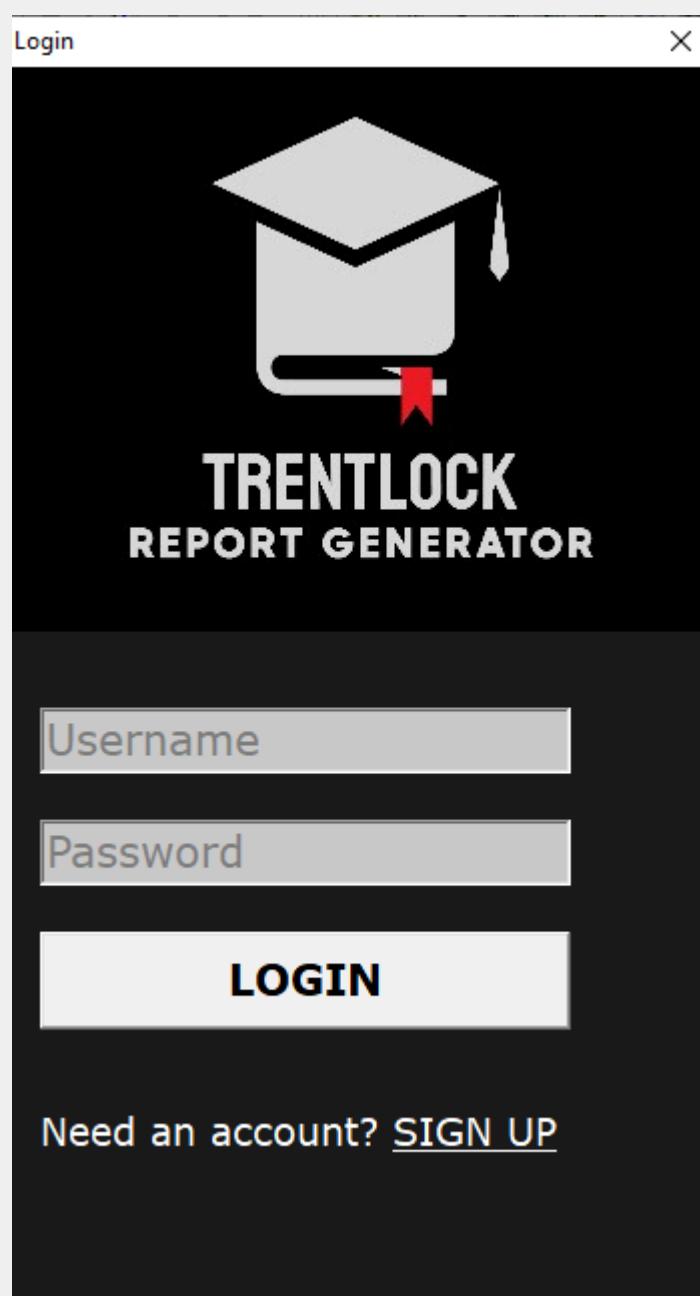
I CONFIRM I WORK AT TRENTLOCK

**SIGN UP**

Have an account? [LOGIN](#)

Username already exists

**2.16 Result**

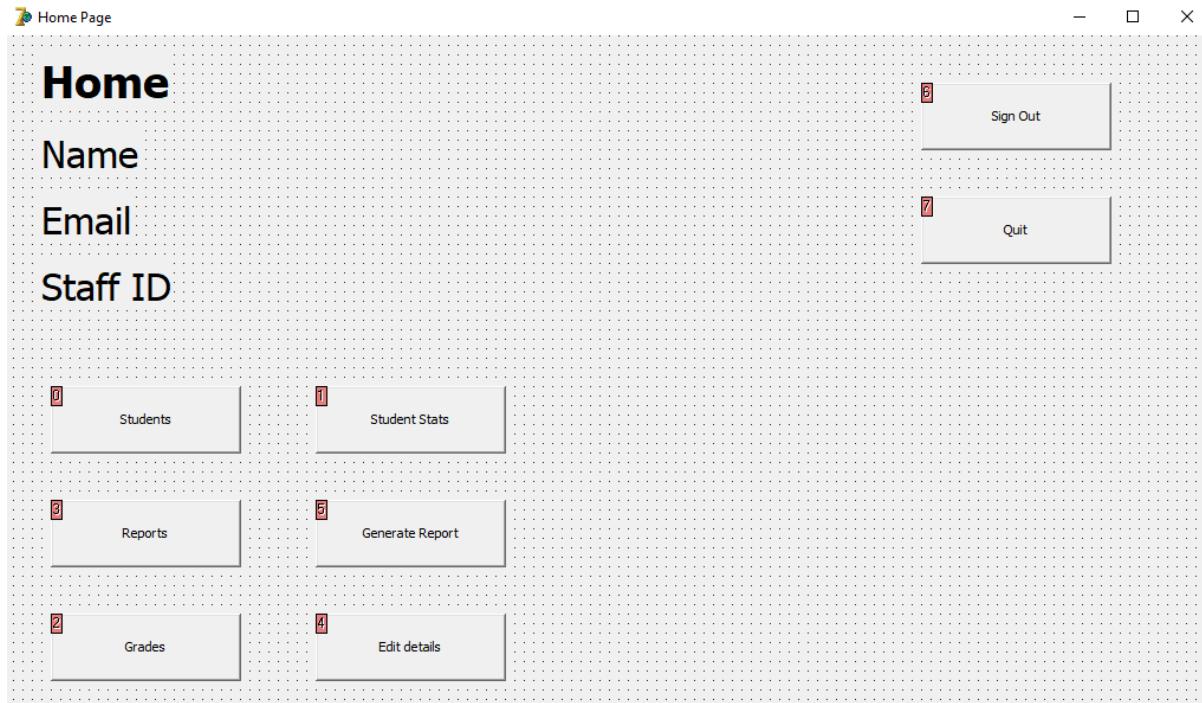


The image shows a login interface for the 'TRENTLOCK REPORT GENERATOR'. At the top left is a 'Login' button and a close ('X') button at the top right. Below the title is a logo featuring a graduation cap resting on an open book with a red ribbon. The title 'TRENTLOCK' is in large, bold, white capital letters, with 'REPORT GENERATOR' in a slightly smaller white font below it. There are two input fields: a light grey 'Username' field and a light grey 'Password' field. Below these is a large white button with the word 'LOGIN' in bold, dark blue capital letters. At the bottom left, there is a link 'Need an account? [SIGN UP](#)'.

## 3.4 Stage 4 home page form

### 3.4.1 Prototype 1

Form screenshot



## Form accomplishments

The first prototype for the home page has been completed, it contains a large title at the top left indicating the purpose of the page. It then contains 3 labels with name, email and staff ID, these labels will be renamed through the code to display the details of the staff member who logged in. Below these are 6 buttons taking the user to different parts of the program, these buttons are students, student statistics/stats, reports, generate report, grades and edit details. These buttons will take the user to that page. On the right of the home page are two more buttons titled sign out and quit, these buttons have been placed away from the other buttons to differentiate between them.

## Code

```
unitHomePage
unit unitHomePage;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, DB, ADODB, unitStudentsPage, unitClassesPage, unitReports,
  unitGradesPage, unitGenerateReport, unitStudentPerformancePage,
```

```
unitEditDetailsPage;

type
  TfrmHomePage = class(TForm)

// All the variables for the objects in the form have been defined
  IblHomePage: TLabel;
  IblFullName: TLabel;
  IblStaffID: TLabel;
  IblEmail: TLabel;
  btnStudents: TButton;
  btnReports: TButton;
  btnGrades: TButton;
  btnGenerateReport: TButton;
  btnPerformance: TButton;
  btnQuit: TButton;
  btnSignOut: TButton;
  btnEditDetails: TButton;

// Event procedures for clicking the buttons have been defined
procedure btnStudentsClick(Sender: TObject);
procedure btnClassesClick(Sender: TObject);
procedure btnReportsClick(Sender: TObject);
procedure btnGradesClick(Sender: TObject);
procedure btnGenerateReportClick(Sender: TObject);
procedure btnPerformanceClick(Sender: TObject);
procedure btnQuitClick(Sender: TObject);
procedure btnSignOutClick(Sender: TObject);
procedure btnEditDetailsClick(Sender: TObject);
end;

var
  frmHomePage: TfrmHomePage;

implementation

uses unitLogin;

{$R *.dfm}

// Each button has been fully implemented, when clicking on the button it closes
the current page and opens the page that the button is supposed to take you to
procedure TfrmHomePage.btnStudentsClick(Sender: TObject);
begin
  Hide;
```

```
frmStudentsPage.Show;
end;

procedure TfrmHomePage.btnClassesClick(Sender: TObject);
begin
  Hide;
  frmClassesPage.Show;
end;

procedure TfrmHomePage.btnReportsClick(Sender: TObject);
begin
  Hide;
  frmReports.Show;
end;

procedure TfrmHomePage.btnGradesClick(Sender: TObject);
begin
  Hide;
  frmHomePage.Show;
end;

procedure TfrmHomePage.btnGenerateReportClick(Sender: TObject);
begin
  Hide;
  frmGenerateReportPage.Show;
end;

procedure TfrmHomePage.btnPerformanceClick(Sender: TObject);
begin
  Hide;
  frmStudentPerformancePage.Show;
end;

// To close the program, this function is used, it will close the program successfully
procedure TfrmHomePage.btnExitClick(Sender: TObject);
begin
  Application.Terminate;
end;

procedure TfrmHomePage.btnExitClick(Sender: TObject);
begin
  Hide;
  frmLogin.Show;
end;
```

```

procedure TfrmHomePage.btnAddDetailsClick(Sender: TObject);
begin
  Hide;
  frmEditDetails.Show;
end;

end.

```

## Requirements met

This has met requirements 7, 26, 28 and 29.

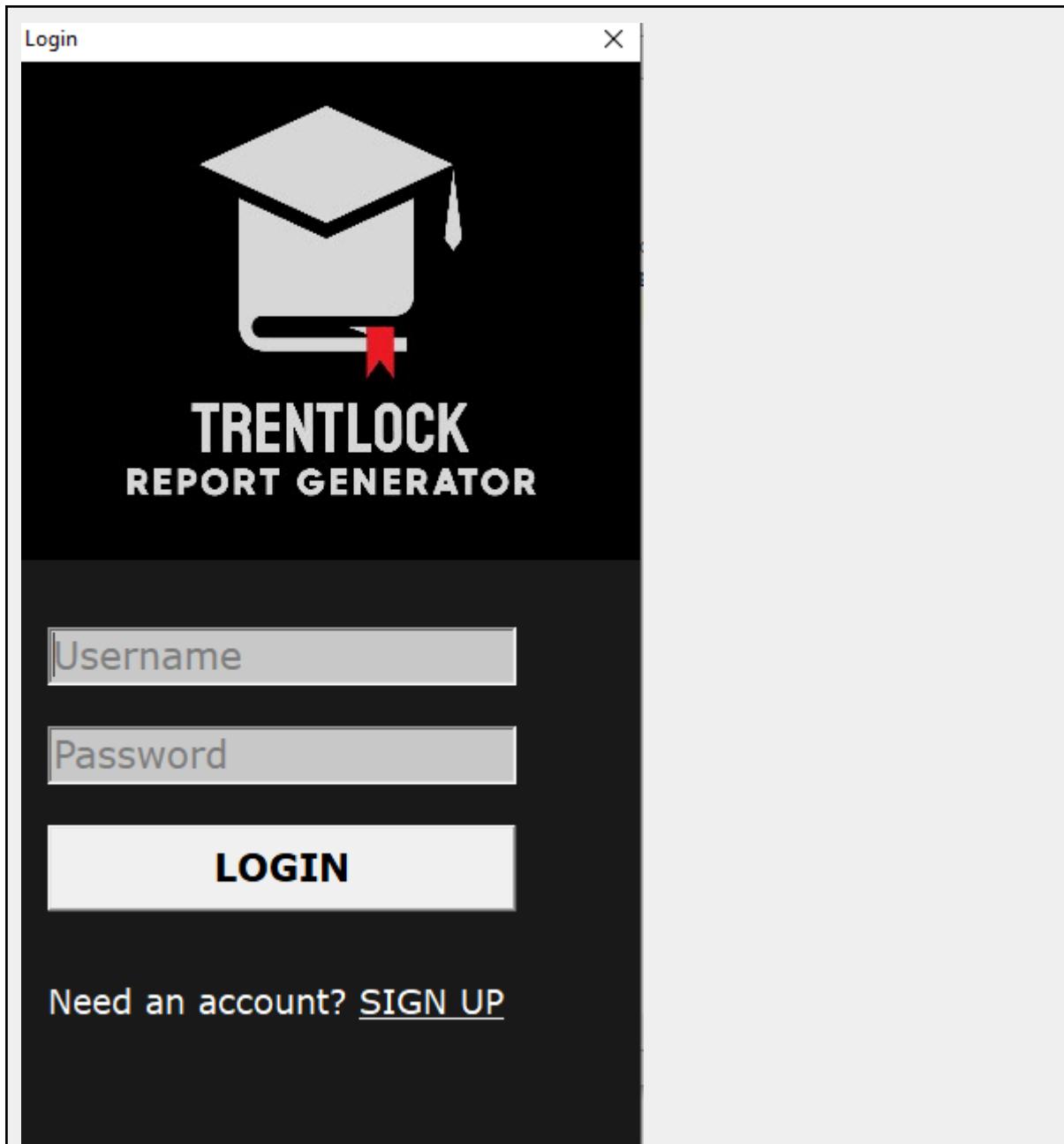
Number	Feature	Explanation and justification
7.	The main program forms must follow a 16:9 aspect ratio and be smaller than the monitor size.	The main forms must follow the standard monitor aspect ratio, they should also be smaller than the monitor to allow for a wider range of monitors. By doing this monitors with a 720p display or lower will still be able to run the program and view each form fully.
26.	Each page should contain a large title at the top with the name of the page.	The user should always know what page they are on, the best way of doing this is having a large title at the top of each page, this makes it clear to the user.
28.	Home page should contain a sign out or quit button.	The home page will be the central part of the program which will be accessed often. A quit button allows the user to quit the application once they have done using it, a sign out button allows the user to sign out and takes them back to the login page. If there is a swap with staff members this is useful.
29.	Home page should contain 6 buttons going to 6 different forms. These are students, grades, student stats, reports,	The home page should be the central point of the application, all the main pages should diverge from the home page, this is the

	edit details and generate reports.	quickest way of navigating the program, having all the main pages at the home page means the user can quickly use the program and get to anything they want.
--	------------------------------------	--

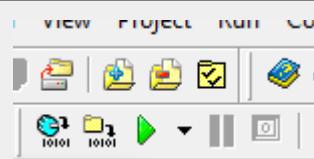
## Testing

Test number	Test purpose	Test data	Expected result	Result
3.1	Check if sign out buttons signs the user out	Sign out button click	Home page closes, login page opens.	Success
3.2	Check if quit button closes the program	Quit button click	Program closes.	Success
3.4	Check if 'students' button opens student page	Students button click	Home page closes, students page opens	Success
3.5	Check if 'student stats' button opens student performance page	Student stats button click	Home page closes, student performance page opens	Success
3.6	Check if 'grades' button opens grades page	grades button click	Home page closes, grades page opens	Fail
3.7	Check if 'reports' button opens reports page'	reports button click	Home page closes, reports page opens	Success
3.8	Check if 'edit details' button opens edit details page'	edit details button click=	Home page closes, edit details page opens	Success
3.9	Check if 'generate report' button opens generate report page	generate report button click	Home page closes, generate report page opens	Success

### 3.1 Result



### 3.2 Result (application not running in delphi environment)



### 3.4 Result

frmStudents

### 3.5 Result

frmStudentPerformance

### 3.6 Previous code

```
procedure TfrmHomePage.btnAddClick(Sender: TObject);
begin
  Hide;
  frmHomePage.Show;
end;
```

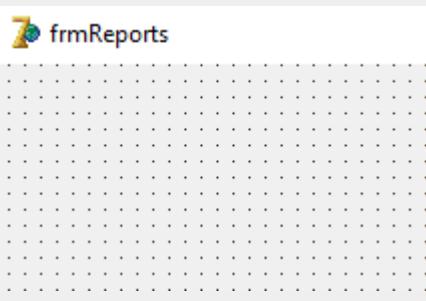
### 3.6 Amended code

```
procedure TfrmHomePage.btnAddClick(Sender: TObject);
begin
  Hide;
  frmGrades.Show;
end;
```

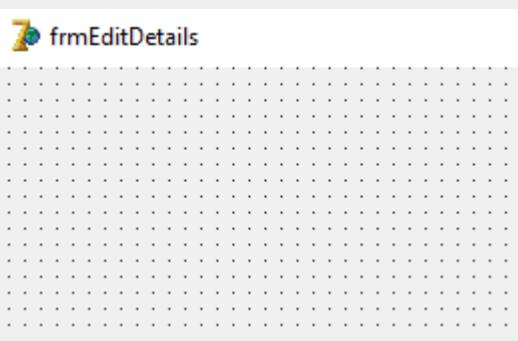
### 3.6 Result

frmGrades

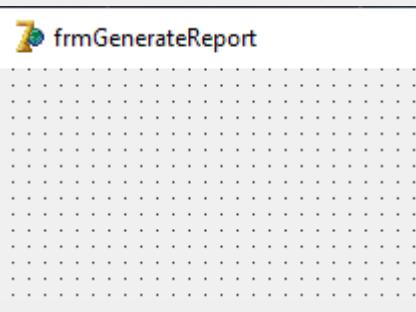
### 3.7 Result



### 3.8 Result



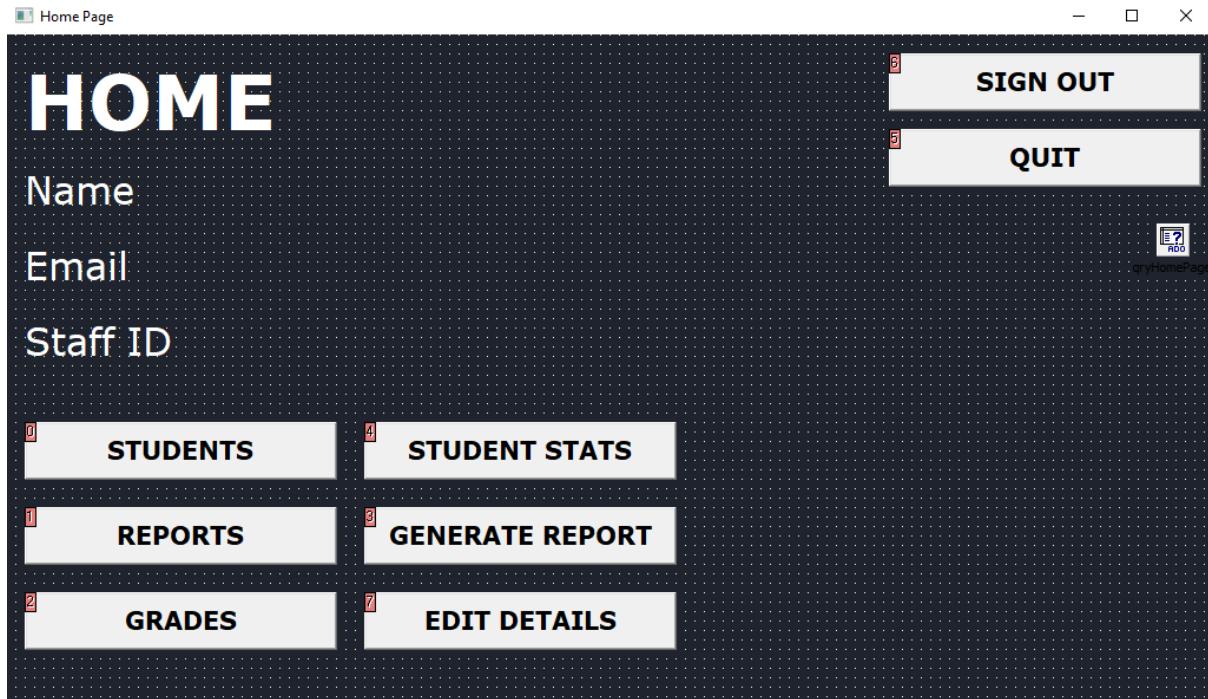
### 3.9 Result



## Summary

### 3.4.2 Prototype 2

#### Form screenshot



## Form accomplishments

The second prototype of the home page has now been made. This includes visual improvements to the UI. The background has changed colour to a dark blue colour which is easier to look at and improves the look. The title and buttons have been adjusted in size to fit the page better and improve the design. Every label and button now uses verdana font. An adoQuery object has been added at the top right to be used in the code.

## Code

```
unitHomePage
unit unitHomePage;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, DB, ADODB, unitStudentsPage, unitClassesPage, unitReports,
  unitGradesPage, unitGenerateReport, unitStudentPerformancePage,
  unitEditDetailsPage;

type
```

```
TfrmHomePage = class(TForm)

  lblHomePage: TLabel;
  lblFullName: TLabel;
  lblStaffID: TLabel;
  lblEmail: TLabel;
  btnStudents: TButton;
  btnReports: TButton;
  btnGrades: TButton;
  btnGenerateReport: TButton;
  btnPerformance: TButton;
  qryHomePage: TADOQuery;
  btnQuit: TButton;
  btnSignOut: TButton;
  btnEditDetails: TButton;

  procedure FormShow(Sender: TObject);
  procedure btnStudentsClick(Sender: TObject);
  procedure btnClassesClick(Sender: TObject);
  procedure btnReportsClick(Sender: TObject);
  procedure btnGradesClick(Sender: TObject);
  procedure btnGenerateReportClick(Sender: TObject);
  procedure btnPerformanceClick(Sender: TObject);
  procedure btnQuitClick(Sender: TObject);
  procedure btnSignOutClick(Sender: TObject);
  procedure btnEditDetailsClick(Sender: TObject);

private
  procedure DisplayStaffDetails;
end;

var
  frmHomePage: TfrmHomePage;

implementation

uses unitLogin;

{$R *.dfm}

procedure TfrmHomePage.FormShow(Sender: TObject);
begin
  DisplayStaffDetails;
end;
```

```
procedure TfrmHomePage.btnAddClick(Sender: TObject);
begin
 Hide;
 frmStudentsPage.Show;
end;

procedure TfrmHomePage.btnAddClassesClick(Sender: TObject);
begin
 Hide;
 frmClassesPage.Show;
end;

procedure TfrmHomePage.btnAddReportsClick(Sender: TObject);
begin
 Hide;
 frmReports.Show;
end;

procedure TfrmHomePage.btnAddGradesClick(Sender: TObject);
begin
 Hide;
 frmGradesPage.Show;
end;

procedure TfrmHomePage.btnAddGenerateReportClick(Sender: TObject);
begin
 Hide;
 frmGenerateReportPage.Show;
end;

procedure TfrmHomePage.btnAddPerformanceClick(Sender: TObject);
begin
 Hide;
 frmStudentPerformancePage.Show;
end;

// A new procedure has been added for displaying the staff details on the home page, this uses the adoQuery and gets all the field required
procedure TfrmHomePage.DisplayStaffDetails;
var
 sqlStatement : string;
 firstName : string;
 lastName : string;
 email : string;
begin
```

```

sqlStatement := 'SELECT * FROM Staff WHERE staffID = ' + frmLogin.StaffID;
// Sets the sql statement string to select all fields from the staff table, uses the public variable from the login form called staffID
with qryHomePage do begin
  Close;
  SQL.Clear;
  SQL.Add(sqlStatement);
  Open;

// If there are record then get the first name, last name and email and assign them to the variables
if RecordCount <> 0 then begin
  firstName := FieldByName('firstName').AsString;
  lastName := FieldByName('lastName').AsString;
  email := FieldByName('email').AsString;

// Then it updates the captions to display the staff information
  lblFullName.Caption := 'Name: ' + firstName + ' ' + lastName;
  lblStaffID.Caption := 'Staff ID: ' + frmLogin.StaffID;
  lblEmail.Caption := 'Email: ' + email;
end;
end;
end;

procedure TfrmHomePage.btnExitClick(Sender: TObject);
begin
  Application.Terminate;
end;

procedure TfrmHomePage.btnExitClick(Sender: TObject);
begin
  Hide;
  frmLogin.Show;
end;

procedure TfrmHomePage.btnExitClick(Sender: TObject);
begin
  Hide;
  frmEditDetails.Show;
end;

end.

```

## Requirements met

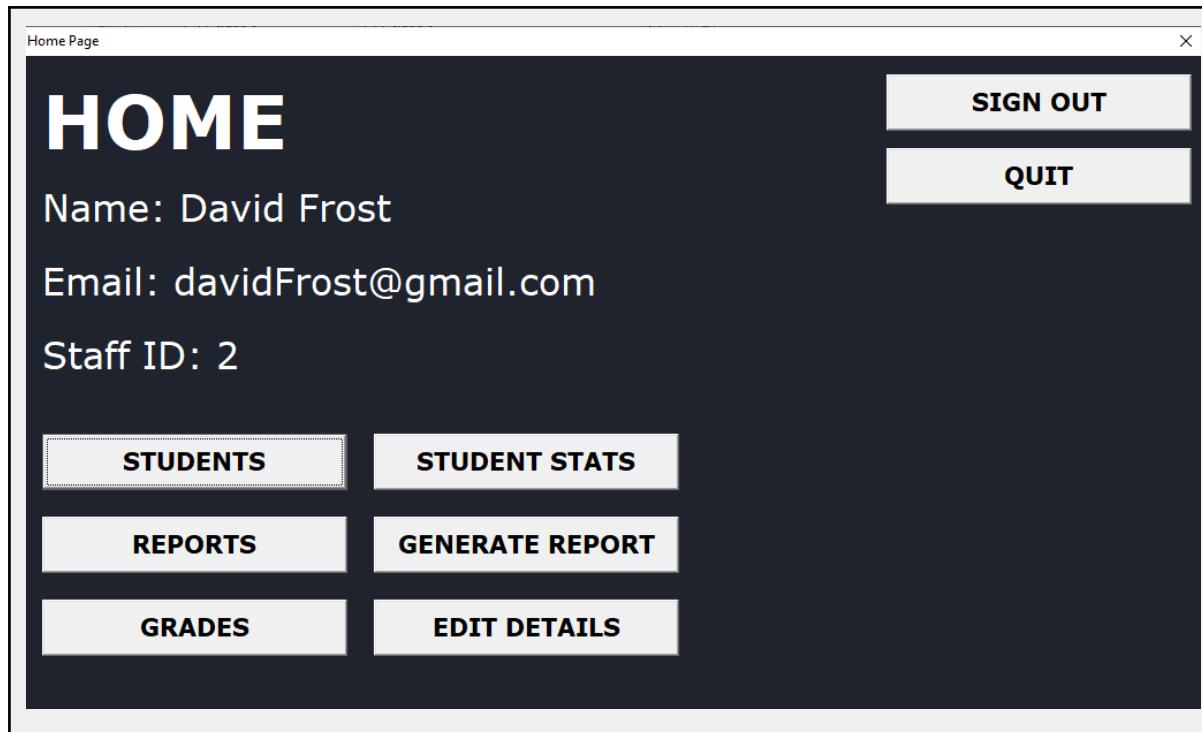
This has met requirements 11, 12 and 13.

Number	Feature	Explanation and justification
11.	Each form should have a dark blue background with contrasting white text.	The software should be easy to look at and have a clear layout, by having a dark blue background it creates a good look without being distracting. During the interview with Mr. Nicolston this was outlined as a key requirement.
12.	Every title, button and edit box should be clearly labelled.	Once again the software should be easy to use, the user should understand what everything does right away, this requirement will help with this.
13.	Verdana font should be used throughout each form. Verdana size 16 bold should be used for each button and size 16 regular for each edit box. Each title should be size 48 bold.	The verdana font is a clean professional look, it will improve the UI significantly if this font is used throughout the program with consistent sizing and bold or regular formats.

## Testing

Test number	Test purpose	Test data	Expected result	Result
3.3	Check, staff ID, email and full name display on home page	username: ‘davidfrost’ password: ‘davidfrost!’	‘Name - David frost’ ‘Email - davidFrost@gmail.com ,	Success

### 3.3 Result



## 3.5 Stage 5 Students page

### 3.5.1 Prototype 1

#### Form screenshot

The screenshot shows a 'Your Students' form with the following components:

- Table:** A grid displaying student data with columns: studentID, firstName, lastName, and yearGroup. The data includes 19 rows of student names and their respective details.
- Buttons:** 'Go back' and 'New'.
- Search Fields:** 'First name' (with value '1'), 'Last name' (with value '2'), and 'Student ID' (with value '3').
- Icons:** Icons for 'Help', 'Print', and 'Save'.

studentID	firstName	lastName	yearGroup
1	Jamarion	Green	09
2	Jaylan	Trujillo	10
3	Barrett	Hays	09
7	Zaniyah	Harrington	11
8	Keith	Zimmerman	07
9	Freddy	Merritt	12
10	Piper	Mckenzie	08
11	Landyn	Mcclain	12
12	Peter	Lara	13
13	Kassidy	Sims	11
14	Landen	Proctor	09
15	Amari	Mccarty	08
16	Uriah	Edwards	12
17	Jerry	Beasley	10
18	Lacey	Townsend	13
19	Kaitlin	Best	12

## Form accomplishments

The first prototype of the students page has been made. This form contains a large table in the centre which is a dbgrid component. There is a title at the top of the page showing the name of the page. To the right of the page is a go back button to return the user to the home page. 3 edit boxes have been added at the bottom of the form, these are first name, last name and student ID. A data source and adoQuery have been added to the form, this allows the dbgrid to connect to the database and display the students.

## Code

### unitStudentsPage

```
unit unitStudentsPage;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, DB, ADODB, Grids, DBGrids, ExtCtrls, DBCtrls, StdCtrls, Mask;

type
  TfrmStudentsPage = class(TForm)
    IblMainText: TLabel;
    IblText1: TLabel;
    IblText2: TLabel;
    btnGoBack: TButton;
    dsStudents: TDataSource;
    dbgrdStudents: TDBGrid;
    qryStudents: TADOQuery;
    edtSearchFirstName: TEdit;
    edtSearchLastName: TEdit;
    edtSearchID: TEdit;

    procedure FormShow(Sender: TObject);
    procedure btnGoBackClick(Sender: TObject);
    procedure edtSearchFirstNameChange(Sender: TObject);
    procedure edtSearchLastNameChange(Sender: TObject);
    procedure edtSearchIDChange(Sender: TObject);
```

```
// private procedure for setting the SQL to update the dbgrid and make the dbgrid
// display the students of the staff member only
private
  procedure SetSQL;
end;

var
  frmStudentsPage: TfrmStudentsPage;

implementation

uses unitHomePage, unitLogin;

{$R *.dfm}

// When the form is show, it runs this procedure to display staffs students straight
// away
procedure TfrmStudentsPage.FormShow(Sender: TObject);
begin
  SetSQL;
end;

// Hides this page and opens the home page
procedure TfrmStudentsPage.btnGoBackClick(Sender: TObject);
begin
  Hide;
  frmHomePage.Show;
end;

// This event function is triggered when the user updates the first name edit box,
// so when the user types in a single key this program is run and updates the query
// to select the student that is typed in
procedure TfrmStudentsPage.edtSearchFirstNameChange(Sender: TObject);
var
  inputText : string;
  searchOptions : TLocateOptions;
begin
  inputText := TEdit(Sender).Text; // Input text is set to the text that is in the first
  // name edit box
  searchOptions := [loPartialKey]; // a search option is set using a pascal feature,
  // the loPartialKey allows for only a part of what the staff inputs to be searched, so if
  // they search for only a few letters it will still update and select the right name
  qryStudents.Locate('firstName', inputText, searchOptions); // Updates the table
  // selection
end;
```

```
// Same search functionally carried out for last name
procedure TfrmStudentsPage.edtSearchLastNameChange(Sender: TObject);
var
  inputText : string;
  searchOptions : TLocateOptions;
begin
  inputText := TEdit(Sender).Text;
  searchOptions := [loPartialKey];
  qryStudents.Locate('lastName', inputText, searchOptions);
end;

// Same search functionality carried out for the student ID
procedure TfrmStudentsPage.edtSearchIDChange(Sender: TObject);
var
  inputText: string;
  value : integer;
begin
  inputText := TEdit(Sender).Text;
  if (TryStrToInt(inputText, value)) then // if the box is empty, it will give an error, so
  this function returns true if the string can be converted to an integer
    qryStudents.Locate('studentID', value, [])
end;

// Setting sql procedure has been implemented
procedure TfrmStudentsPage.SetSQL;
var
  sqlStatement : string;
begin
  // Sets the sql statement string variable
  sqlStatement :=
    'SELECT DISTINCT Students.studentID, firstName, lastName, yearGroup,
formGroup, studentEmail, parentEmail, candidateNumber ' +
    'FROM Students INNER JOIN Reports ' +
    'ON Students.studentID = Reports.studentID ' +
    'WHERE Reports.staffID = ' + frmLogin.StaffID;

  // Opens student query and the table will now be updated with only the student
  // the staff member teachers
  with qryStudents do begin
    Close;
    SQL.Clear;
    SQL.Add(sqlStatement);
    Open;
  end;
end;
```

```
end;
end.
```

### Requirements met

This has met requirements 7, 26, 27, 35, 36, 41.

Number	Feature	Explanation and justification
7.	The main program forms must follow a 16:9 aspect ratio and be smaller than the monitor size.	The main forms must follow the standard monitor aspect ratio, they should also be smaller than the monitor to allow for a wider range of monitors. By doing this monitors with a 720p display or lower will still be able to run the program and view each form fully.
26.	Each page should contain a large title at the top with the name of the page.	The user should always know what page they are on, the best way of doing this is having a large title at the top of each page, this makes it clear to the user.
27.	Each page should contain a 'Go Back' button at the top right to return to the previous page.	To navigate the program quickly there should be a back button on each page, this will go to the previous page, if a user accidentally misclicks they should easily be able to go back. By having this at the same position on each page the user will become familiar with it and can use it quickly.
35.	Students page must contain a large table in the centre of the page with 3 search boxes below which are: first name, last name and student ID.	In order for the staff to navigate the students table there should be search boxes easily readable and accessible. Having these search boxes at the bottom of the page makes it easy to search for a student.
36.	Program must allow staff to view their students in a	Staff will want to view their students, by adding this form it will

	database grid from the students table.	contain a large table showing their students and important information.
41.	Program must allow staff to view students' information in a database grid from various tables in the database. This would include grades, attendance, achievementPoints.	The staff need to know important information about their students to help when creating their report, these forms allow the staff to quickly check how their student is doing and make a decision.

## Testing

Test number	Test purpose	Test data	Expected result	Result
4.1	Check if go back button works on students page	go back button click	Students page closes, home page opens	Success
4.2	Check if student table displays correctly when opening the form, showing staffs students	staff ID: 2	Table should be populated with the students that staff with staff ID 2 search	Success
4.3	Check if first name search box correctly updates the table	first name: 'Jerry'	Table selection updates to locate search input	Success
4.4	Check if last name text box correctly updates the table	last name: 'Leon'	Table selection updates to locate search input	Success
4.5	Check if student ID search box correctly updates the table	Student ID: 14	Table selection updates to locate search input	Fail

### 4.1 Results

Home Page X

# HOME

Name: David Frost

Email: davidFrost@gmail.com

Staff ID: 2

**STUDENTS**      **STUDENT STATS**

**REPORTS**      **GENERATE REPORT**

**GRADES**      **EDIT DETAILS**

## 4.2 Results

Students X

### Your Students

Go Back

student	firstName	lastName	yearGroup	formGroup
10	Piper	Mckenzie	08	RRW
11	Landyn	Mcclain	12	SAA
12	Peter	Lara	13	SHP
13	Kassidy	Sims	11	SLH
14	Landen	Proctor	09	TJG
15	Amari	McCarty	08	WJC
16	Uriah	Edwards	12	LYG
17	Jerry	Beasley	10	LAL
18	Lacey	Townsend	13	TJG
19	Kaitlin	Best	12	BJH
20	Abraham	Leon	08	ADL

First Name Last Name ID 

## 4.3 Results

Students X

## Your Students

Go Back

student	firstName	lastName	yearGroup	formGroup
10	Piper	McKenzie	08	RRW
11	Landyn	McClain	12	SAA
12	Peter	Lara	13	SHP
13	Kassidy	Sims	11	SLH
14	Landen	Proctor	09	TJG
15	Amari	McCarty	08	WJC
16	Uriah	Edwards	12	LYG
► 17	Jerry	Beasley	10	LAL
18	Lacey	Townsend	13	TJG
19	Kaitlin	Best	12	BJH
20	Abraham	Leon	08	ADL

First Name  Last Name  ID

#### 4.4 Results

Students X

## Your Students

Go Back

student	firstName	lastName	yearGroup	formGroup
10	Piper	McKenzie	08	RRW
11	Landyn	McClain	12	SAA
12	Peter	Lara	13	SHP
13	Kassidy	Sims	11	SLH
14	Landen	Proctor	09	TJG
15	Amari	McCarty	08	WJC
16	Uriah	Edwards	12	LYG
17	Jerry	Beasley	10	LAL
18	Lacey	Townsend	13	TJG
19	Kaitlin	Best	12	BJH
► 20	Abraham	Leon	08	ADL

First Name  Last Name  ID

#### 4.5 Previous code

```

begin
  inputText := TEdit(Sender).Text;
  qryStudents.Locate('studentID', inputText, [])
end;

```

#### 4.5 Amended code

```

begin
  inputText := TEdit(Sender).Text;
  if (TryStrToInt(inputText, value)) then
    qryStudents.Locate('studentID', value, [])
end;
  
```

#### 4.5 Results

Students X

**Your Students** Go Back

student	firstName	lastName	yearGroup	formGroup
10	Piper	Mckenzie	08	RRW
11	Landyn	Mcclain	12	SAA
12	Peter	Lara	13	SHP
13	Kassidy	Sims	11	SLH
14	Landen	Proctor	09	TJG
15	Amari	McCarty	08	WJC
16	Uriah	Edwards	12	LYG
17	Jerry	Beasley	10	LAL
18	Lacey	Townsend	13	TJG
19	Kaitlin	Best	12	BJH
20	Abraham	Leon	08	ADL

<
>

First Name  Last Name  ID

#### Summary

#### 3.5.2 Prototype 2

##### Form screenshot

Students

# YOUR STUDENTS

(View Only)

Student ID	First Name	Last Name	Year Group	Form Group	Student Email

GO BACK

orystudentsdssStudents

SEARCH:  First Name  Last Name  Student ID

## Form accomplishments

The second prototype for the students page has now been implemented, this includes major visual improvements to the overall look and UI. Additionally labels have been added to help users understand better the functionality of the form and the purpose of the text boxes. One of the labels is at the top of the form below the title, this outlines the fact that the table is a view only and the staff cannot edit it. Another label is displayed beside the search boxes that indicate to the user they must search in those boxes. The font has been changed for all labels and buttons, they now use verdana in regular or bold. The title, table size and button sizes have all been resized and repositioned for a better look and user experience. The form now uses a dark blue background which improves the look, the table uses a grey colour with blue headers.

## Code

```
unitStudentsPage
unit unitStudentsPage;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
```

```

Dialogs, DB, ADODB, Grids, DBGrids, ExtCtrls, DBCtrls, StdCtrls, Mask;

type
  TfrmStudentsPage = class(TForm)

    lblMainText: TLabel;
    lblText1: TLabel;
    lblText2: TLabel;
    btnGoBack: TButton;
    dsStudents: TDataSource;
    dbgrdStudents: TDBGrid;
    qryStudents: TADOQuery;
    edtSearchFirstName: TEdit;
    edtSearchLastName: TEdit;
    edtSearchID: TEdit;

// Second prototype mostly contains code to allow for the visual change, functions defined here

procedure FormShow(Sender: TObject);
procedure btnGoBackClick(Sender: TObject);
procedure dbgrdStudentsDrawColumnCell(Sender: TObject;
  const Rect: TRect; DataCol: Integer; Column: TColumn;
  State: TGridDrawState);
procedure dbgrdStudentsTitleClick(Column: TColumn);
procedure edtSearchFirstNameClick(Sender: TObject);
procedure edtSearchFirstNameKeyPress(Sender: TObject; var Key: Char);
procedure edtSearchLastNameClick(Sender: TObject);
procedure edtSearchLastNameKeyPress(Sender: TObject; var Key: Char);
procedure edtSearchIDClick(Sender: TObject);
procedure edtSearchIDKeyPress(Sender: TObject; var Key: Char);
procedure edtSearchFirstNameChange(Sender: TObject);
procedure edtSearchLastNameChange(Sender: TObject);
procedure edtSearchIDChange(Sender: TObject);
procedure FormCreate(Sender: TObject);

private
  sortByToggle : boolean; // Used for sorting functionality explained below
  searchFirstNamePromptHidden : boolean;
  searchLastNamePromptHidden : Boolean;
  searchIDPromptHidden : boolean;
// variables to control if the prompt is displayed inside the edit box

procedure SetSQL;
procedure HideTextPrompt(textEdit : TEdit; var hidden : boolean);

```

```
end;

var
  frmStudentsPage: TfrmStudentsPage;

implementation

uses unitHomePage, unitLogin;

{$R *.dfm}

procedure TfrmStudentsPage.FormShow(Sender: TObject);
begin
  SetSQL;
end;

// When the form is created, it sets the first column to a different blue colour, when clicking the column headers the user can sort the columns and the colour will change, by default this first column is set to a darker blue
procedure TfrmStudentsPage.FormCreate(Sender: TObject);
begin
  dbgrdStudents.Columns[0].Title.Color := RGB(28, 54, 120);
end;

procedure TfrmStudentsPage.btnGoBackClick(Sender: TObject);
begin
  Hide;
  frmHomePage.Show;
end;

// This is an event function called when the dbgrid is refreshing, this has been implemented to improve the visual aspect of the table only. It alternates the colours of the rows to make the table more visually pleasing to the user, this was not possible without code
procedure TfrmStudentsPage.dbgrdStudentsDrawColumnCell(Sender: TObject;
  const Rect: TRect; DataCol: Integer; Column: TColumn;
  State: TGridDrawState);
begin
  if (qryStudents.RecNo MOD 2 = 0) then begin // If the row is an even number, set the colour to a darker grey
    if (dbgrdStudents.Canvas.Brush.Color = dbgrdStudents.Color) then begin
      dbgrdStudents.Canvas.Brush.Color := RGB(220, 220, 220);
    end;
  end;
  dbgrdStudents.DefaultDrawColumnCell(Rect, DataCol, Column, State);
```

```
end;

// This event procedure is called when the title of the column is clicked, this allows the user to sort the column by descending or ascending
procedure TfrmStudentsPage.dbgrdStudentsTitleClick(Column: TColumn);
var
  i : integer;
begin
  for i := 0 to dbgrdStudents.Columns.Count - 1 do begin
    dbgrdStudents.Columns[i].Title.Color := RGB(28, 54, 72);
  end;

  for i := 0 to dbgrdStudents.Columns.Count - 1 do begin
    if Column.FieldName = dbgrdStudents.Columns[i].FieldName then begin
      Column.Title.Color := RGB(28, 54, 120); // Sets the colour to a dark blue to indicate
      if not sortByToggle then begin // To alternate between ascending and descending it flips the boolean variable
        qryStudents.Sort := dbgrdStudents.Columns[i].FieldName + ' ASC';
        sortByToggle := true;
      // uses sort function to add 'ASC' keyword to sort the columns
      end
      else begin
        qryStudents.Sort := dbgrdStudents.Columns[i].FieldName + ' DESC';
        sortByToggle := false;
      end;
    end
  end;
end;
end;

// When clicking first name or pressing a key, it runs the Hide text prompt function which removes the prompt from the edit box, this prompt is again a visual improvement
procedure TfrmStudentsPage.edtSearchFirstNameClick(Sender: TObject);
begin
  HideTextPrompt(TEdit(Sender), searchFirstNamePromptHidden);
end;

procedure TfrmStudentsPage.edtSearchFirstNameKeyPress(Sender: TObject; var Key: Char);
begin
  HideTextPrompt(TEdit(Sender), searchFirstNamePromptHidden);
end;

procedure TfrmStudentsPage.edtSearchLastNameClick(Sender: TObject);
```

```
begin
  HideTextPrompt(TEdit(Sender), searchLastNamePromptHidden);
end;

procedure TfrmStudentsPage.edtSearchLastNameKeyPress(Sender: TObject; var
Key: Char);
begin
  HideTextPrompt(TEdit(Sender), searchLastNamePromptHidden);
end;

procedure TfrmStudentsPage.edtSearchIDClick(Sender: TObject);
begin
  HideTextPrompt(TEdit(Sender), searchIDPromptHidden);
end;

procedure TfrmStudentsPage.edtSearchIDKeyPress(Sender: TObject; var Key:
Char);
begin
  HideTextPrompt(TEdit(Sender), searchIDPromptHidden);
end;

procedure TfrmStudentsPage.edtSearchFirstNameChange(Sender: TObject);
var
  inputText : string;
  searchOptions : TLocateOptions;
begin
  inputText := TEdit(Sender).Text;
  searchOptions := [loPartialKey];
  qryStudents.Locate('firstName', inputText, searchOptions);
end;

procedure TfrmStudentsPage.edtSearchLastNameChange(Sender: TObject);
var
  inputText : string;
  searchOptions : TLocateOptions;
begin
  inputText := TEdit(Sender).Text;
  searchOptions := [loPartialKey];
  qryStudents.Locate('lastName', inputText, searchOptions);
end;

procedure TfrmStudentsPage.edtSearchIDChange(Sender: TObject);
var
  inputText: string;
  value : integer;
```

```
begin
  inputText := TEdit(Sender).Text;
  if (TryStrToInt(inputText, value)) then
    qryStudents.Locate('studentID', value, []);
end;

procedure TfrmStudentsPage.SetSQL;
var
  sqlStatement : string;
begin
  sqlStatement :=
    'SELECT DISTINCT Students.studentID, firstName, lastName, yearGroup,
formGroup, studentEmail, parentEmail, candidateNumber ' +
    'FROM Students INNER JOIN Reports ' +
    'ON Students.studentID = Reports.studentID ' +
    'WHERE Reports.staffID = ' + frmLogin.StaffID;

  with qryStudents do begin
    Close;
    SQL.Clear;
    SQL.Add(sqlStatement);
    Open;
  end;
end;

// Sets the text to empty and the colour back to black so the prompt looks as if it has gone away when the user starts typing
procedure TfrmStudentsPage.HideTextPrompt(textEdit : TEdit; var hidden : Boolean);
begin
  if not hidden then begin
    textEdit.Text := '';
    textEdit.Font.Color := clBlack;
    hidden := true;
  end;
end;
end.
```

## Requirements met

This has met requirements 11, 12 and 13.

<b>Number</b>	<b>Feature</b>	<b>Explanation and justification</b>
11.	Each form should have a dark blue background with contrasting white text.	The software should be easy to look at and have a clear layout, by having a dark blue background it creates a good look without being distracting. During the interview with Mr. Nicolston this was outlined as a key requirement.
12.	Every title, button and edit box should be clearly labelled.	Once again the software should be easy to use, the user should understand what everything does right away, this requirement will help with this.
13.	Verdana font should be used throughout each form. Verdana size 16 bold should be used for each button and size 16 regular for each edit box. Each title should be size 48 bold.	The verdana font is a clean professional look, it will improve the UI significantly if this font is used throughout the program with consistent sizing and bold or regular formats.

## Testing

Test number	Test purpose	Test data	Expected result	Result
4.6	Check if clicking column header sorts the column from ascending to descending or descending to ascending	N/A	Data in column should be resorted from ascending or descending	Success

### 4.6 Results

**Students**

# YOUR STUDENTS

**(View Only)**

Student ID	First Name	Last Name	Year Group	Form Group	Student Email
20	Abraham	Leon	08	ADL	5067@trentloc...
15	Amari	Mccarty	08	WJC	5062@trentloc...
17	Jerry	Beasley	10	LAL	5064@trentloc...
19	Kaitlin	Best	12	BJH	5066@trentloc...
13	Kassidy	Sims	11	SLH	5060@trentloc...
18	Lacey	Townsend	13	TJG	5065@trentloc...
14	Landen	Proctor	09	TJG	5061@trentloc...
11	Landyn	Mcclain	12	SAA	5058@trentloc...
12	Peter	Lara	13	SHP	5059@trentloc...
10	Piper	Mckenzie	08	RRW	5057@trentloc...

**SEARCH:**  First Name  Last Name  Student ID

**Students**

# YOUR STUDENTS

**(View Only)**

Student ID	First Name	Last Name	Year Group	Form Group	Student Email
16	Uriah	Edwards	12	LYG	5063@trentloc...
10	Piper	Mckenzie	08	RRW	5057@trentloc...
12	Peter	Lara	13	SHP	5059@trentloc...
11	Landyn	Mcclain	12	SAA	5058@trentloc...
14	Landen	Proctor	09	TJG	5061@trentloc...
18	Lacey	Townsend	13	TJG	5065@trentloc...
13	Kassidy	Sims	11	SLH	5060@trentloc...
19	Kaitlin	Best	12	BJH	5066@trentloc...
17	Jerry	Beasley	10	LAL	5064@trentloc...
15	Amari	Mccarty	08	WJC	5062@trentloc...

**SEARCH:**  First Name  Last Name  Student ID

## 3.6 Stage 6 Student performance page

### 3.6.1 Prototype 1

Form screenshot

attendance	punctuality	behaviourPoints	achievementPoints	studentID
80	96	1	6	1
32	100	4	13	8
97	56	23	2	4
79	6	345	75	2
59	68	4	48	3
98	88	753	67	5
100	68	5	6	6
45	65	345	68	7
12	54	5	86	9
8	91	88	2	10
46	86	3	1	20

## Form accomplishments

The first prototype of the student performance page has been made. This form contains a large table in the centre which is a dbgrid component. There is a title at the top of the page showing the name of the page. To the right of the page is a go back button to return the user to the home page. 3 edit boxes have been added at the bottom of the form, these are first name, last name and student ID. This is similar in design to the students page but the dbgrid will display from a different table in the database. A data source and adoQuery have been added to the form, this allows the dbgrid to connect to the database and display the students.

## Code

### unitStudentPerformancePage

```
unit unitStudentPerformancePage;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, StdCtrls, Grids, DBGrids, DB, ADODB;
```

```

type
TfrmStudentPerformancePage = class(TForm)

  btnGoBack: TButton;
  dsStudents: TDataSource;
  dbgrdStudents: TDBGrid;
  qryStudents: TADOQuery;
  lblMainText: TLabel;
  lblText1: TLabel;
  lblText2: TLabel;
  edtSearchFirstName: TEdit;
  edtSearchLastName: TEdit;
  edtSearchID: TEdit;

  procedure FormShow(Sender: TObject);
  procedure btnGoBackClick(Sender: TObject);
  procedure edtSearchFirstNameChange(Sender: TObject);
  procedure edtSearchLastNameChange(Sender: TObject);
  procedure edtSearchIDChange(Sender: TObject);

// private procedure for setting the SQL to update the dbgrid and make the dbgrid
display the student performance
private
  procedure SetSQL;

end;

var
  frmStudentPerformancePage: TfrmStudentPerformancePage;

implementation

uses unitHomePage, unitLogin;

{$R *.dfm}

// When the form is show, it runs this procedure to display student performance
straight away
procedure TfrmStudentPerformancePage.FormShow(Sender: TObject);
begin
  SetSQL;
end;

// Hides this page and opens the home page
procedure TfrmStudentPerformancePage.btnGoBackClick(Sender: TObject);

```

```
begin
  Hide;
  frmHomePage.Show;
end;

// Setting sql procedure has been implemented
procedure TfrmStudentPerformancePage.SetSQL;
var
  sqlStatement : string;
begin
// Sets the sql statement string variable
  sqlStatement := 
    'SELECT DISTINCT Students.studentID, firstName, lastName, attendance,
    punctuality, behaviourPoints, achievementPoints ' +
    'FROM Students, Performance, Reports ' +
    'WHERE (Students.studentID = Performance.studentID) ' +
    'AND (Reports.studentID = Performance.studentID) ' +
    'AND (Reports.staffID = ' + frmLogin.StaffID + ')';
// Opens student query and the table will now be updated with only the student
the staff member teaches
  with qryStudents do begin
    Close;
    SQL.Clear;
    SQL.Add(sqlStatement);
    Open;
  end;
end;

// This event function is triggered when the user updates the first name edit box,
so when the user types in a single key this program is run and updates the query
to select the student that is typed in
procedure TfrmStudentPerformancePage.edtSearchFirstNameChange(Sender: TObject);
var
  inputText : string;
  searchOptions : TLocateOptions;
begin
  inputText := TEdit(Sender).Text; // Input text is set to the text that is in the first
name edit box
  searchOptions := [loPartialKey]; // a search option is set using a pascal feature,
the loPartialKey allows for only a part of what the staff inputs to be searched, so if
they search for only a few letters it will still update and select the right name
  qryStudents.Locate('firstName', inputText, searchOptions);
// Updates the table selection
```

```

end;

// Same search functionally carried out for last name
procedure TfrmStudentPerformancePage.edtSearchLastNameChange(Sender: TObject);
var
  inputText : string;
  searchOptions : TLocateOptions;
begin
  inputText := TEdit(Sender).Text;
  searchOptions := [loPartialKey];
  qryStudents.Locate('lastName', inputText, searchOptions);
end;

// Same search functionally carried out for Student ID
procedure TfrmStudentPerformancePage.edtSearchIDChange(Sender: TObject);
var
  inputText: string;
  value : integer;
begin
  inputText := Tedit(Sender).Text;
  if (TryStrToInt(inputText, value)) then
    qryStudents.Locate('studentID', value, [])
end;

end.

```

## Requirements met

This has met requirements 7, 26, 27, 37, 38.

Number	Feature	Explanation and justification
7.	The main program forms must follow a 16:9 aspect ratio and be smaller than the monitor size.	The main forms must follow the standard monitor aspect ratio, they should also be smaller than the monitor to allow for a wider range of monitors. By doing this monitors with a 720p display or lower will still be able to run the program and view each form fully.
26.	Each page should contain a large title at the top with the	The user should always know what page they are on, the best way of

	name of the page.	doing this is having a large title at the top of each page, this makes it clear to the user.
27.	Each page should contain a 'Go Back' button at the top right to return to the previous page.	To navigate the program quickly there should be a back button on each page, this will go to the previous page, if a user accidentally misclicks they should easily be able to go back. By having this at the same position on each page the user will become familiar with it and can use it quickly.
37.	Program must allow staff to search for the students full name or student ID.	The staff may have many students so searching for their name or ID is very important to save time. This feature was a key requirement outlined during the second interview with Mr. Nicolston because he said it is important to search through the large number of students.
38.	Student performance page must contain a large table in the centre of the page with 3 search boxes below which are again: first name, last name and student ID.	In order for the staff to navigate the student performance table there should be search boxes easily readable and accessible. Having these search boxes at the bottom of the page makes it easy to search for a student.

## Testing

Test number	Test purpose	Test data	Expected result	Result
5.1	Check if go back button works on student performance page	go back button click	Student performance page closes, home page opens	Success
5.2	Check if performance table displays correctly when opening the form, showing performance of students	staff ID: 2	Table should be populated with the student performance that staff with staff ID 2 search	Success

5.3	Check if first name search box correctly updates the table	first name: 'Abraham'	Table selection updates to locate search input	Success
5.4	Check if last name text box correctly updates the table	last name: 'Mckenzie'	Table selection updates to locate search input	Success
5.5	Check if student ID search box correctly updates the table	Student ID: 20	Table selection updates to locate search input	Success

## 5.1 Results

Home Page X

# HOME

**SIGN OUT**

**QUIT**

Name: David Frost

Email: davidFrost@gmail.com

Staff ID: 2

**STUDENTS**

**STUDENT STATS**

**REPORTS**

**GENERATE REPORT**

**GRADES**

**EDIT DETAILS**

## 5.2 Results

Students X

## Your Students

Go Back

studentID	firstName	lastName	attendance	punctuality	behaviourPoints	achievementPoints
10	Piper	Mckenzie	8	91	88	
20	Abraham	Leon	46	86	3	

First Name  Last Name  ID

### 5.3 Results

Students X

## Your Students

Go Back

studentID	firstName	lastName	attendance	punctuality	behaviourPoints	achievementPoints
10	Piper	Mckenzie	8	91	88	
20	Abraham	Leon	46	86	3	

First Name  Last Name  ID

### 5.4 Results

**Your Students**

studentID	firstName	lastName	attendance	punctuality	behaviourPoints	achievementPoints
10	Piper	Mckenzie	8	91	88	
20	Abraham	Leon	46	86	3	

First Name  Last Name  ID

**5.5 Results**

**Your Students**

studentID	firstName	lastName	attendance	punctuality	behaviourPoints	achievementPoints
10	Piper	Mckenzie	8	91	88	
20	Abraham	Leon	46	86	3	

First Name  Last Name  ID

## Summary

### 3.6.2 Prototype 2

#### Form screenshot

The screenshot shows a software application window titled "Student Performance". At the top right are standard window controls: minimize, maximize, and close. Below the title is a large, bold header "YOUR STUDENTS". To the right of the header is a "GO BACK" button. Underneath the header, the text "Performance Overview (View Only)" is displayed. A table follows, with columns labeled "Student ID", "First Name", "Last Name", "Attendance", "Punctuality", and "Behaviour Poi". The table body is currently empty. At the bottom of the form, there is a search section labeled "SEARCH:" followed by three input fields: "First Name", "Last Name", and "Student ID".

Student ID	First Name	Last Name	Attendance	Punctuality	Behaviour Poi

## Form accomplishments

The second prototype for the student performance page has now been implemented, this includes major visual improvements to the overall look and UI. Additionally labels have been added to help users understand better the functionality of the form and the purpose of the text boxes. One of the labels is at the top of the form below the title, this outlines the fact that the table is a view only and the staff cannot edit it and states the table shows an overview of the students performance. Another label is displayed beside the search boxes that indicate to the user they must search in those boxes. The font has been changed for all labels and buttons, they now use verdana in regular or bold. The title, table size and button sizes have all been resized and repositioned for a better look and user experience. The form now uses a dark blue background which improves the look, the table uses a grey colour with blue headers. The column names have been reordered and re sized

## Code

```
unitStudentPerformancePage
```

```
unit unitStudentPerformancePage;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, StdCtrls, Grids, DBGrids, DB, ADODB;
```

```
type
```

```
TfrmStudentPerformancePage = class(TForm)
```

```
  btnGoBack: TButton;  
  dsStudents: TDataSource;  
  dbgrdStudents: TDBGrid;  
  qryStudents: TADOQuery;  
  lblMainText: TLabel;  
  lblText1: TLabel;  
  lblText2: TLabel;  
  edtSearchFirstName: TEdit;  
  edtSearchLastName: TEdit;  
  edtSearchID: TEdit;
```

*// Second prototype mostly contains code to allow for the visual change, functions defined here*

```
procedure FormShow(Sender: TObject);  
procedure btnGoBackClick(Sender: TObject);  
procedure dbgrdStudentsDrawColumnCell(Sender: TObject;  
  const Rect: TRect; DataCol: Integer; Column: TColumn;  
  State: TGridDrawState);  
procedure dbgrdStudentsTitleClick(Column: TColumn);  
procedure FormCreate(Sender: TObject);  
procedure edtSearchFirstNameClick(Sender: TObject);  
procedure edtSearchFirstNameKeyPress(Sender: TObject; var Key: Char);  
procedure edtSearchLastNameClick(Sender: TObject);  
procedure edtSearchLastNameKeyPress(Sender: TObject; var Key: Char);  
procedure edtSearchIDClick(Sender: TObject);  
procedure edtSearchIDKeyPress(Sender: TObject; var Key: Char);  
procedure edtSearchFirstNameChange(Sender: TObject);  
procedure edtSearchLastNameChange(Sender: TObject);  
procedure edtSearchIDChange(Sender: TObject);
```

```
private
```

```
  sortByToggle : Boolean; // Used for sorting functionality explained below  
  searchFirstNamePromptHidden : boolean;  
  searchLastNamePromptHidden : Boolean;  
  searchIDPromptHidden : boolean;
```

```
// variables to control if the prompt is displayed inside the edit box

procedure SetSQL;
procedure HideTextPrompt(textEdit : TEdit; var hidden : boolean);

end;

var
  frmStudentPerformancePage: TfrmStudentPerformancePage;

implementation

uses unitHomePage, unitLogin;

{$R *.dfm}

procedure TfrmStudentPerformancePage.FormShow(Sender: TObject);
begin
  SetSQL;
end;

// When the form is created, it sets the first column to a different blue colour, when clicking the column headers the user can sort the columns and the colour will change, by default this first column is set to a darker blue
procedure TfrmStudentPerformancePage.FormCreate(Sender: TObject);
begin
  dbgrdStudents.Columns[0].Title.Color := RGB(28, 54, 120);
end;

procedure TfrmStudentPerformancePage.btnGoBackClick(Sender: TObject);
begin
  Hide;
  frmHomePage.Show;
end;

procedure TfrmStudentPerformancePage.SetSQL;
var
  sqlStatement : string;
begin
  sqlStatement :=
    'SELECT DISTINCT Students.studentID, firstName, lastName, attendance,
punctuality, behaviourPoints, achievementPoints ' +
    'FROM Students, Performance, Reports ' +
    'WHERE (Students.studentID = Performance.studentID) ' +
    'AND (Reports.studentID = Performance.studentID) ' +
```

```
'AND (Reports.staffID = ' + frmLogin.StaffID + ')';

with qryStudents do begin
  Close;
  SQL.Clear;
  SQL.Add(sqlStatement);
  Open;
end;
end;

// This is an event function called when the dbgrid is refreshing, this has been implemented to improve the visual aspect of the table only. It alternates the colours of the rows to make the table visually pleasing to the user, this was not possible without code
procedure TfrmStudentPerformancePage.dbgrdStudentsDrawColumnCell(
  Sender: TObject; const Rect: TRect; DataCol: Integer; Column: TColumn;
  State: TGridDrawState);
begin
  if (qryStudents.RecNo MOD 2 = 0) then begin // If the row is an even number, set the colour to a darker grey
    if (dbgrdStudents.Canvas.Brush.Color = dbgrdStudents.Color) then begin
      dbgrdStudents.Canvas.Brush.Color := RGB(220, 220, 220);
    end;
  end;
  dbgrdStudents.DefaultDrawColumnCell(Rect, DataCol, Column, State);
end;

// This event procedure is called when the title of the column is clicked, this allows the user to sort the column by descending or ascending
procedure TfrmStudentPerformancePage.dbgrdStudentsTitleClick(Column: TColumn);
var
  i : integer;
begin
  for i := 0 to dbgrdStudents.Columns.Count - 1 do begin
    dbgrdStudents.Columns[i].Title.Color := RGB(28, 54, 72);
  end;

  for i := 0 to dbgrdStudents.Columns.Count - 1 do begin
    if Column.FieldName = dbgrdStudents.Columns[i].FieldName then begin
      Column.Title.Color := RGB(28, 54, 120); // Sets the colour to a dark blue to indicate
      if not sortByToggle then begin // To alternate between ascending and descending it flips the boolean variable
        qryStudents.Sort := dbgrdStudents.Columns[i].FieldName + ' ASC';
      end;
    end;
  end;
end;
```

```

sortByToggle := true;
// uses sort function to add 'ASC' keyword to sort the columns
  end
else begin
  qryStudents.Sort := dbgrdStudents.Columns[i].FieldName + ' DESC';
  sortByToggle := false;
end;
end;
end;
end;

// When clicking first name or pressing a key, it runs the Hide text prompt function
which removes the prompt from the edit box, this prompt is again a visual
improvement
procedure TfrmStudentPerformancePage.HideTextPrompt(textEdit : TEdit; var
hidden : Boolean);
begin
if not hidden then begin
  textEdit.Text := '';
  textEdit.Font.Color := clBlack;
  hidden := true;
end;
end;

procedure TfrmStudentPerformancePage.edtSearchFirstNameClick(Sender:
TObject);
begin
  HideTextPrompt(TEdit(Sender), searchFirstNamePromptHidden);
end;

procedure TfrmStudentPerformancePage.edtSearchFirstNameKeyPress(Sender:
TObject; var Key: Char);
begin
  HideTextPrompt(TEdit(Sender), searchFirstNamePromptHidden);
end;

procedure TfrmStudentPerformancePage.edtSearchLastNameClick(Sender:
TObject);
begin
  HideTextPrompt(TEdit(Sender), searchLastNamePromptHidden);
end;

procedure TfrmStudentPerformancePage.edtSearchLastNameKeyPress(Sender:
TObject; var Key: Char);
begin

```

```
HideTextPrompt(TEdit(Sender), searchLastNamePromptHidden);
end;

procedure TfrmStudentPerformancePage.edtSearchIDClick(Sender: TObject);
begin
  HideTextPrompt(TEdit(Sender), searchIDPromptHidden);
end;

procedure TfrmStudentPerformancePage.edtSearchIDKeyPress(Sender: TObject;
  var Key: Char);
begin
  HideTextPrompt(TEdit(Sender), searchIDPromptHidden);
end;

procedure TfrmStudentPerformancePage.edtSearchFirstNameChange(Sender: TObject);
var
  inputText : string;
  searchOptions : TLocateOptions;
begin
  inputText := TEdit(Sender).Text;
  searchOptions := [loPartialKey];
  qryStudents.Locate('firstName', inputText, searchOptions);
end;

procedure TfrmStudentPerformancePage.edtSearchLastNameChange(Sender: TObject);
var
  inputText : string;
  searchOptions : TLocateOptions;
begin
  inputText := TEdit(Sender).Text;
  searchOptions := [loPartialKey];
  qryStudents.Locate('lastName', inputText, searchOptions);
end;

procedure TfrmStudentPerformancePage.edtSearchIDChange(Sender: TObject);
var
  inputText: string;
  value : integer;
begin
  inputText := Tedit(Sender).Text;
  if (TryStrToInt(inputText, value)) then
    qryStudents.Locate('studentID', value, [])
end;
```

end.

## Requirements met

This has met requirements 11, 12 and 13.

Number	Feature	Explanation and justification
11.	Each form should have a dark blue background with contrasting white text.	The software should be easy to look at and have a clear layout, by having a dark blue background it creates a good look without being distracting. During the interview with Mr. Nicolston this was outlined as a key requirement.
12.	Every title, button and edit box should be clearly labelled.	Once again the software should be easy to use, the user should understand what everything does right away, this requirement will help with this.
13.	Verdana font should be used throughout each form. Verdana size 16 bold should be used for each button and size 16 regular for each edit box. Each title should be size 48 bold.	The verdana font is a clean professional look, it will improve the UI significantly if this font is used throughout the program with consistent sizing and bold or regular formats.

## Testing

Test number	Test purpose	Test data	Expected result	Result
5.6	Check if clicking column header sorts the column from ascending to descending or descending to ascending	N/A	Data in column should be resorted from ascending or descending	Success

## 5.6 Results

Student Performance X

# YOUR STUDENTS

**GO BACK**

**Performance Overview (View Only)**

Student ID	First Name	Last Name	Attendance	Punctuality	Behaviour P
20	Abraham	Leon	46	86	
10	Piper	Mckenzie	8	91	

**SEARCH:**

Student Performance X

# YOUR STUDENTS

**GO BACK**

**Performance Overview (View Only)**

Student ID	First Name	Last Name	Attendance	Punctuality	Behaviour P
10	Piper	Mckenzie	8	91	
20	Abraham	Leon	46	86	

**SEARCH:**

## 3.7 Stage 7 Grades page

### 3.7.1 Prototype 1

**Form screenshot**

The screenshot shows a Delphi application window titled "All Students". At the top, there are six dropdown menus labeled "Working At Grade", "Target Grade", "Year", "predicted", "EOY Grade", and "Subject". Below these is a large dbgrid component displaying student records. The columns in the dbgrid are: firstName, lastName, studentID, academicYear, subject, workingAt, target, and predicted. The data in the dbgrid is as follows:

firstName	lastName	studentID	academicYear	subject	workingAt	target	predicted
Jamarion	Green	1	2022	Maths	B	A	A*
Jamarion	Green	1	2022	English	C	B	B
Jamarion	Green	1	2022	History	A*	A*	A*
Barrett	Hays	3	2022	Chemistry	B	D	B
Barrett	Hays	3	2021	Chemistry	B	B	C

Below the dbgrid, there are three edit boxes labeled "First name", "Last name", and "Student ID". On the right side of the form, there is a "Go Back" button and two icons labeled "qryGrades" and "dsGrades".

## Form accomplishments

The first prototype for the grades page has been made. This form contains a large table in the centre which is a dbgrid component. At the top there is a large title with the name of the form, below the title there are 6 drop down boxes, these are: working at grade, target grade, year, predicted grade, end of year grade and subject. These drop down boxes can be clicked to drop down and display the grades from A\* to E or the year. These have been positioned clearly above the table. Below the table are 3 edit boxes for first name, last name and student ID. On the right of the form there is a go back button. Additionally a datasource and adoQuery have been added to the form, these are not visible when the program is running but used to access the database in the code.

## Code

### unitGradesPage

```
unit unitGradesPage;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, DB, ADODB, Grids, DBGrids, DBCtrls, ExtCtrls, ComCtrls;
```

```

type
TfrmGradesPage = class(TForm)

  btnGoBack: TButton;
  dbgrdGrades: TDBGrid;
  dsGrades: TDataSource;
  qryGrades: TADOQuery;
  cbbWorkingAt: TComboBox;
  cbbTarget: TComboBox;
  cbbPredicted: TComboBox;
  cbbEndOfYear: TComboBox;
  lblText2: TLabel;
  cbbSubject: TComboBox;
  cbbYear: TComboBox;
  lblMainText: TLabel;
  lblText1: TLabel;
  lblText3: TLabel;
  edtSearchFirstName: TEdit;
  edtSearchLastName: TEdit;
  edtSearchID: TEdit;

```

*// Functions for when changing the drop down combo box have been defined here along with edit box changing*

```

procedure btnGoBackClick(Sender: TObject);
procedure cbbWorkingAtChange(Sender: TObject);
procedure cbbTargetChange(Sender: TObject);
procedure cbbPredictedChange(Sender: TObject);
procedure cbbEndOfYearChange(Sender: TObject);
procedure cbbSubjectChange(Sender: TObject);
procedure cbbYearChange(Sender: TObject);
procedure edtSearchFirstNameChange(Sender: TObject);
procedure edtSearchLastNameChange(Sender: TObject);
procedure edtSearchIDChange(Sender: TObject);

```

*// private procedure defined, this is used for updating the sql query when a field from the drop down box is selected, it takes in a string which is the field name and another string which is the record type, example of its use is below*

```

private
  procedure SetSQL(fieldName : string; recordType : string);
end;
```

```

var
  frmGradesPage: TfrmGradesPage;
```

implementation

uses unitHomePage;

{\$R \*.dfm}

// Hides this page, opens home page

procedure TfrmGradesPage.btnGoBackClick(Sender: TObject);

begin

  Hide;

  frmHomePage.Show;

end;

// Procedure for updating the sql and therefore the table results has been implemented here

procedure TfrmGradesPage.SetSQL(fieldName : string; recordType : string);

var

  sqlStatement : string;

begin

// If the record type from the combo box is set to all, there is no filter and the sql statement is set to this which displays all the grades of the students, otherwise it sets the sql statement to show all the grades of the students where the field name matches the record type

  if recordType = 'ALL' then

    sqlStatement :=

      'SELECT Students.firstName, Students.lastName, Students.studentID,  
      academicYear, subject, workingAt, target, predicted, endOfYear ' +

      'FROM Grades INNER JOIN Students ON Students.studentID =

      Grades.studentID'

  else

    sqlStatement := 'SELECT Students.firstName, Students.lastName,  
      Students.studentID, academicYear, subject, workingAt, target, predicted,  
      endOfYear ' +

      'FROM Grades INNER JOIN Students ON Students.studentID =

      Grades.studentID ' +

      'WHERE ' + fieldName + ' = ' + QuotedStr(recordType);

with qryGrades do begin

  Close;

  SQL.Clear;

  SQL.Add(sqlStatement);

  Open; // Opens query so change takes effect

end;

end;

```
// Goes through each drop down combo box and sets the sql to the field that they
effect, for example the working at combo box runs the sql function with
'workingAt' as the field name and 'A*' 'B' or anything selected as the data
procedure TfrmGradesPage.cbbWorkingAtChange(Sender: TObject);
begin
  SetSQL('workingAt', TComboBox(Sender).Text);
end;

procedure TfrmGradesPage.cbbTargetChange(Sender: TObject);
begin
  SetSQL('target', TComboBox(Sender).Text);
end;

procedure TfrmGradesPage.cbbPredictedChange(Sender: TObject);
begin
  SetSQL('predicted', TComboBox(Sender).Text);
end;

procedure TfrmGradesPage.cbbEndOfYearChange(Sender: TObject);
begin
  SetSQL('endOfYear', TComboBox(Sender).Text);
end;

procedure TfrmGradesPage.cbbSubjectChange(Sender: TObject);
begin
  SetSQL('subject', TComboBox(Sender).Text);
end;

// Same process for the year except it doesn't use the set sql functions parameters
procedure TfrmGradesPage.cbbYearChange(Sender: TObject);
var
  sqlStatement : string;
begin
  if TComboBox(Sender).Text = 'ALL' then
    sqlStatement :=
      'SELECT Students.firstName, Students.lastName, Students.studentID,
      academicYear, subject, workingAt, target, predicted, endOfYear ' +
      'FROM Grades INNER JOIN Students ON Students.studentID =
      Grades.studentID'
  else
    sqlStatement := 'SELECT Students.firstName, Students.lastName,
      Students.studentID, academicYear, subject, workingAt, target, predicted,
      endOfYear ' +
      'FROM Grades INNER JOIN Students ON Students.studentID =
      Grades.studentID ' +
```

```

'WHERE academicYear = ' + TComboBox(Sender).Text;

with qryGrades do begin
  Close;
  SQL.Clear;
  SQL.Add(sqlStatement);
  Open;
end
end;

// This event function is triggered when the user updates the first name edit box,
so when the user types in a single key this program is run and updates the query
to select the student that is typed in
procedure TfrmGradesPage.edtSearchFirstNameChange(Sender: TObject);
var
  inputText : string;
  searchOptions : TLocateOptions;// Input text is set to the text that is in the first
name edit box
begin
  inputText := TEdit(Sender).Text;
  searchOptions := [loPartialKey]; // a search option is set using a pascal feature,
the loPartialKey allows for only a part of what the staff inputs to be searched, so if
they search for only a few letters it will still update and select the right name
  qryGrades.Locate('firstName', inputText, searchOptions);// Updates the table
selection
end;

// Same search functionality carried out for last name
procedure TfrmGradesPage.edtSearchLastNameChange(Sender: TObject);
var
  inputText : string;
  searchOptions : TLocateOptions;
begin
  inputText := TEdit(Sender).Text;
  searchOptions := [loPartialKey];
  qryGrades.Locate('lastName', inputText, searchOptions);
end;

// Same search functionality carried out for the student ID
procedure TfrmGradesPage.edtSearchIDChange(Sender: TObject);
var
  inputText: string;
  value : integer;
begin
  inputText := TEdit(Sender).Text;

```

```

if (TryStrToInt(inputText, value)) then
    qryGradesLocate('studentID', value, [])
end;

end.

```

## Requirements met

This has met requirements 7, 26, 27, 37, 39, 40 and 41.

Number	Feature	Explanation and justification
7.	The main program forms must follow a 16:9 aspect ratio and be smaller than the monitor size.	The main forms must follow the standard monitor aspect ratio, they should also be smaller than the monitor to allow for a wider range of monitors. By doing this monitors with a 720p display or lower will still be able to run the program and view each form fully.
26.	Each page should contain a large title at the top with the name of the page.	The user should always know what page they are on, the best way of doing this is having a large title at the top of each page, this makes it clear to the user.
27.	Each page should contain a 'Go Back' button at the top right to return to the previous page.	To navigate the program quickly there should be a back button on each page, this will go to the previous page, if a user accidentally misclicks they should easily be able to go back. By having this at the same position on each page the user will become familiar with it and can use it quickly.
37.	Program must allow staff to search for the students full name or student ID.	The staff may have many students so searching for their name or ID is very important to save time. This feature was a key requirement outlined during the second interview with Mr. Nicolston because he said it is important to

		search through the large number of students.
39.	Grades page must contain 6 drop down boxes for filtering grades which will be: working at, target, predicted, end of year, subject, year.	If the staff want to search for a specific grade, like all students with an A* last year, or all students with a B in maths they can do so with these drop down boxes. By having these at the top of the page it is clear to the staff how to use them and easy to filter the data whenever. During the questionnaire this was a key requirement that came up, staff members wanted to be able to filter out and view grades quickly which they could not do with the current system.
40.	Grades page must contain a table in the middle after the filter boxes and then 3 search boxes below which are: first name, last name and student ID.	Like the other pages there should be large search boxes at the bottom of the page and the table in the middle of the page, having a large table in the middle allows the staff to view all the details and the search boxes at the bottom allow them to search for students quickly.
41.	Program must allow staff to view students' information in a database grid from various tables in the database. This would include grades, attendance, achievementPoints.	The staff need to know important information about their students to help when creating their report, these forms allow the staff to quickly check how their student is doing and make a decision.

## Testing

Test number	Test purpose	Test data	Expected result	Result
6.1	Check if go back button works on grades page	go back button click	Grades page closes, home page opens	Success
6.2	Check if grades table displays correctly when	N/A	Table should be populated with the	Success

	opening the form		grade information for all students	
6.3	Check if first name search box correctly updates the table	first name: 'Barret'	Table selection updates to locate search input	Success
6.4	Check if last name text box correctly updates the table	last name: 'Green'	Table selection updates to locate search input	Success
6.5	Check if student ID search box correctly updates the table	Student ID: 3	Table selection updates to locate search input	Success
6.7	Check if 'working at' grade drop down box correctly updates table	target grade: 'B'	Table should be sorted and should now only show students with working at grade B	Success

## 6.1 Results

Home Page X

**HOME**
**SIGN OUT**

Name: David Frost
**QUIT**

Email: davidFrost@gmail.com

Staff ID: 2

**STUDENTS**

**STUDENT STATS**

**REPORTS**

**GENERATE REPORT**

**GRADES**

**EDIT DETAILS**

## 6.2 Results

Grades

## All Students

Go Back

Working At Grade ▾ Target Grade ▾ Year ▾

Predicted ▾ EOY Grade ▾ Subject ▾

firstName	lastName	studentID	academicYear	subject	workingAt	target	predicted
Jamarion	Green	1	2022	Maths	B	A	A*
Jamarion	Green	1	2022	English	C	B	B
Jamarion	Green	1	2022	History	A*	A*	A*
Barrett	Hays	3	2022	Chemistry	B	D	B
Barrett	Hays	3	2021	Chemistry	B	B	C

First name \_\_\_\_\_ Last name \_\_\_\_\_ Student ID \_\_\_\_\_

### 6.3 Results

Grades

## All Students

Go Back

Working At Grade ▾ Target Grade ▾ Year ▾

Predicted ▾ EOY Grade ▾ Subject ▾

firstName	lastName	studentID	academicYear	subject	workingAt	target	predicted
Jamarion	Green	1	2022	Maths	B	A	A*
Jamarion	Green	1	2022	English	C	B	B
Jamarion	Green	1	2022	History	A*	A*	A*
Barrett	Hays	3	2022	Chemistry	B	D	B
Barrett	Hays	3	2021	Chemistry	B	B	C

First name \_\_\_\_\_ Last name \_\_\_\_\_ Student ID \_\_\_\_\_

### 6.4 Results

Grades X

## All Students

[Go Back](#)

Working At Grade ▾ Target Grade ▾ Year ▾  
 Predicted ▾ EOY Grade ▾ Subject ▾

firstName	lastName	studentID	academicYear	subject	workingAt	target	predicted
Jamarion	Green	1	2022	Maths	B	A	A*
Jamarion	Green	1	2022	English	C	B	B
Jamarion	Green	1	2022	History	A*	A*	A*
Barrett	Hays	3	2022	Chemistry	B	D	B
Barrett	Hays	3	2021	Chemistry	B	B	C

First name  Last name  Student ID

## 6.5 Results

Grades X

## All Students

[Go Back](#)

Working At Grade ▾ Target Grade ▾ Year ▾  
 Predicted ▾ EOY Grade ▾ Subject ▾

firstName	lastName	studentID	academicYear	subject	workingAt	target	predicted
Jamarion	Green	1	2022	Maths	B	A	A*
Jamarion	Green	1	2022	English	C	B	B
Jamarion	Green	1	2022	History	A*	A*	A*
Barrett	Hays	3	2022	Chemistry	B	D	B
Barrett	Hays	3	2021	Chemistry	B	B	C

First name  Last name  Student ID

## 6.7 Results

Grades

## All Students

Go Back

firstName	lastName	studentID	academicYear	subject	workingAt	target	predicted
Jamarion	Green	1	2022	Maths	B	A	A*
Barrett	Hays	3	2022	Chemistry	B	D	B
Barrett	Hays	3	2021	Chemistry	B	B	C

First name: \_\_\_\_\_ Last name: \_\_\_\_\_ Student ID: \_\_\_\_\_

## Summary

### 3.7.2 Prototype 2

#### Form screenshot

Grades

# ALL STUDENTS

**Grade Overview (View Only)**

**GO BACK**

**FILTER BY:**

Student ID	First Name	Last Name	Year	Subject	Working A
1	Jamarion	Green	2022	Maths	B
1	Jamarion	Green	2022	English	C
1	Jamarion	Green	2022	History	A*
3	Barrett	Hays	2022	Chemistry	B
3	Barrett	Hays	2021	Chemistry	B

**SEARCH:** First Name: \_\_\_\_\_ Last Name: \_\_\_\_\_ Student ID: \_\_\_\_\_

## Form accomplishments

The second prototype for the grades page has now been implemented, this includes major visual improvements to the overall look and UI. Additionally labels have been added to help users understand better the functionality of the form and the purpose of the text boxes. One of the labels is at the top of the form below the title, this outlines the fact that the table is a view only and the staff cannot edit it. A label has been indeed beside the drop down boxes to indicate their purpose and again improve the user experience. Another label is displayed beside the search boxes that indicate to the user they must search in those boxes. The font has been changed for all labels and buttons, they now use verdana in regular or bold. The title, table size and button sizes have all been resized and repositioned for a better look and user experience. The form now uses a dark blue background which improves the look, the table uses a grey colour with blue headers. The drop down boxes and search boxes use a grey background which looks more pleasing for the user. The column headings for the table have been resized and repositioned, the spacing of each heading has been adjusted to better fit the data it displays.

## Code

### **unitGradesPage**

unit unitGradesPage;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, DB, ADODB, Grids, DBGrids, DBCtrls, ExtCtrls, ComCtrls;

type

TfrmGradesPage = class(TForm)

  btnGoBack: TButton;

  dbgrdGrades: TDBGrid;

  dsGrades: TDataSource;

  qryGrades: TADOQuery;

  cbbWorkingAt: TComboBox;

  cbbTarget: TComboBox;

  cbbPredicted: TComboBox;

  cbbEndOfYear: TComboBox;

  lblText2: TLabel;

```
cbbSubject: TComboBox;
cbbYear: TComboBox;
lblMainText: TLabel;
lblText1: TLabel;
lblText3: TLabel;
edtSearchFirstName: TEdit;
edtSearchLastName: TEdit;
edtSearchID: TEdit;

// Event procedures have been defined for changing the visual look of the program through code
procedure btnGoBackClick(Sender: TObject);
procedure cbbWorkingAtChange(Sender: TObject);
procedure cbbTargetChange(Sender: TObject);
procedure cbbPredictedChange(Sender: TObject);
procedure cbbEndOfYearChange(Sender: TObject);
procedure dbgrdGradesDrawColumnCell(Sender: TObject; const Rect: TRect;
  DataCol: Integer; Column: TColumn; State: TGridDrawState);
procedure dbgrdGradesTitleClick(Column: TColumn);
procedure FormCreate(Sender: TObject);
procedure cbbSubjectChange(Sender: TObject);
procedure cbbYearChange(Sender: TObject);
procedure edtSearchFirstNameClick(Sender: TObject);
procedure edtSearchFirstNameKeyPress(Sender: TObject; var Key: Char);
procedure edtSearchLastNameClick(Sender: TObject);
procedure edtSearchLastNameKeyPress(Sender: TObject; var Key: Char);
procedure edtSearchIDClick(Sender: TObject);
procedure edtSearchFirstNameChange(Sender: TObject);
procedure edtSearchLastNameChange(Sender: TObject);
procedure edtSearchIDChange(Sender: TObject);

private
  sortByToggle : Boolean; // Boolean variable to toggle the sorting functionality
  searchFirstNamePromptHidden : boolean; // Boolean variables to control whether the text prompt is hidden
  searchLastNamePromptHidden : Boolean;
  searchIDPromptHidden : boolean;

  procedure SetSQL(fieldName : string; recordType : string);
  procedure HideTextPrompt(textEdit : TEdit; var hidden : boolean);
end;

var
  frmGradesPage: TfrmGradesPage;
```

```

implementation

uses unitHomePage;

{$R *.dfm}

// When the form is created, it sets the first column to a different blue colour, when clicking the column headers the user can sort the columns and the colour will change, by default this first column is set to a darker blue
procedure TfrmGradesPage.FormCreate(Sender: TObject);
begin
  dbgrdGrades.Columns[0].Title.Color := RGB(28, 54, 120);
end;

// Hides this page, opens home page
procedure TfrmGradesPage.btnGoBackClick(Sender: TObject);
begin
  Hide;
  frmHomePage.Show;
end;

procedure TfrmGradesPage.SetSQL(fieldName : string; recordType : string);
var
  sqlStatement : string;
begin

  if recordType = 'ALL' then
    sqlStatement :=
      'SELECT Students.firstName, Students.lastName, Students.studentID,
      academicYear, subject, workingAt, target, predicted, endOfYear ' +
      'FROM Grades INNER JOIN Students ON Students.studentID =
      Grades.studentID'
  else
    sqlStatement := 'SELECT Students.firstName, Students.lastName,
      Students.studentID, academicYear, subject, workingAt, target, predicted,
      endOfYear ' +
      'FROM Grades INNER JOIN Students ON Students.studentID =
      Grades.studentID ' +
      'WHERE ' + fieldName + ' = ' + QuotedStr(recordType);

  with qryGrades do begin
    Close;
    SQL.Clear;
    SQL.Add(sqlStatement);
    Open;
  end;
end;

```

```
end;
end;

procedure TfrmGradesPage.cbbWorkingAtChange(Sender: TObject);
begin
  SetSQL('workingAt', TComboBox(Sender).Text);
end;

procedure TfrmGradesPage.cbbTargetChange(Sender: TObject);
begin
  SetSQL('target', TComboBox(Sender).Text);
end;

procedure TfrmGradesPage.cbbPredictedChange(Sender: TObject);
begin
  SetSQL('predicted', TComboBox(Sender).Text);
end;

procedure TfrmGradesPage.cbbEndOfYearChange(Sender: TObject);
begin
  SetSQL('endOfYear', TComboBox(Sender).Text);
end;

procedure TfrmGradesPage.cbbSubjectChange(Sender: TObject);
begin
  SetSQL('subject', TComboBox(Sender).Text);
end;

procedure TfrmGradesPage.cbbYearChange(Sender: TObject);
var
  sqlStatement : string;
begin
  if TComboBox(Sender).Text = 'ALL' then
    sqlStatement :=
      'SELECT Students.firstName, Students.lastName, Students.studentID,
      academicYear, subject, workingAt, target, predicted, endOfYear ' +
      'FROM Grades INNER JOIN Students ON Students.studentID =
      Grades.studentID'
  else
    sqlStatement := 'SELECT Students.firstName, Students.lastName,
      Students.studentID, academicYear, subject, workingAt, target, predicted,
      endOfYear ' +
      'FROM Grades INNER JOIN Students ON Students.studentID =
      Grades.studentID ' +
      'WHERE academicYear = ' + TComboBox(Sender).Text;
```

```
with qryGrades do begin
  Close;
  SQL.Clear;
  SQL.Add(sqlStatement);
  Open;
end
end;

// This is an event function called when the dbgrid is refreshing, this has been implemented to improve the visual aspect of the table only. It alternates the colours of the rows to make the table visually pleasing to the user, this was not possible without code
procedure TfrmGradesPage.dbgrdGradesDrawColumnCell(Sender: TObject;
  const Rect: TRect; DataCol: Integer; Column: TColumn;
  State: TGridDrawState);
begin
  if (qryGrades.RecNo MOD 2 = 0) then begin // If the row is an even number, set the colour to a darker grey
    if (dbgrdGrades.Canvas.Brush.Color = dbgrdGrades.Color) then begin
      dbgrdGrades.Canvas.Brush.Color := RGB(220, 220, 220);
    end;
  end;
  dbgrdGrades.DefaultDrawColumnCell(Rect, DataCol, Column, State);
end;

// This event procedure is called when the title of the column is clicked, this allows the user to sort the column by descending or ascending
procedure TfrmGradesPage.dbgrdGradesTitleClick(Column: TColumn);
var
  i : integer;
begin
  for i := 0 to dbgrdGrades.Columns.Count - 1 do begin
    dbgrdGrades.Columns[i].Title.Color := RGB(28, 54, 72);
  end;

  for i := 0 to dbgrdGrades.Columns.Count - 1 do begin
    if Column.FieldName = dbgrdGrades.Columns[i].FieldName then begin
      Column.Title.Color := RGB(28, 54, 120); // Sets the colour to a dark blue to indicate
      if not sortByToggle then begin // To alternate between ascending and descending it flips the boolean variable
        qryGrades.Sort := dbgrdGrades.Columns[i].FieldName + ' ASC';
        sortByToggle := true;
      end
    end;
  end;
end;
```

```
else begin
    qryGrades.Sort := dbgrdGrades.Columns[i].FieldName + ' DESC';
    sortByToggle := false;
end;
end;
end;
end;

// Sets the text to empty and the colour back to black so the prompt looks as if it has gone away when the user starts typing
procedure TfrmGradesPage.HideTextPrompt(textEdit : TEdit; var hidden : Boolean);
begin
if not hidden then begin
    textEdit.Text := '';
    textEdit.Font.Color := clBlack;
    hidden := true;
end;
end;

// When clicking first name or pressing a key, it runs the Hide text prompt function which removes the prompt from the edit box, this prompt is again a visual improvement
procedure TfrmGradesPage.edtSearchFirstNameClick(Sender: TObject);
begin
    HideTextPrompt(TEdit(Sender), searchFirstNamePromptHidden);
end;

procedure TfrmGradesPage.edtSearchFirstNameKeyPress(Sender: TObject; var Key: Char);
begin
    HideTextPrompt(TEdit(Sender), searchFirstNamePromptHidden);
end;

procedure TfrmGradesPage.edtSearchLastNameClick(Sender: TObject);
begin
    HideTextPrompt(TEdit(Sender), searchLastNamePromptHidden);
end;

procedure TfrmGradesPage.edtSearchLastNameKeyPress(Sender: TObject; var Key: Char);
begin
    HideTextPrompt(TEdit(Sender), searchLastNamePromptHidden);
end;
```

```

procedure TfrmGradesPage.edtSearchIDClick(Sender: TObject);
begin
  HideTextPrompt(TEdit(Sender), searchIDPromptHidden);
end;

procedure TfrmGradesPage.edtSearchFirstNameChange(Sender: TObject);
var
  inputText : string;
  searchOptions : TLocateOptions;
begin
  inputText := TEdit(Sender).Text;
  searchOptions := [loPartialKey];
  qryGrades.Locate('firstName', inputText, searchOptions);
end;

procedure TfrmGradesPage.edtSearchLastNameChange(Sender: TObject);
var
  inputText : string;
  searchOptions : TLocateOptions;
begin
  inputText := TEdit(Sender).Text;
  searchOptions := [loPartialKey];
  qryGrades.Locate('lastName', inputText, searchOptions);
end;

procedure TfrmGradesPage.edtSearchIDChange(Sender: TObject);
var
  inputText: string;
  value : integer;
begin
  inputText := TEdit(Sender).Text;
  if (TryStrToInt(inputText, value)) then
    qryGrades.Locate('studentID', value, [])
end;

end.

```

## Requirements met

This has met requirements 11, 12 and 13.

Number	Feature	Explanation and justification
11.	Each form should have a dark	The software should be easy to

	blue background with contrasting white text.	look at and have a clear layout, by having a dark blue background it creates a good look without being distracting. During the interview with Mr. Nicolston this was outlined as a key requirement.
12.	Every title, button and edit box should be clearly labelled.	Once again the software should be easy to use, the user should understand what everything does right away, this requirement will help with this.
13.	Verdana font should be used throughout each form. Verdana size 16 bold should be used for each button and size 16 regular for each edit box. Each title should be size 48 bold.	The verdana font is a clean professional look, it will improve the UI significantly if this font is used throughout the program with consistent sizing and bold or regular formats.

## Testing

Test number	Test purpose	Test data	Expected result	Result
6.6	Check if clicking column header sorts the column from ascending to descending or descending to ascending	N/A	Data in column should be resorted from ascending or descending	Success

### 6.6 Results

Grades X

# ALL STUDENTS

**Grade Overview (View Only)**

**FILTER BY:** Working At Grade Target Grade Year  
                   Predicted Grade EOY Grade Subject

Student ID	First Name	Last Name	Year	Subject	Working At
3	Barrett	Hays	2022	Chemistry	B
3	Barrett	Hays	2021	Chemisty	B
1	Jamarion	Green	2022	English	C
1	Jamarion	Green	2022	History	A*
1	Jamarion	Green	2022	Maths	B

**SEARCH:** First Name Last Name Student ID

---

Grades X

# ALL STUDENTS

**Grade Overview (View Only)**

**FILTER BY:** Working At Grade Target Grade Year  
                   Predicted Grade EOY Grade Subject

Student ID	First Name	Last Name	Year	Subject	Working At
1	Jamarion	Green	2022	Maths	B
1	Jamarion	Green	2022	History	A*
1	Jamarion	Green	2022	English	C
3	Barrett	Hays	2021	Chemistry	B
3	Barrett	Hays	2022	Chemistry	B

**SEARCH:** First Name Last Name Student ID

## 3.8 Stage 8 Reports page

### 3.8.1 Prototype 1

Form screenshot

**Your Reports**

reportID	firstName	lastName	reportDocument	dateModified	studentID
1	Jamarion	Green	(MEMO)	28/03/2023	
2	Jaylan	Trujillo	(MEMO)	16/01/2023	
3	Barrett	Hays	(MEMO)	16/01/2023	
4	Skye	Morales	(MEMO)	16/01/2023	
5	Kaylie	Bartlett	(Memo)	17/01/2023	
6	Zavier	Hoffman	(MEMO)	17/01/2023	
7	Zaniyah	Harrington	(MEMO)	17/01/2023	
8	Keith	Zimmerman	(Memo)		
9	Freddy	Merritt	(MEMO)	17/01/2023	
10	Piper	McKenzie	(MEMO)	11/03/2023	1
11	Landyn	Mcclain	(MEMO)	11/03/2023	1
12	Peter	Lara	(Memo)	25/02/2023	1
13	Kassidy	Sims	(MEMO)	17/01/2023	1
14	Landen	Proctor	(MEMO)	20/03/2023	1
15	Amarie	Mccarty	(MEMO)	08/03/2023	1
16	Uriah	Edwards	(Memo)		1
17	Jerry	Beasley	(MEMO)	21/01/2023	1

Full name:  Student ID:   
date modified:

qryReports    dsReports  
qryUpdate

## Form accomplishments

The first prototype for the reports form has been made. This form contains a large title at the top left with the name of the form. On the right side of the page there is a large go back button to take the user to the home page. The page is divided into two separate sections, on the left side is a dbgrid showing the reports table from the database, on the ride side is a large rich text edit box showing the report document. At the bottom of the screen there are 3 search boxes, these are the date modified search box, the full name search box and student ID search box. On the right corner of the form there is a query for the reports and a query for updating the reports, there is also a data source component for displaying the data in the dbgrid table on the right of the page.

## Code

### unitReports

```
unit unitReports;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, StdCtrls, ComCtrls, DBCtrls, Grids, DBGrids, DB, ADODB, Mask;
```

```
type
TfrmReports = class(TForm)

// All variables for the objects in the form have been defined
  btnGoBack: TButton;
  dtpDateModified: TDateTimePicker;
  lblMainText: TLabel;
  lblText1: TLabel;
  dbgrdReports: TDBGrid;
  dsReports: TDataSource;
  qryReports: TADOQuery;
  dbredtReport: TDBRichEdit;
  lblText2: TLabel;
  qryUpdate: TADOQuery;
  edtSearchFullName: TEdit;
  edtSearchID: TEdit;

// event procedures have been defined here for when clicking buttons and using
search boxes
procedure FormShow(Sender: TObject);
procedure btnGoBackClick(Sender: TObject);
procedure dtpDateModifiedChange(Sender: TObject);
procedure edtSearchFullNameChange(Sender: TObject);
procedure edtSearchIDChange(Sender: TObject);

// private procedure called SetSQL, this is used to update the ado query to display
the reports for the staff who is logged in
private
  procedure SetSQL;
end;

var
  frmReports: TfrmReports;

implementation

uses unitHomePage, unitLogin;

{$R *.dfm}

// When the form is show it sets the sql by running the procedure
procedure TfrmReports.FormShow(Sender: TObject);
begin
  SetSQL;

```

```
end;

// Hides this page, opens the home page
procedure TfrmReports.btnGoBackClick(Sender: TObject);
begin
  Hide;
  frmHomePage.Show;
end;

// SetSQL procedure has been implemented, it selects reports from the reports
table where the staffID matches the one with the staff who is logged in
procedure TfrmReports.SetSQL;
var
  sqlStatement : string;
begin
  sqlStatement := 
    'SELECT reportID, firstName, lastName, reportDocument, dateModified,
  Reports.studentID ' +
    'FROM Reports INNER JOIN Students ON Reports.studentID =
  Students.studentID ' +
    'WHERE staffID = ' + frmLogin.StaffID; // uses public variable from login page to
access staff ID

  with qryReports do begin
    Close;
    SQL.Clear;
    SQL.Add(sqlStatement);
    Open; // runs sql
  end;
end;

// This is an event triggered when the user changes the date from the date time
picker, it uses the query locate function to locate the report with the date picked
procedure TfrmReports.dtpDateModifiedChange(Sender: TObject);
var
  inputDate : string;
begin
  inputDate := DateToStr(TDateTimePicker(Sender).Date);
  qryReports.Locate('dateModified', inputDate, []);
end;

// Same process as locating the date time picker but for the full name
procedure TfrmReports.edtSearchFullNameChange(Sender: TObject);
var
  inputText : string;
```

```

searchOptions : TLocateOptions;
begin
  inputText := TEdit(Sender).Text;
  searchOptions := [loPartialKey];
  qryReports.Locate('firstName', inputText, searchOptions);
  qryReports.Locate('lastName', inputText, searchOptions);
end;

// Same process as locating the date time picker but for the student ID
procedure TfrmReports.edtSearchIDChange(Sender: TObject);
var
  inputText: string;
  value : integer;
begin
  inputText := TEdit(Sender).Text;
  if (TryStrToInt(inputText, value)) then // If the edit box is left empty, because it's locating an integer this causes errors, so this function returns true if it's an integer before locating
    qryReports.Locate('studentID', value, [])
  end;
end.

```

## Requirements met

This has met requirements 7, 26, 27, 37, 42 and 43.

Number	Feature	Explanation and justification
7.	The main program forms must follow a 16:9 aspect ratio and be smaller than the monitor size.	The main forms must follow the standard monitor aspect ratio, they should also be smaller than the monitor to allow for a wider range of monitors. By doing this monitors with a 720p display or lower will still be able to run the program and view each form fully.
26.	Each page should contain a large title at the top with the name of the page.	The user should always know what page they are on, the best way of doing this is having a large title at the top of each page, this makes it clear to the user.

27.	Each page should contain a 'Go Back' button at the top right to return to the previous page.	To navigate the program quickly there should be a back button on each page, this will go to the previous page, if a user accidentally misclicks they should easily be able to go back. By having this at the same position on each page the user will become familiar with it and can use it quickly.
37.	Program must allow staff to search for the students full name or student ID.	The staff may have many students so searching for their name or ID is very important to save time. This feature was a key requirement outlined during the second interview with Mr. Nicolston because he said it is important to search through the large number of students.
42.	Reports page must contain a table on the left of the page and an edit box on the right of the page displaying the reports. Below must include 3 search boxes: date modified box, full name and student ID.	There should be a large table on the left side of the form displaying the students and on the right side there should be a large text box displaying the reports. This allows the staff to easily view the table and read through the reports.
43.	Program must allow staff to view their previous reports written for their students.	This allows the staff to view their previous reports and also make edits to their reports if they are not happy with it.

## Testing

Test number	Test purpose	Test data	Expected result	Result
7.1	Check if go back button works on reports page	go back button click	Reports page closes, home page opens	Success
7.2	Check if reports table displays correctly when opening the form	N/A	Table should be populated with the reports table from the database	Success

7.3	Check if full name search box correctly updates the table	first name: 'Kassidy'	Table selection updates to locate search input	Success
7.4	Check if student ID search box correctly updates the table	Student ID: 11	Table selection updates to locate search input	Success
7.5	Check if date picker search box correctly updates the table	date: 21/3/2023	Table selection updates to locate search input	Success
7.6	Check if reports are displayed in the rich edit text box correctly	N/A	Rich edit box should be populated with the report document from the database	Success

## 7.1 Results

Home Page X

**HOME**

Name: David Frost

Email: davidFrost@gmail.com

Staff ID: 2

**STUDENTS**    **STUDENT STATS**

**REPORTS**    **GENERATE REPORT**

**GRADES**    **EDIT DETAILS**

**SIGN OUT**

**QUIT**

## 7.2 Results

Reports X

## Your Reports

Go Back

reportID	firstName	lastName	reportDocume	dateModified	studentID
14	Landen	Proctor	(MEMO)	20/03/2023	1
15	Amari	Mccarty	(MEMO)	08/03/2023	1
16	Uriah	Edwards	(Memo)		1
17	Jerry	Beasley	(MEMO)	21/01/2023	1
18	Lacey	Townsend	(Memo)		1
19	Kaitlin	Best	(Memo)		1
► 20	Abraham	Leon	(MEMO)	17/01/2023	2
21	Abraham	Leon	(MEMO)	17/01/2023	2
22	Landen	Proctor	(MEMO)	20/03/2023	1
23	Landen	Proctor	(MEMO)	20/03/2023	1
24	Barrett	Hays	(MEMO)	20/03/2023	
25	Kaitlin	Best	(MEMO)	21/03/2023	1
26	Zavier	Hoffman	(MEMO)	21/03/2023	
27	Keith	Zimmerman	(MEMO)	21/03/2023	
28	Uriah	Edwards	(MEMO)	22/03/2023	1
29	Jamarion	Green	(MEMO)	28/03/2023	
30	Barrett	Hays	(MEMO)	28/03/2023	

28/03/2023

**THIS IS MY FIRST REPORT FOR LEON**

7.3 Results

Reports X

## Your Reports

Go Back

reportID	firstName	lastName	reportDocume	dateModified	studentID
1	Jamarion	Green	(MEMO)	28/03/2023	
2	Jaylan	Trujillo	(MEMO)	16/01/2023	
3	Barrett	Hays	(MEMO)	16/01/2023	
4	Skye	Morales	(MEMO)	16/01/2023	
5	Kaylie	Bartlett	(Memo)	17/01/2023	
6	Zavier	Hoffman	(MEMO)	17/01/2023	
7	Zaniyah	Harrington	(MEMO)	17/01/2023	
8	Keith	Zimmerman	(Memo)		
9	Freddy	Merritt	(MEMO)	17/01/2023	
10	Piper	McKenzie	(MEMO)	11/03/2023	1
11	Landyn	McDain	(MEMO)	11/03/2023	1
12	Peter	Lara	(Memo)	25/02/2023	1
► 13	Kassidy	Sims	(MEMO)	17/01/2023	1
14	Landen	Proctor	(MEMO)	20/03/2023	1
15	Amari	Mccarty	(MEMO)	08/03/2023	1
16	Uriah	Edwards	(Memo)		1
17	Jerry	Beasley	(MEMO)	21/01/2023	1

28/03/2023

**TEST**

Reports X

## Your Reports

Go Back

reportID	firstName	lastName	reportDocume	dateModified	studentID
1	Jamarion	Green	(MEMO)	28/03/2023	1
2	Jaylan	Trujillo	(MEMO)	16/01/2023	1
3	Barrett	Hays	(MEMO)	16/01/2023	1
4	Skye	Morales	(MEMO)	16/01/2023	1
5	Kaylie	Bartlett	(Memo)	17/01/2023	1
6	Zavier	Hoffman	(MEMO)	17/01/2023	1
7	Zaniyah	Harrington	(MEMO)	17/01/2023	1
8	Keith	Zimmerman	(Memo)		1
9	Freddy	Merritt	(MEMO)	17/01/2023	1
10	Piper	Mckenzie	(MEMO)	11/03/2023	1
► 11	Landyn	Mcclain	(MEMO)	11/03/2023	1
12	Peter	Lara	(Memo)	25/02/2023	1
13	Kassidy	Sims	(MEMO)	17/01/2023	1
14	Landen	Proctor	(MEMO)	20/03/2023	1
15	Amari	Mccarty	(MEMO)	08/03/2023	1
16	Uriah	Edwards	(Memo)		1
17	Jerry	Beasley	(MEMO)	21/01/2023	1

**TODAY**

28/03/2023 Full name Student ID  
11

## 7.5 Results

Reports X

## Your Reports

Go Back

reportID	firstName	lastName	reportDocume	dateModified	studentID
14	Landen	Proctor	(MEMO)	20/03/2023	1
15	Amari	Mccarty	(MEMO)	08/03/2023	1
16	Uriah	Edwards	(Memo)		1
17	Jerry	Beasley	(MEMO)	21/01/2023	1
18	Lacey	Townsend	(Memo)		1
19	Kaitlin	Best	(Memo)		1
20	Abraham	Leon	(MEMO)	17/01/2023	2
21	Abraham	Leon	(MEMO)	17/01/2023	2
22	Landen	Proctor	(MEMO)	20/03/2023	1
23	Landen	Proctor	(MEMO)	20/03/2023	1
24	Barrett	Hays	(MEMO)	20/03/2023	1
25	Kaitlin	Best	(MEMO)	21/03/2023	1
► 26	Zavier	Hoffman	(MEMO)	21/03/2023	
27	Keith	Zimmerman	(MEMO)	21/03/2023	
28	Uriah	Edwards	(MEMO)	22/03/2023	1
29	Jamarion	Green	(MEMO)	28/03/2023	
30	Barrett	Hays	(MEMO)	28/03/2023	

Zavier has been a cause of concern. Zavier is a hard working student, he works well with friends in class and picks up new concepts and skills quickly, applying them confidently to exam questions. The quality of his work is to a high standard which shows Zavier is paying attention in class and working correctly. Zavier has been causeOfConcern in work completion. Zavier has been causeOfConcern in homework quality. Zavier has been causeOfConcern in homework completion. Zavier has been

21/03/2023 Full name Student ID  
11

## 7.6 Results

Reports X

## Your Reports

[Go Back](#)

reportID	firstName	lastName	reportDocume	dateModified	studentID
14	Landen	Proctor	(MEMO)	20/03/2023	1
15	Amari	Mccarty	(MEMO)	08/03/2023	1
16	Uriah	Edwards	(Memo)		1
17	Jerry	Beasley	(MEMO)	21/01/2023	1
18	Lacey	Townsend	(Memo)		1
19	Kaitlin	Best	(Memo)		1
20	Abraham	Leon	(MEMO)	17/01/2023	2
21	Abraham	Leon	(MEMO)	17/01/2023	2
22	Landen	Proctor	(MEMO)	20/03/2023	1
23	Landen	Proctor	(MEMO)	20/03/2023	1
24	Barrett	Hays	(MEMO)	20/03/2023	1
25	Kaitlin	Best	(MEMO)	21/03/2023	1
26	Zavier	Hoffman	(MEMO)	21/03/2023	1
27	Keith	Zimmerman	(MEMO)	21/03/2023	1
28	Uriah	Edwards	(MEMO)	22/03/2023	1
29	Jamarion	Green	(MEMO)	28/03/2023	
30	Barrett	Hays	(MEMO)	28/03/2023	

Zavier has been a cause of concern. Zavier is a hard working student, he works well with friends in class and picks up new concepts and skills quickly, applying them confidently to exam questions. The quality of his work is to a high standard which shows Zavier is paying attention in class and working correctly. Zavier has been causeOfConcern in work completion. Zavier has been causeOfConcern in homework quality. Zavier has been causeOfConcern in homework completion. Zavier has been

## Summary

### 3.8.2 Prototype 2

#### Form screenshot

Reports X

# YOUR REPORTS

(Edit Permissions Enabled)

S-ID	First Name	Last Name	Date Modified

[GO BACK](#)

**abredtReport**

SEARCH:  Full Name

## Form accomplishments

The second prototype for the reports page has now been made. This prototype contains large visual improvements to the UI that improves the user experience. Additional labels have been added to aid the user and help them understand the purpose of each button and edit box. The first label is below the title at the top of the page, this informs the user they are able to edit the reports in the edit box. The second new label is a large search label at the bottom of the page beside the search boxes. This shows the purpose of the boxes below and helps users use the form easily. The labels, buttons and edit boxes have been resized to better fill the page and the font on all components has been changed to verdana in bold or regular font size. The background now uses a dark blue colour which is easier to look at and less distracting. The table has been improved, the column headers are large and spaced appropriately. The edit boxes and reports box are now grey instead of white, this improves the look and readability of the page.

## Code

### **unitReports**

```
unit unitReports;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls, DBCtrls, Grids, DBGrids, DB, ADODB, Mask;

type
  TfrmReports = class(TForm)

    btnGoBack: TButton;
    dtpDateModified: TDateTimePicker;
    lblMainText: TLabel;
    lblText1: TLabel;
    dbgrdReports: TDBGrid;
    dsReports: TDataSource;
    qryReports: TADOQuery;
    dbrestrtReport: TDBRichEdit;
    lblText2: TLabel;
    qryUpdate: TADOQuery;
    edtSearchFullName: TEdit;
    edtSearchID: TEdit;
```

```

// new functions have been added for improving the UI of the page through code
procedure FormShow(Sender: TObject);
procedure btnGoBackClick(Sender: TObject);
procedure dbgrdReportsDrawColumnCell(Sender: TObject;
  const Rect: TRect; DataCol: Integer; Column: TColumn;
  State: TGridDrawState);
procedure dbgrdReportsTitleClick(Column: TColumn);
procedure FormCreate(Sender: TObject);
procedure dtpDateModifiedChange(Sender: TObject);
procedure dbredtReportKeyPress(Sender: TObject; var Key: Char);
procedure edtSearchFullNameClick(Sender: TObject);
procedure edtSearchFullNameKeyPress(Sender: TObject; var Key: Char);
procedure edtSearchFullNameChange(Sender: TObject);
procedure edtSearchIDClick(Sender: TObject);
procedure edtSearchIDKeyPress(Sender: TObject; var Key: Char);
procedure edtSearchIDChange(Sender: TObject);

private
  sortByToggle : boolean;
  searchFullNamePromptHidden : boolean;
  searchIDPromptHidden : boolean;
// variables to control if the prompt is displayed inside the edit box

procedure SetSQL;
procedure HideTextPrompt(textEdit : TEdit; var hidden : boolean);
end;

var
  frmReports: TfrmReports;

implementation

uses unitHomePage, unitLogin;

{$R *.dfm}

procedure TfrmReports.FormShow(Sender: TObject);
begin
  SetSQL;
end;

// When the form is created, it sets the first column to a different blue colour, when clicking the column headers the user can sort the columns and the colour will change, by default this first column is set to a darker blue
procedure TfrmReports.FormCreate(Sender: TObject);

```

```

begin
  dbgrdReports.Columns[0].Title.Color := RGB(28, 54, 120);
end;

procedure TfrmReports.btnGoBackClick(Sender: TObject);
begin
  Hide;
  frmHomePage.Show;
end;

procedure TfrmReports.SetSQL;
var
  sqlStatement : string;
begin
  sqlStatement :=
    'SELECT reportID, firstName, lastName, reportDocument, dateModified,
  Reports.studentID ' +
    'FROM Reports INNER JOIN Students ON Reports.studentID =
  Students.studentID ' +
    'WHERE staffID = ' + frmLogin.StaffID;

  with qryReports do begin
    Close;
    SQL.Clear;
    SQL.Add(sqlStatement);
    Open;
  end;
end;

// This is an event function called when the dbgrid is refreshing, this has been
  implemented to improve the visual aspect of the table only. It alternates the
  colours of the rows to make the table more visually pleasing to the user, this was not
  possible without code

procedure TfrmReports.dbgrdReportsDrawColumnCell(Sender: TObject;
  const Rect: TRect; DataCol: Integer; Column: TColumn;
  State: TGridDrawState);
begin
  if (qryReports.RecNo MOD 2 = 0) then begin // If the row is an even number, set
    the colour to a darker grey
    if (dbgrdReports.Canvas.Brush.Color = dbgrdReports.Color) then begin
      dbgrdReports.Canvas.Brush.Color := RGB(220, 220, 220);
    end;
  end;
  dbgrdReports.DefaultDrawColumnCell(Rect, DataCol, Column, State);
end;

```

```
// This event procedure is called when the title of the column is clicked, this allows
// the user to sort the column by descending or ascending
procedure TfrmReports.dbgrdReportsTitleClick(Column: TColumn);
var
  i : integer;
begin
  for i := 0 to dbgrdReports.Columns.Count - 1 do begin
    dbgrdReports.Columns[i].Title.Color := RGB(28, 54, 72); // Sets the colour to a
    dark blue to indicate it is clickable
  end;

  for i := 0 to dbgrdReports.Columns.Count - 1 do begin
    if Column.FieldName = dbgrdReports.Columns[i].FieldName then begin
      Column.Title.Color := RGB(28, 54, 120);
      if not sortByToggle then begin // To alternate between ascending and
        sortByToggle := true;
        qryReports.Sort := dbgrdReports.Columns[i].FieldName + ' ASC';
      end
    end
    else begin
      qryReports.Sort := dbgrdReports.Columns[i].FieldName + ' DESC';
      sortByToggle := false;
    end;
  end;
end;
end;
end;
end;

procedure TfrmReports.dtpDateModifiedChange(Sender: TObject);
var
  inputDate : string;
begin
  inputDate := DateToStr(TDateTimePicker(Sender).Date);
  qryReports.Locate('dateModified', inputDate, []);
end;

// This procedure is triggered when the user updates the report by typing into it, it
// will update the date modified, if the user last edited the report a week ago, once
// they start typing it will change to today
procedure TfrmReports.dbrdtReportKeyPress(Sender: TObject; var Key: Char);
var
  sqlStatement : string;
begin
  if dbgrdReports.SelectedField.Text = '' then
```

```

exit; // Checks if the text is empty, otherwise program will crash if you try to
type

sqlStatement := ' UPDATE Reports SET dateModified = ' +
QuotedStr(DateToStr(Now())) + ' WHERE studentID = ' + // Now() function returns
the current date time
dbgrdReports.SelectedField.Text;

// Sets the sql statement string to update the date modified in the reports table

with qryUpdate do begin
  SQL.Clear;
  SQL.Add(sqlStatement);
  ExecSQL;
end;
end;

// Sets the text to empty and the colour back to black so the prompt looks as if it
has gone away when the user starts typing
procedure TfrmReports.HideTextPrompt(textEdit : TEdit; var hidden : Boolean);
begin
  if not hidden then begin
    textEdit.Text := '';
    textEdit.Font.Color := clBlack;
    hidden := true;
  end;
end;

// When clicking first name or pressing a key, it runs the Hide text prompt function
which removes the prompt from the edit box, this prompt is again a visual
improvement
procedure TfrmReports.edtSearchFullNameClick(Sender: TObject);
begin
  HideTextPrompt(TEdit(Sender), searchFullNamePromptHidden);
end;

procedure TfrmReports.edtSearchFullNameKeyPress(Sender: TObject; var Key:
Char);
begin
  HideTextPrompt(TEdit(Sender), searchFullNamePromptHidden);
end;

procedure TfrmReports.edtSearchFullNameChange(Sender: TObject);
var
  inputText : string;

```

```

searchOptions : TLocateOptions;
begin
  inputText := TEdit(Sender).Text;
  searchOptions := [loPartialKey];
  qryReports.Locate('firstName', inputText, searchOptions);
  qryReports.Locate('lastName', inputText, searchOptions);
end;

procedure TfrmReports.edtSearchIDClick(Sender: TObject);
begin
  HideTextPrompt(TEdit(Sender), searchIDPromptHidden);
end;

procedure TfrmReports.edtSearchIDKeyPress(Sender: TObject; var Key: Char);
begin
  HideTextPrompt(TEdit(Sender), searchIDPromptHidden);
end;

procedure TfrmReports.edtSearchIDChange(Sender: TObject);
var
  inputText: string;
  value : integer;
begin
  inputText := TEdit(Sender).Text;
  if (TryStrToInt(inputText, value)) then
    qryReports.Locate('studentID', value, [])
end;

end.

```

## Requirements met

This has met requirements 11, 12 and 13.

Number	Feature	Explanation and justification
11.	Each form should have a dark blue background with contrasting white text.	The software should be easy to look at and have a clear layout, by having a dark blue background it creates a good look without being distracting. During the interview with Mr. Nicolston this was outlined as a key requirement.

12.	Every title, button and edit box should be clearly labelled.	Once again the software should be easy to use, the user should understand what everything does right away, this requirement will help with this.
13.	Verdana font should be used throughout each form. Verdana size 16 bold should be used for each button and size 16 regular for each edit box. Each title should be size 48 bold.	The verdana font is a clean professional look, it will improve the UI significantly if this font is used throughout the program with consistent sizing and bold or regular formats.

## Testing

Test number	Test purpose	Test data	Expected result	Result
7.7	Check if clicking column header sorts the column from ascending to descending or descending to ascending	N/A	Data in column should be resorted from ascending or descending	Success

**7.7 Results**

The screenshot shows a software application window titled "Reports". The main title is "YOUR REPORTS" with a subtitle "(Edit Permissions Enabled)". Below the title is a table with columns: S-ID, First Name, Last Name, and Date Modified. The table contains the following data:

S-ID	First Name	Last Name	Date Modified
16	Uriah	Edwards	
18	Lacey	Townsend	
19	Kaitlin	Best	
13	Kassidy	Sims	17/01/2023
20	Abraham	Leon	17/01/2023
20	Abraham	Leon	17/01/2023
17	Jerry	Beasley	21/01/2023
12	Peter	Lara	25/02/2023
15	Amari	Mccarty	08/03/2023

At the bottom of the window, there is a search bar with the placeholder "SEARCH: 16/01/2023" and dropdown menus for "Full Name" and "Student ID". A "GO BACK" button is located in the top right corner of the title bar.

Reports X

# YOUR REPORTS

**(Edit Permissions Enabled)**

S-ID	First Name	Last Name	Date Modified
16	Uriah	Edwards	22/03/2023
19	Kaitlin	Best	21/03/2023
14	Landen	Proctor	20/03/2023
14	Landen	Proctor	20/03/2023
14	Landen	Proctor	20/03/2023
11	Landyn	Mcclain	11/03/2023
10	Piper	Mckenzie	11/03/2023
15	Amari	Mccarty	08/03/2023
12	Peter	Lara	25/02/2023

Uriah has been unsatisfactory.Uriah has been causeOfConcern in work quality.Uriah has been improvements in work completion.Uriah has been causeOfConcern in homework quality.Uriah has been improvements in homework completion.Uriah has been improvements in behaviour.Uriah has been causeOfConcern in lesson involvement.Uriah has of course started A-Levels now, they are difficult and the work will be frequent so she must keep on top of it all.Uriah has been causeOfConcern in how to improve.To conclude, Uriah has been unsatisfactory.

**SEARCH:**

**GO BACK**

## 3.9 Stage 9 Edit details page

### 3.9.1 Prototype 1

#### Form screenshot

>Edit Details - □ ×

### Edit details

First Name:  Last Name:

Email:

Address:

Phone Number:

National Insurance Number:

Username:  Password:

## Form accomplishments

The first prototype for the edit details page has now been created. This page contains a title at the top with the name of the page. On the right hand side of the page there is a go back button to return to the home page. In the centre of the page are several labelled edit boxes, each can be filled in to change a detail. These are: first name, last name, email, address, phone number, national insurance number, username and password. To the right of the edit boxes are two buttons, save and cancel. The email, address, phone number and national insurance number edit boxes are wider to allow more space for a staff member to fill in the box.

## Code

### **unitEditDetailsPage**

unit unitEditDetailsPage;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, StdCtrls, ExtCtrls, DB, ADODB;

type

TfrmEditDetails = class(TForm)

btnGoBack: TButton;  
lblMainText: TLabel;  
btnSave: TButton;  
btnCancel: TButton;  
IbledtFirstName: TLabeledEdit;  
IbledtLastName: TLabeledEdit;  
IbledtEmail: TLabeledEdit;  
IbledtAddress: TLabeledEdit;  
IbledtPhoneNumber: TLabeledEdit;  
IbledtNIN: TLabeledEdit;  
IbledtUsername: TLabeledEdit;  
IbledtPassword: TLabeledEdit;

procedure btnGoBackClick(Sender: TObject);  
procedure btnSaveClick(Sender: TObject);  
procedure btnCancelClick(Sender: TObject);  
procedure FormShow(Sender: TObject);

```
// Two private procedure have been defined, one for filling in the edit boxes, the
// edit boxes will contain the current details of the staff member logged in, the
// second is for updating the sql
private
  procedure FillEditBoxes;
  procedure UpdateSQL;

end;

var
  frmEditDetails: TfrmEditDetails;

implementation

uses unitHomePage, unitLogin;

{$R *.dfm}

// When the form is shown it fills the edit boxes so the user can see their current
// details in the edit box
procedure TfrmEditDetails.FormShow(Sender: TObject);
begin
  FillEditBoxes;
end;

// Hides this form, opens home page
procedure TfrmEditDetails.btnExitClick(Sender: TObject);
begin
  Hide;
  frmHomePage.Show;
end;

procedure TfrmEditDetails.btnSaveClick(Sender: TObject);
var
  hasChangedDetails : Boolean;
begin
  // currently with this prototype the hasChangedDetails is always true, in the next
  // prototype it will test to see if the user doesn't enter anything into the database and
  // contain a confirmation message to show their changes
  hasChangedDetails := true;

  if hasChangedDetails then begin // If they have changed the details show a
    confirmation message and then run the sql if the user selects yes
    if MessageDlg('Are you sure you want to update your details?', mtConfirmation,
```

```
[mbYes, mbNo], 0) = mrYes then begin
    UpdateSQL;
end;
end

// Otherwise it should show an error, this is a task for prototype 2
else begin
end;

end;

// When the user presses cancel, the changes they made should be reverted, by
making use of procedures this one procedure can be used for multiple things, this
procedure will again fill the edit boxes back to their current values so any changes
that are made get disregarded
procedure TfrmEditDetails.btnCancelClick(Sender: TObject);
begin
    FillEditBoxes;
end;

procedure TfrmEditDetails.FillEditBoxes;
begin

// Updates the data into the staff table and the login table
procedure TfrmEditDetails.UpdateSQL;
var
    sqlStatementStaff : string;
    sqlStatementLogin : string;
begin
    sqlStatementStaff :=
        'UPDATE Staff SET ' +
        'firstName = ' + QuotedStr(IbledtFirstName.Text) +
        ', lastName = ' + QuotedStr(IbledtLastName.Text) +
        ', email = ' + QuotedStr(IbledtEmail.Text) +
        ', address = ' + QuotedStr(IbledtAddress.Text) +
        ', phoneNumber = ' + QuotedStr(IbledtPhoneNumber.Text) +
        ', nationalInsuranceNumber = ' + QuotedStr(IbledtNIN.Text) +
        ' WHERE staffID = ' + frmLogin.StaffID;

// The purpose of this statement is to see if a username already exists, it checks
for all the fields where the username equals the one entered and where the staff ID
is not the staff ID logged in, otherwise it would obviously return that there is a user
who already exists but it's just the current user
    sqlStatementLogin := 'SELECT * FROM Login WHERE Username = ' +
        QuotedStr(IbledtUsername.Text) + ' AND staffID <> ' + frmLogin.StaffID;
```

```

with qryEditDetailsPage do begin
  Close;
  SQL.Clear;
  SQL.Add(sqlStatementStaff);
  ExecSQL;

  Close;
  SQL.Clear;
  SQL.Add(sqlStatementLogin);
  Open;
// executes this sql and if the record count is zero it can continue updating the login table
  if RecordCount = 0 then begin
    sqlStatementStaff :=
      'UPDATE Login SET ' +
      'username = ' + QuotedStr(IbledtUsername.Text) +
      ', [password] = ' + QuotedStr(IbledtPassword.Text) +
      ' WHERE staffID = ' + frmLogin.StaffID;

    with qryEditDetailsPage do begin
      Close;
      SQL.Clear;
      SQL.Add(sqlStatementStaff);
      ExecSQL;
    end;

    end
    else begin
// Should display error here but not implemented in this prototype
    end;
  end;
end;
end.

```

## Requirements met

This has met requirements 7, 26, 27, 31, 32 and 33.

Number	Feature	Explanation and justification
7.	The main program forms must follow a 16:9 aspect ratio and be smaller than the monitor	The main forms must follow the standard monitor aspect ratio, they should also be smaller than

	size.	the monitor to allow for a wider range of monitors. By doing this monitors with a 720p display or lower will still be able to run the program and view each form fully.
26.	Each page should contain a large title at the top with the name of the page.	The user should always know what page they are on, the best way of doing this is having a large title at the top of each page, this makes it clear to the user.
27.	Each page should contain a 'Go Back' button at the top right to return to the previous page.	To navigate the program quickly there should be a back button on each page, this will go to the previous page, if a user accidentally misclicks they should easily be able to go back. By having this at the same position on each page the user will become familiar with it and can use it quickly.
31.	Edit details page should contain edit boxes for first name, last name, email, address, phone number, national insurance number, username and password.	The edit details page should allow the user to change or add all possible details they can, some of these details may not be required or given during the sign up process so by having these edit boxes the user can now set all their details. This was another key requirement that came about during the interviewing.
32.	Edit details page should have a save button and a cancel button on the right size of the page.	If the user starts entering new details and changes their mind or makes a mistake they should be able to cancel to reset their changes. A save button is required so they can save their details.
33.	Program should allow the user to update their details in the edit details page.	If there is a change of details like address or surname they need to update that information on their account and the database.

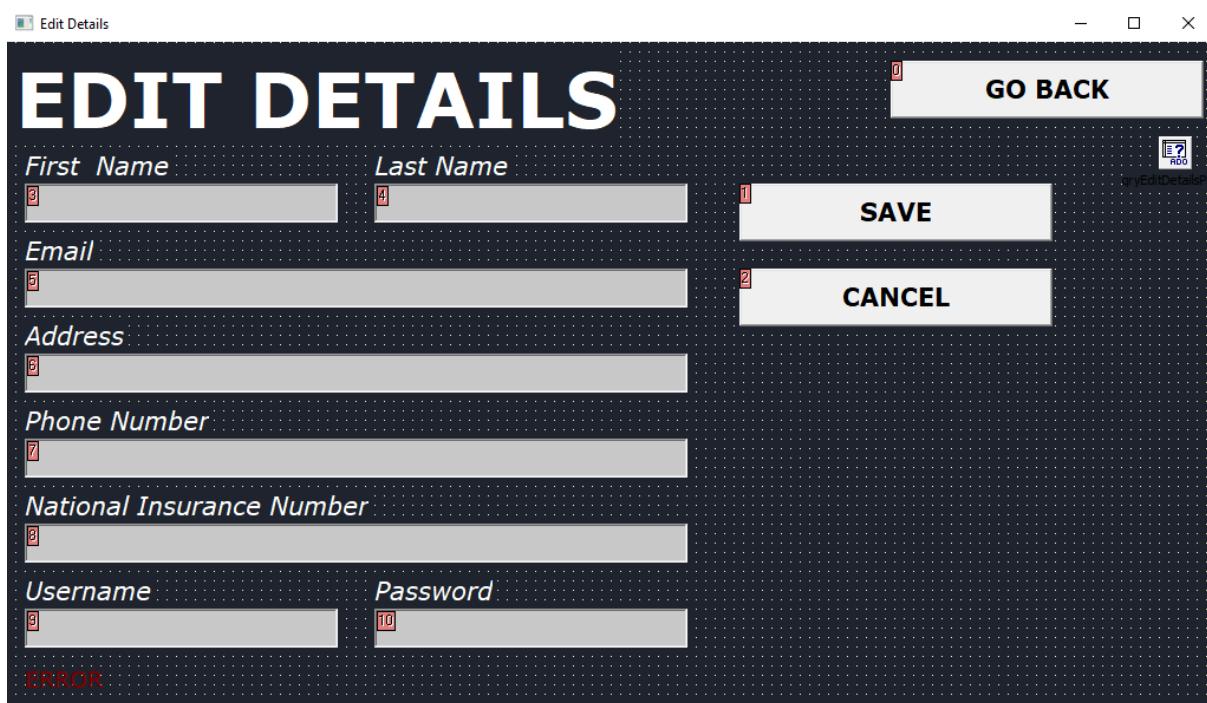
## Testing

No testing is required for this prototype, all the test plans cannot be tested with the current implementation however they will be tested with prototype 2.

## Summary

### 3.9.2 Prototype 2

#### Form screenshot



#### Form accomplishments

The second prototype for the edit details page has now been created. This contains major improvements visually and improvements with the UI in all aspects. The form now has a dark blue background colour, this is minimal and less distracting compared to the old white colour. The edit boxes, buttons and labels all use verdana font in either bold or regular, this improves the look a lot. The title, edit boxes and buttons have been resized to better fill the form size. The labels have been increased in size to improve readability. Functionality wise there is now an error message label at the bottom left.

#### Code

**unitEditDetailsPage**

unit unitEditDetailsPage;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, StdCtrls, ExtCtrls, DB, ADODB;

type

TfrmEditDetails = class(TForm)

    btnGoBack: TButton;  
    lblMainText: TLabel;  
    btnSave: TButton;  
    btnCancel: TButton;  
    IbledtFirstName: TLabeledEdit;  
    IbledtLastName: TLabeledEdit;  
    IbledtEmail: TLabeledEdit;  
    IbledtAddress: TLabeledEdit;  
    IbledtPhoneNumber: TLabeledEdit;  
    IbledtNIN: TLabeledEdit;  
    IbledtUsername: TLabeledEdit;  
    IbledtPassword: TLabeledEdit;  
    qryEditDetailsPage: TADOQuery;  
    lblErrorMessage: TLabel;

    procedure btnGoBackClick(Sender: TObject);  
    procedure btnSaveClick(Sender: TObject);  
    procedure btnCancelClick(Sender: TObject);  
    procedure FormShow(Sender: TObject);

private

    procedure FillEditBoxes;  
    procedure UpdateSQL;

end;

var

    frmEditDetails: TfrmEditDetails;

implementation

uses unitHomePage, unitLogin;

```
{$R *.dfm}

procedure TfrmEditDetails.FormShow(Sender: TObject);
begin
  FillEditBoxes;
end;

procedure TfrmEditDetails.btnGoBackClick(Sender: TObject);
begin
  Hide;
  frmHomePage.Show;
end;

// This procedure now calculates the confirmation message that will be shown
procedure TfrmEditDetails.btnSaveClick(Sender: TObject);
var
  sqlStatementStaff : string;
  sqlStatementLogin : string;
  confirmationMessage : string;
  hasChangedDetails : Boolean;
begin
  sqlStatementStaff := 'SELECT * FROM Staff WHERE staffID = ' +
  frmLogin.StaffID;
  sqlStatementLogin := 'SELECT * FROM Login WHERE staffID = ' +
  frmLogin.StaffID;

  hasChangedDetails := false;
  lblErrorMessage.Visible := False;

  with qryEditDetailsPage do begin
    Close;
    SQL.Clear;
    SQL.Add(sqlStatementStaff);
    Open;

    confirmationMessage := 'Are you sure you want to update your details? The
changes are as follows: ';

// it goes through each field, if it is different to the one in the edit box that means it
has changed, then the confirmation box shows the changes from the old to the
new so the user can see the difference
    if FieldByName('firstName').AsString <> IbledtFirstName.Text then begin
      confirmationMessage := confirmationMessage + #13'Current First Name: ' +
      FieldByName('firstName').AsString + ' --> New First Name: ' +
    end;
  end;
end;
```

```

IbledtFirstName.Text;
  hasChangedDetails := true;
end;
if FieldByName('lastName').AsString <> IbledtLastName.Text then begin
  confirmationMessage := confirmationMessage + #13'Current Last Name: ' +
FieldByName('lastName').AsString + ' --> New Last Name: ' +
IbledtLastName.Text;
  hasChangedDetails := true;
end;
if FieldByName('email').AsString <> IbledtEmail.Text then begin
  confirmationMessage := confirmationMessage + #13'Current Email: ' +
FieldByName('email').AsString + ' --> New Email: ' + IbledtEmail.Text;
  hasChangedDetails := true;
end;
if FieldByName('address').AsString <> IbledtAddress.Text then begin
  confirmationMessage := confirmationMessage + #13'Current Address: ' +
FieldByName('address').AsString + ' --> New Address: ' + IbledtAddress.Text;
  hasChangedDetails := true;
end;
if FieldByName('phoneNumber').AsString <> IbledtPhoneNumber.Text then
begin
  confirmationMessage := confirmationMessage + #13'Current Phone Number: ' +
+ FieldByName('phoneNumber').AsString + ' --> New Phone Number: ' +
IbledtPhoneNumber.Text;
  hasChangedDetails := true;
end;
if FieldByName('nationalInsuranceNumber').AsString <> IbledtNIN.Text then
begin
  confirmationMessage := confirmationMessage + #13'Current National
Insurance Number: ' + FieldByName('nationalInsuranceNumber').AsString + ' -->
New National Insurance Number: ' + IbledtNIN.Text;
  hasChangedDetails := true;
end;

Close;
SQL.Clear;
SQL.Add(sqlStatementLogin);
Open;

if FieldByName('username').AsString <> IbledtUsername.Text then begin
  confirmationMessage := confirmationMessage + #13'Current Username: ' +
FieldByName('username').AsString + ' --> New Username: ' +
IbledtUsername.Text;
  hasChangedDetails := true;
end;

```

```
if FieldByName('password').AsString <> lbledtPassword.Text then begin
    confirmationMessage := confirmationMessage + #13'Current Password: ' +
FieldByName('password').AsString + ' --> New Password: ' +
lbledtPassword.Text;
    hasChangedDetails := true;
end;

end;

if hasChangedDetails then begin
    if MessageDlg(confirmationMessage, mtConfirmation, [mbYes, mbNo], 0) =
mrYes then begin
        UpdateSQL;
    end;
end
else begin // Error message can now be displayed by setting the label to visible
    lblErrorMessage.Caption := 'You have not changed your details, cannot save';
    lblErrorMessage.Visible := true;
end;

end;

procedure TfrmEditDetails.btnCancelClick(Sender: TObject);
begin
    FillEditBoxes;
    lblErrorMessage.Visible := false;
end;

// This procedure has now been implemented fully, when the form opens this is run and fills in each edit box to show the current details
procedure TfrmEditDetails.FillEditBoxes;
var
    sqlStatementStaff : string;
    sqlStatementLogin : string;
begin
    sqlStatementStaff := 'SELECT * FROM Staff WHERE staffID = ' +
frmLogin.StaffID;
    sqlStatementLogin := 'SELECT * FROM Login WHERE staffID = ' +
frmLogin.StaffID;

    with qryEditDetailsPage do begin
        Close;
        SQL.Clear;
        SQL.Add(sqlStatementStaff);
        Open;
    end;
end;
```

```
// If there is at least one record it will fill in each edit box with the field returned
// from the query
if RecordCount <> 0 then begin
    IbledtFirstName.Text := FieldByName('firstName').AsString;
    IbledtLastName.Text := FieldByName('lastName').AsString;
    IbledtEmail.Text := FieldByName('email').AsString;
    IbledtAddress.Text := FieldByName('address').AsString;
    IbledtPhoneNumber.Text := FieldByName('phoneNumber').AsString;
    IbledtNIN.Text := FieldByName('nationalInsuranceNumber').AsString;
end;

Close;
SQL.Clear;
SQL.Add(sqlStatementLogin);
Open;

if RecordCount <> 0 then begin
    IbledtUsername.Text := FieldByName('username').AsString;
    IbledtPassword.Text := FieldByName('password').AsString;
end;
end;
end;

procedure TfrmEditDetails.UpdateSQL;
var
    sqlStatementStaff : string;
    sqlStatementLogin : string;
begin
    sqlStatementStaff :=
        'UPDATE Staff SET ' +
        'firstName = ' + QuotedStr(IbledtFirstName.Text) +
        ', lastName = ' + QuotedStr(IbledtLastName.Text) +
        ', email = ' + QuotedStr(IbledtEmail.Text) +
        ', address = ' + QuotedStr(IbledtAddress.Text) +
        ', phoneNumber = ' + QuotedStr(IbledtPhoneNumber.Text) +
        ', nationalInsuranceNumber = ' + QuotedStr(IbledtNIN.Text) +
        ' WHERE staffID = ' + frmLogin.StaffID;

    sqlStatementLogin := 'SELECT * FROM Login WHERE Username = ' +
        QuotedStr(IbledtUsername.Text) + ' AND staffID <> ' + frmLogin.StaffID;

    with qryEditDetailsPage do begin
        Close;
        SQL.Clear;
```

```

SQL.Add(sqlStatementStaff);
ExecSQL;

Close;
SQL.Clear;
SQL.Add(sqlStatementLogin);
Open;

if RecordCount = 0 then begin
  sqlStatementStaff := 
    'UPDATE Login SET ' +
    'username = ' + QuotedStr(lbledtUsername.Text) +
    ', [password] = ' + QuotedStr(lbledtPassword.Text) +
    ' WHERE staffID = ' + frmLogin.StaffID;

  with qryEditDetailsPage do begin
    Close;
    SQL.Clear;
    SQL.Add(sqlStatementStaff);
    ExecSQL;
  end;

end

else begin // Displays appropriate error message for when the username already
exists, sets the label to visible
  lblErrorMessage.Caption := 'Username already exists';
  lblErrorMessage.Visible := true;
end;
end;
end;
end.

```

## Requirements met

This has met requirements 11, 12, 13, 25 and 34.

Number	Feature	Explanation and justification
11.	Each form should have a dark blue background with contrasting white text.	The software should be easy to look at and have a clear layout, by having a dark blue background it creates a good look without being distracting. During the interview with Mr. Nicolston this was

		outlined as a key requirement.
12.	Every title, button and edit box should be clearly labelled.	Once again the software should be easy to use, the user should understand what everything does right away, this requirement will help with this.
13.	Verdana font should be used throughout each form. Verdana size 16 bold should be used for each button and size 16 regular for each edit box. Each title should be size 48 bold.	The verdana font is a clean professional look, it will improve the UI significantly if this font is used throughout the program with consistent sizing and bold or regular formats.
25.	Program should have confirmation messages throughout.	This prevents the user from accidentally changing a setting they didn't want to. For example the sign up page should confirm if they want to continue or not.
34.	Program must tell the user if their new username already exists when updating their username.	Similar to before, logging in requires a username for one specific user, so there cannot be duplicate usernames.

## Testing

Test number	Test purpose	Test data	Expected result	Result
8.1	Check if go back button works on edit details page	go back button click	Edit details page closes, home page opens	Success
8.2	Check if cancel button reverts the edit boxes to the current details	Cancel button click	Edit boxes should revert to previous data they were displaying	Success
8.3	Check if program displays error if changes have not been made	Save button click	'You have not changed your details, cannot save' error message appears.	Success
8.4	Check if confirmation box appears when user clicks save	new first name: 'test' new last name: 'test2'	Confirmation box appears, it outlines the changes that have been made which	Success

			should be what the test data is	
--	--	--	---------------------------------	--

## 8.1 Results

Home Page X

# HOME

Name: David Frost

Email: davidFrost@gmail.com

Staff ID: 2

**STUDENTS**      **STUDENT STATS**

**REPORTS**      **GENERATE REPORT**

**GRADES**      **EDIT DETAILS**

**SIGN OUT**  
**QUIT**

## 8.2 Results

Edit Details X

## EDIT DETAILS

*First Name*  *Last Name*  **GO BACK**

*Email*  **SAVE**

*Address*

*Phone Number*

*National Insurance Number*

*Username*  *Password*

Edit Details

# EDIT DETAILS

First Name  Last Name  GO BACK

Email  SAVE

Address

Phone Number

National Insurance Number

Username  Password

### 8.3 Results

Edit Details

# EDIT DETAILS

First Name  Last Name  GO BACK

Email  SAVE

Address

Phone Number

National Insurance Number

Username  Password

You have not changed your details, cannot save

### 8.4 Results

Edit Details

# EDIT DETAILS

GO BACK

First Name	Last Name	SAVE
test	test2	
Email	Confirm	
davidFrost@gmail.com	Are you sure you want to update your details? The changes are as follows: Current First Name: David --> New First Name: test Current Last Name: Frost --> New Last Name: test2	
Address	Yes No	
SW23 12345		
Phone Number		
07234567890		
National Insurance Number		
QQ 234567 C		
Username	Password	
d	d	

## 3.10 Stage 10 Generate report page

### 3.10.1 Prototype 1

#### Form screenshot

Generate Report

### Select student

Search Full Name

Student ID

Year Group

Form Group

Candidate Number

Parent email

Student email

Go Back

Continue

## Form accomplishments

The first prototype for the generate report page has now been made. This form contains a title at the top of the page. To the right of the page is a go back button so the user can go back to the home page. Below that in the bottom right a continue button has been added, this button will open the next page in the process of generating a report. In the centre of the page a large list box has been added for displaying a list of students. Below the list box there are now 6 boxes to display student information, these are: student ID, year group, form group, candidate number, parent email and student email. These boxes have been set to non-editable so the user cannot edit them because their purpose is to only display the student information.

## Code

### unitGenerateReport

```
unit unitGenerateReport;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, DB, ADODB, DBCtrls, Mask, ExtCtrls, strUtils;

type
  TfrmGenerateReportPage = class(TForm)

// Labeled edits have been defined here for all the buttons in the form
  btnGoBack: TButton;
  btnContinue: TButton;
  lblText1: TLabel;
  lblMainText: TLabel;
  lstStudents: TListBox;
  lbledtStudentID: TLabeledEdit;
  lbledtYearGroup: TLabeledEdit;
  lbledtFormGroup: TLabeledEdit;
  lblText2: TLabel;
  edtSearchFullName: TEdit;
  lbledtCandidateNumber: TLabeledEdit;
  lbledtParentEmail: TLabeledEdit;
  lbledtStudentEmail: TLabeledEdit;
```

```
procedure btnGoBackClick(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure btnContinueClick(Sender: TObject);

// A public variable called StudentID has been created, this will be used for the next form, when creating the reports the next form needs to know the student ID so it must be accessible from this class, public changes the access of this variable and now that class can see it
public
  StudentID : string;

// private procedure has been added to get the list of students
private
  procedure SetSQL;
end;

var
  frmGenerateReportPage: TfrmGenerateReportPage;
  studentIDs : array of integer; // Stores a list of the student ID's returned from the query. This is required because the staff sees a list of students and the program must know which ID the staff selects, so this array can store that value.

implementation

uses unitHomePage, unitLogin, unitReportGeneratorPage;

{$R *.dfm}

// Hides this page, opens the home page
procedure TfrmGenerateReportPage.btnGoBackClick(Sender: TObject);
begin
  Hide;
  frmHomePage.Show;
end;

// This procedure is run when the form is shown, this allows the form to reset or initialise certain data
procedure TfrmGenerateReportPage.FormShow(Sender: TObject);
begin
  lstStudents.Clear;
  SetLength(studentIDs, 0); // initialises the studentID's array to 0

  SetSQL; // runs setSQL function so that the list of students can be displayed and the studentID's array will be updated here

```

```

if Length(studentIDs) > 1 then begin // If there is at least 1 student then set the
selected student to the first index in the list displayed on the form
  lstStudents.Selected[0] := true;
end;
end;

// This procedure runs a query to get all the student names, then updates the list
of students so it can keep track of the student ID
procedure TfrmGenerateReportPage.SetSQL;
var
  sqlStatement : string;
begin
  sqlStatement := 
    'SELECT DISTINCT Students.studentID, firstName, lastName, yearGroup,
formGroup, studentEmail, parentEmail, candidateNumber ' +
    'FROM Students INNER JOIN Reports ' +
    'ON Students.studentID = Reports.studentID ' +
    'WHERE Reports.staffID = ' + frmLogin.StaffID;

  with qryStudents do begin
    Close;
    SQL.Clear;
    SQL.Add(sqlStatement);
    Open;

    First;
    while not Eof do begin // loops through result of query and adds the first and
last name to the list box items, this causes the names to display in the form
      lstStudents.Items.Add(FieldByName('firstName').AsString + ' ' +
FieldByName('lastName').AsString);
      SetLength(studentIDs, Length(studentIDs) + 1);
      studentIDs[High(studentIDs)] := FieldByName('studentID').AsInteger;
    // Updates the studentIDs array, increases the length by 1 since its a dynamic
array, then sets the current highest index to the studentID
    Next;
  end;
end;
end;

procedure TfrmGenerateReportPage.btnContinueClick(Sender: TObject);
begin
// This ensures there is at least 1 student before continuing to the next form and
closing this form
  if Length(studentIDs) > 1 then begin

```

```

    Hide;
    frmReportGeneratorPage.Show;
end;
end;

end.

```

## Code accomplishments

[this prototype does not have the update data function compared to prototype 2 but it does have setting sql]

## Requirements met

This has met requirements 7, 26, 27, 45, 46, 47 and 48.

Number	Feature	Explanation and justification
7.	The main program forms must follow a 16:9 aspect ratio and be smaller than the monitor size.	The main forms must follow the standard monitor aspect ratio, they should also be smaller than the monitor to allow for a wider range of monitors. By doing this monitors with a 720p display or lower will still be able to run the program and view each form fully.
26.	Each page should contain a large title at the top with the name of the page.	The user should always know what page they are on, the best way of doing this is having a large title at the top of each page, this makes it clear to the user.
27.	Each page should contain a 'Go Back' button at the top right to return to the previous page.	To navigate the program quickly there should be a back button on each page, this will go to the previous page, if a user accidentally misclicks they should easily be able to go back. By having this at the same position on each page the user will become familiar with it and can use it quickly.
45.	Generate report page must	The staff must be able to select a

	include a list box in the centre of the page showing a list of students.	student before continuing to the next section, having a list box in the centre of the page is the best way of displaying this information, it will contain a list of students and the staff can easily scroll through and select the one they want.
46.	Generate report page must include 6 boxes below showing student information, these will be: student ID, year group, form group, candidate number, parent email and student email	When the staff selects a student they might want to know important information on the student before creating a report, similar to the report page these fields will be displayed to easily show the staff information at a glance. They will be updated whenever they select another student.
47.	Generate report page must have a continue button at the bottom right to continue to the next page	Once the staff has selected a student they need to go to another page to create the report, a large continue button at the bottom right is a clear way to indicate to the staff how to move to the next section.
48.	Program must require the staff to select a student they want to write a report for.	Before writing the report, the staff need to tell the program which student they want to make the report for, this stage needs to be quick and simple for the staff to do so they can do multiple students.

## Testing

Test number	Test purpose	Test data	Expected result	Result
9.1	Check if go back button works on generate report page	go back button click	generate report page closes, home page opens	Success
9.2	Check if continue button opens report generator page	continue button click	generate report page closes, report generator page opens	Success

9.3	Check if continue button opens report generator page when there is no student selected	continue button click selected student: null	Next page should not open because a student has not been selected or is not in the list	Success
-----	--	---	---	---------

### 9.1 Results

Home Page

# HOME

Name: David Frost

Email: davidFrost@gmail.com

Staff ID: 2

SIGN OUT

QUIT

STUDENTS

STUDENT STATS

REPORTS

GENERATE REPORT

GRADES

EDIT DETAILS

### 9.2 Results

frmReportGenerator

### 9.3 Results

Generate Report X

## Select student

Search Full Name

Student ID  Year Group

Form Group  Candidate Number

Parent email

Student email

Go Back Continue

### Summary

## 3.10.2 Prototype 2

### Form screenshot

Generate Report - X

# SELECT STUDENT

Select the student you want to write a report for

SEARCH:  Full Name

2

Student ID	Year Group	Form Group	Candidate No.
3	4	5	7

Parent Email  Student Email

GO BACK CONTINUE

## Form accomplishments

The second prototype for the generate report form has now been created. This prototype has improved the visuals drastically to be up to standard with the other forms. Firstly additional labels have been added to the form, these include a large label below the title with a small explanation of what the user must do, this improves user experience and helps the user know the purpose of the form easily. The second label is beside the search box that says 'search' in clear text, this lets the user know it's a search box. The background of the form has been changed to a dark blue background, this improves the look. The labels, buttons and boxes have all been changed to use the verdana font in either regular or bold format. The font is much larger than before which improves readability. The list box has been aligned properly with the boxes below and the boxes below have been resized to fit the information more. The colours of the boxes and list box have been changed to a grey colour, this is less distracting and more readable.

## Code

### **unitGenerateReport**

```
unit unitGenerateReport;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, DB, ADODB, DBCtrls, Mask, ExtCtrls, strUtils;

type
  TfrmGenerateReportPage = class(TForm)

    btnGoBack: TButton;
    btnContinue: TButton;
    lblText1: TLabel;
    lblMainText: TLabel;
    qryStudents: TADOQuery;
    lstStudents: TListBox;
    lblEditStudentID: TLabeledEdit;
    lblEditYearGroup: TLabeledEdit;
    lblEditFormGroup: TLabeledEdit;
    lblText2: TLabel;
    edtSearchFullName: TEdit;
```

```
IbledtCandidateNumber: TLabeledEdit;
IbledtParentEmail: TLabeledEdit;
IbledtStudentEmail: TLabeledEdit;

procedure btnGoBackClick(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure lstStudentsClick(Sender: TObject);
procedure edtSearchFullNameChange(Sender: TObject);
procedure edtSearchFullNameClick(Sender: TObject);
procedure edtSearchFullNameKeyPress(Sender: TObject; var Key: Char);
procedure btnContinueClick(Sender: TObject);

public
  StudentID : string;

private
  searchFullNamePromptHidden : boolean;
// variable to control if the prompt is displayed inside the search box

  procedure SetSQL;
  procedure UpdateData; // new private procedure has been added, this updates the edit boxes to display the information of the student selected, this will make use of the studentID's array to know the ID of the student
    procedure HideTextPrompt(textEdit : TEdit; var hidden : boolean);
  end;

var
  frmGenerateReportPage: TfrmGenerateReportPage;
  studentIDs : array of integer;

implementation

uses unitHomePage, unitLogin, unitReportGeneratorPage;

{$R *.dfm}

procedure TfrmGenerateReportPage.btnGoBackClick(Sender: TObject);
begin
  Hide;
  frmHomePage.Show;
end;

procedure TfrmGenerateReportPage.FormShow(Sender: TObject);
begin
  lstStudents.Clear;
```

```
SetLength(studentIDs, 0);

SetSQL;

if Length(studentIDs) > 1 then begin
  IstStudents.Selected[0] := true;
  UpdateData;
end;
end;

procedure TfrmGenerateReportPage.IstStudentsClick(Sender: TObject);
begin
  UpdateData; // This event is triggered when the user clicks the list box, so it
  updates the data in the edit boxes
end;

procedure TfrmGenerateReportPage.SetSQL;
var
  sqlStatement : string;
begin
  sqlStatement := 
    'SELECT DISTINCT Students.studentID, firstName, lastName, yearGroup,
formGroup, studentEmail, parentEmail, candidateNumber ' +
    'FROM Students INNER JOIN Reports ' +
    'ON Students.studentID = Reports.studentID ' +
    'WHERE Reports.staffID = ' + frmLogin.StaffID;

  with qryStudents do begin
    Close;
    SQL.Clear;
    SQL.Add(sqlStatement);
    Open;

    First;
    while not Eof do begin
      IstStudents.Items.Add(FieldByName('firstName').AsString + ' ' +
        FieldByName('lastName').AsString);
      SetLength(studentIDs, Length(studentIDs) + 1);
      studentIDs[High(studentIDs)] := FieldByName('studentID').AsInteger;
      Next;
    end;
    end;
  end;

// New procedure UpdateData() has now been implemented completing the form
```

**functionality**

```
procedure TfrmGenerateReportPage.UpdateData;
var
  sqlStatement : string;
begin
  StudentID := IntToStr(studentIDs[IstStudents.ItemIndex]);
// Sets the global student ID variable to the current selection for the next form to use and for the ado query to use
  sqlStatement := 'SELECT * FROM Students WHERE studentID = ' + StudentID;

  with qryStudents do begin
    Close;
    SQL.Clear;
    SQL.Add(sqlStatement);
    Open;
// Runs an aqoQuery and setse each editBox to display a different field for the user to see
    IbledtStudentID.Text := StudentID;
    IbledtYearGroup.Text := FieldByName('yearGroup').AsString;
    IbledtFormGroup.Text := FieldByName('formGroup').AsString;
    IbledtCandidateNumber.Text := FieldByName('candidateNumber').AsString;
    IbledtParentEmail.Text := FieldByName('parentEmail').AsString;
    IbledtStudentEmail.Text := FieldByName('studentEmail').AsString;
  end;
end;

// When clicking first name or pressing a key, it runs the Hide text prompt function which removes the prompt from the edit box, this prompt is again a visual improvement
procedure TfrmGenerateReportPage.HideTextPrompt(textEdit : TEdit; var hidden : Boolean);
begin
  if not hidden then begin
    textEdit.Text := '';
    textEdit.Font.Color := clBlack;
    hidden := true;
  end;
end;

procedure TfrmGenerateReportPage.edtSearchFullNameChange(Sender: TObject);
var
  inputText : string;
  i : Integer;
begin
```

```

inputText := TEdit(Sender).Text;

for i := 0 to lstStudents.Items.Count - 1 do begin
  if AnsiContainsText(lstStudents.Items[i], inputText) then begin
    lstStudents.Selected[i] := true;
    UpdateData;
  end;
end;
end;

procedure TfrmGenerateReportPage.edtSearchFullNameClick(Sender: TObject);
begin
  HideTextPrompt(TEdit(Sender), searchFullNamePromptHidden);
end;

procedure TfrmGenerateReportPage.edtSearchFullNameKeyPress(Sender: TObject;
  var Key: Char);
begin
  HideTextPrompt(TEdit(Sender), searchFullNamePromptHidden);
end;

procedure TfrmGenerateReportPage.btnContinueClick(Sender: TObject);
begin
  if Length(studentIDs) > 1 then begin
    Hide;
    frmReportGeneratorPage.Show;
  end;
end;
end.

```

## Requirements met

This has met requirements 11, 12 and 13.

Number	Feature	Explanation and justification
11.	Each form should have a dark blue background with contrasting white text.	The software should be easy to look at and have a clear layout, by having a dark blue background it creates a good look without being distracting. During the interview with Mr. Nicolston this was

		outlined as a key requirement.
12.	Every title, button and edit box should be clearly labelled.	Once again the software should be easy to use, the user should understand what everything does right away, this requirement will help with this.
13.	Verdana font should be used throughout each form. Verdana size 16 bold should be used for each button and size 16 regular for each edit box. Each title should be size 48 bold.	The verdana font is a clean professional look, it will improve the UI significantly if this font is used throughout the program with consistent sizing and bold or regular formats.

## Key validation

## Testing

Test number	Test purpose	Test data	Expected result	Result
9.4	Check if search box correctly updates list box selection	search input: 'Peter'	List box selection should update and be selected on the name searched which with this test is 'Peter'	Success
9.5	Check if display boxes update to display selected student	search input: 'Amari'	display boxes should display information for Amari: student ID: 15, year group: 8	Success

### 9.4 Results

Generate Report X

# SELECT STUDENT

Select the student you want to write a report for

**SEARCH:** peter

- Piper Mckenzie
- Landyn Mcclain
- Peter Lara**
- Kassidy Sims
- Landen Proctor
- Amari Mccarty
- Uriah Edwards
- Jerry Beasley
- Lacey Townsend

Student ID	Year Group	Form Group	Candidate No.
12	13	SHP	5059

*Parent Email*  *Student Email*

**CONTINUE**

---

## 9.5 Results

Generate Report X

# SELECT STUDENT

Select the student you want to write a report for

**SEARCH:** Amari

- Kassidy Sims
- Landen Proctor
- Amari Mccarty**
- Uriah Edwards
- Jerry Beasley
- Lacey Townsend
- Kaitlin Best
- Abraham Leon

Student ID	Year Group	Form Group	Candidate No.
15	08	WJC	5062

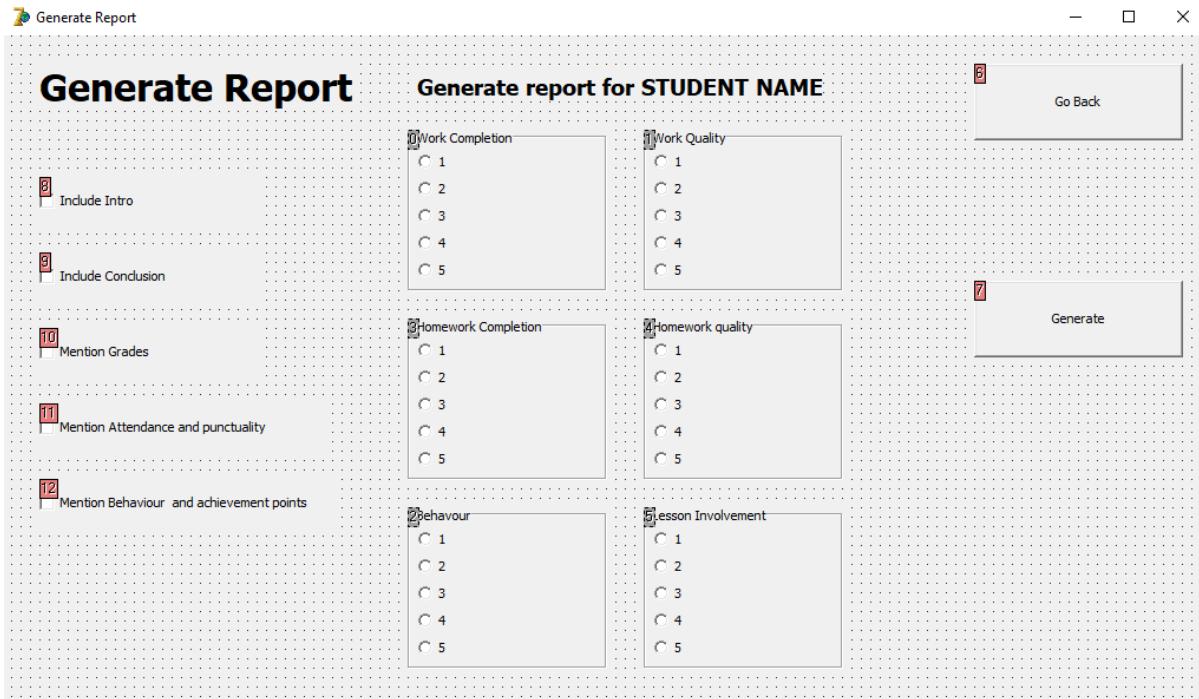
*Parent Email*  *Student Email*

**CONTINUE**

## 3.11 Stage 11 Report generator page

### 3.11.1 Prototype 1

Form screenshot



## Form accomplishments

The first prototype of the report generator page has now been made. This page includes a title at the top with the name of the form. Beside the title is a subtitle which will be updated through code to display the student name that the report is for. At the top right corner there is a go back button, below this button is a generate button which will create the report. In the centre of the page there are 6 radio groups, these are: work completion, work quality, homework completion, homework quality, behaviour and lesson involvement. These each contain radio buttons from 1 to 5 that the user can select. To the left of the form are 5 check boxes, these are: include intro, include conclusion, mention grades, mention attendance and punctuality, mention behaviour and achievement points. These can be checked on or off by the user to control what the report will contain.

## Code

### unitReportGeneratorPage

```
unit unitReportGeneratorPage;
```

```
interface
```

```
uses
```

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, StdCtrls, ExtCtrls, DB, ADODB, ComCtrls;

```
type
  TfrmReportGeneratorPage = class(TForm)
    btnGoBack: TButton;
    lblMainText: TLabel;
    lblText1: TLabel;
    grpInclude: TGroupBox;
    chkIncludeIntroduction: TCheckBox;
    chkIncludeConclusion: TCheckBox;
    chkMentionGrades: TCheckBox;
    chkMentionAttendancePunctuality: TCheckBox;
    chkMentionAchievementBehaviour: TCheckBox;
    qryStudent: TADOQuery;
    rgWorkCompletion: TRadioGroup;
    rgHomeworkCompletion: TRadioGroup;
    rgWorkQuality: TRadioGroup;
    rgLessonInvolvement: TRadioGroup;
    rgBehaviour: TRadioGroup;
    lblText2: TLabel;
    rgHomeworkQuality: TRadioGroup;
    btnGenerate: TButton;

    procedure btnGoBackClick(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure btnGenerateClick(Sender: TObject);

// Large list of private variables that will be used when creating the sentences for the report

private
  firstName : string;
  lastName : string;
  yearGroup : string;
  formGroup : string;
  gender : string;
  attendance : integer;
  punctuality : integer;
  behaviourPoints : integer;
  achievementPoints : integer;

  averageAttendance : integer;
  averagePunctuality : integer;
  averageBehaviourPoints : integer;
```

```
averageAchievementPoints : integer;

workingAtGrades : array of string;
subjects : array of string;

heShePronoun : string;
himHerPronoun : string;
hisHerPronoun : string;

// currently these are not implemented and will be in the next prototype, these integers will store the value from the radio groups that the user has inputted
workCompletion : integer;
workQuality : integer;
homeworkCompletion : integer;
homeworkQuality : integer;
behaviour : integer;
lessonInvolvement : integer;

// this variable will be the mean average of all the inputs to have a better understanding of the student overall and use that value to create sentences
overallWeight : Integer;

// Two private procedure have been created, one gets all the student information that will be used later, the second generates the report which has not been implemented in this prototype
procedure GetStudentInformation;
procedure GenerateReport;
end;

var
frmReportGeneratorPage: TfrmReportGeneratorPage;

implementation

uses unitGenerateReport, unitFinalParagraphPage;

{$R *.dfm}

// Hides this page, goes back to the generate report page, this is if the staff needs to go back and select another student
procedure TfrmReportGeneratorPage.btnGoBackClick(Sender: TObject);
begin
Hide;
frmGenerateReportPage.Show;
end;
```

```

// When the form is shown it runs this procedure to get the student information,
visually nothing changes but the variables will be stored to be used when creating
the report
procedure TfrmReportGeneratorPage.FormShow(Sender: TObject);
begin
  GetStudentInformation;
end;

procedure TfrmReportGeneratorPage.btnGenerateClick(Sender: TObject);
begin
  GenerateReport;
end;

procedure TfrmReportGeneratorPage.GetStudentInformation;
var
  sqlStatementStudent : string;
  sqlStatementPerformance : string;
  i : integer;
begin
  sqlStatementStudent := 'SELECT * FROM Students WHERE studentID = ' +
  frmGenerateReportPage.StudentID;
  sqlStatementPerformance := 'SELECT * FROM Performance WHERE studentID = ' +
  ' + frmGenerateReportPage.StudentID;
// uses the public StudentID variable from the generate report page to get the
fields from the database
  with qryStudent do begin
    Close;
    SQL.Clear;
    SQL.Add(sqlStatementStudent);
    Open;

// assigns all values
  firstName := FieldByName('firstName').AsString;
  lastName := FieldByName('lastName').AsString;
  yearGroup := FieldByName('yearGroup').AsString;
  formGroup := FieldByName('formGroup').AsString;
  gender := FieldByName('gender').AsString;

// The report must use variables to correctly word the sentences, these are
created here and will be used effectively in the next prototype
  if gender = 'M' then begin
    heShePronoun := 'he';
    himHerPronoun := 'him';

```

```
hisHerPronoun := 'his';
end
else if gender = 'F' then begin
    heShePronoun := 'she';
    himHerPronoun := 'her';
    hisHerPronoun := 'her';
end;

Close;
SQL.Clear;
SQL.Add(sqlStatementPerformance);
Open;

attendance := FieldByName('attendance').AsInteger;
punctuality := FieldByName('punctuality').AsInteger;
behaviourPoints := FieldByName('behaviourPoints').AsInteger;
achievementPoints := FieldByName('achievementPoints').AsInteger;

Close;
SQL.Clear;
SQL.Add('SELECT AVG(attendance) FROM Performance'); // Uses the AVG()
command to get the average of a whole column, then sets that to the variable
Open;

averageAttendance := Fields[0].AsInteger;

Close;
SQL.Clear;
SQL.Add('SELECT AVG(punctuality) FROM Performance');
Open;

averagePunctuality := Fields[0].AsInteger;

Close;
SQL.Clear;
SQL.Add('SELECT AVG(behaviourPoints) FROM Performance');
Open;

averageBehaviourPoints := Fields[0].AsInteger;

Close;
SQL.Clear;
SQL.Add('SELECT AVG(achievementPoints) FROM Performance');
Open;
```

```

averageAchievementPoints := Fields[0].AsInteger;

Close;
SQL.Clear;
SQL.Add('SELECT subject, workingAt FROM Grades WHERE studentID = ' +
frmGenerateReportPage.StudentID);
Open;

// The reports will look at the working at grade, there are two arrays which are for all the subjects the student takes and the working at grades in each subject

SetLength(workingAtGrades, RecordCount);
SetLength(subjects, RecordCount);

First;
i := 0;
while not Eof do begin // loops through the query result and sets the working at grade to fields[1] which is the working at grade, then sets subjectst to fields[0]
  workingAtGrades[i] := Fields[1].AsString;
  subjects[i] := Fields[0].AsString;
  i := i + 1;
  Next;
end;
end;
end;

// In the next prototype this feature will be implemented fully
procedure TfrmReportGeneratorPage.GenerateReport;
begin
end;

end.

```

## Requirements met

This has met requirements 7, 26, 27, 49, 50.

Number	Feature	Explanation and justification
7.	The main program forms must follow a 16:9 aspect ratio and be smaller than the monitor size.	The main forms must follow the standard monitor aspect ratio, they should also be smaller than the monitor to allow for a wider range of monitors. By doing this

		monitors with a 720p display or lower will still be able to run the program and view each form fully.
26.	Each page should contain a large title at the top with the name of the page.	The user should always know what page they are on, the best way of doing this is having a large title at the top of each page, this makes it clear to the user.
27.	Each page should contain a 'Go Back' button at the top right to return to the previous page.	To navigate the program quickly there should be a back button on each page, this will go to the previous page, if a user accidentally misclicks they should easily be able to go back. By having this at the same position on each page the user will become familiar with it and can use it quickly.
49.	Report generator page must contain a series of radio buttons defining how the final report will be structured. The radio button groups should be from 1-5 and the sections should be: work completion, homework completion, work quality, homework quality, behaviour, and lesson involvement.	The staff must decide how good or bad the report is for the student. By displaying several radio groups ranking from 1 to 5, the staff can choose what the final report will be like. This is crucial as it makes the reports personal and not just generic or repeated for every student.
50.	Report generator page must contain a series of checkboxes to control which section to include in the report, these will be: Include introduction, include conclusion, mention grades, mention attendance and punctuality, mention behaviour and achievement points.	The staff should be able to control the outcome of the report, these check boxes allow the staff to pick what will be included. If the staff don't want a conclusion they can simply untick it. This was requested by Mr. Nicolston during the interview we had, as a key requirement.

## Testing

Test number	Test purpose	Test data	Expected result	Result

10.1	Check if go back button works on report generator page	go back button click	report generator page closes, generate report page opens	Success
10.2	Check if student name is displayed at the top of the page	student name selected: 'Peter Lara'	'Generate report for Peter Lara' should be displayed at top of form	Success

### 10.1 Results

Generate Report X

# SELECT STUDENT

Select the student you want to write a report for

**SEARCH:**

Piper Mckenzie  
Landyn Mcclain  
**Peter Lara**  
Kassidy Sims  
Landen Proctor  
Amari Mccarty  
Uriah Edwards  
Jerry Beasley  
Lacey Townsend

Student ID	Year Group	Form Group	Candidate No.
12	13	SHP	5059

*Parent Email*  *Student Email*  **CONTINUE**

### 10.2 Results

Generate Report

## Generate Report

**Generate report for Peter Lara**

Include Intro

Include Conclusion

Mention Grades

Mention Attendance and punctuality

Mention Behaviour and achievement points

Work Completion  
 1  
 2  
 3  
 4  
 5

Work Quality  
 1  
 2  
 3  
 4  
 5

Homework Completion  
 1  
 2  
 3  
 4  
 5

Homework quality  
 1  
 2  
 3  
 4  
 5

Behaviour  
 1  
 2  
 3  
 4  
 5

Lesson Involvement  
 1  
 2  
 3  
 4  
 5

[Go Back](#)

[Generate](#)

## Summary

### 3.11.2 Prototype 2

#### Form screenshot

Generate Report

# GENERATE REPORT

Generate report for STUDENT NAME

(1 – Major Cause Of Concern 5 – Excellent)

**Include Sections**

- 0 Include Introduction
- 1 Include Conclusion
- 2 Mention Grades
- 3 Mention Attendance And Punctuality
- 4 Mention Behaviour And Achievement Points

Work Completion <input checked="" type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5	Work Quality <input checked="" type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5
Homework Completion <input checked="" type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5	Homework Quality <input checked="" type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5
Behaviour <input checked="" type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5	Lesson Involvement <input checked="" type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5

**ERROR**

[GO BACK](#)

[GENERATE](#)

## Form accomplishments

The second prototype for the report generator page has now been added, this prototype has improved the UI and overall design by a large amount. The generate button has moved to the bottom right to match the previous forms layout, it also makes it clearer to the user what they need to do to continue. The sub title text has been positioned below the title text, this improves the look of the form and is easier to read. The radio buttons now show horizontally instead of vertically, this is easier to look at and easier to use. The checkboxes are now grouped and a title has been added to them so the user knows what the purpose of the checkboxes are. A new label has been added to help the user understand the radio buttons, this title explains that 1 is the worst option and 5 is the best option. Having that clears confusion and improves the user experience. By default the check boxes are now ticked as often the staff will want to include each and they won't have to click each time. An error label message has been added, an ado query has been added. The form now uses a dark blue background, this is more visually appealing. The font is now in verdana across the whole form, buttons have been resized and rescaled to fit the size of the form better and improve readability.

## Code

### unitReportGeneratorPage

```
unit unitReportGeneratorPage;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, StdCtrls, ExtCtrls, DB, ADODB, ComCtrls;
```

```
// A new class has been created for storing the sentences, there will be many  
different sentences for the different rankings so by using objects of type  
TSentences each object can have these 5 arrays of string to rank the student, this  
reduces the number of variables that would have to be made drastically as each  
category would need 5 variables for each ranking
```

```
// This also drastically reduces the code required by making use of functions, the  
get array function has been defined below which will take in a value from the user  
which is the ranking, it will then randomly select a sentence from the 5 arrays that  
matches with the value passed in. Without using object oriented programming and  
defining this class this would have been tricky and less simple
```

```
type
TSentences = class
public
  excellent : array of string;
  good : array of string;
  improvements : array of string;
  unsatisfactory : array of string;
  causeOfConcern : array of string;

  function GetArray(value : integer) : string;
end;

type
TfrmReportGeneratorPage = class(TForm)

  btnGoBack: TButton;
  lblMainText: TLabel;
  lblText1: TLabel;
  grpInclude: TGroupBox;
  chkIncludeIntroduction: TCheckBox;
  chkIncludeConclusion: TCheckBox;
  chkMentionGrades: TCheckBox;
  chkMentionAttendancePunctuality: TCheckBox;
  chkMentionAchievementBehaviour: TCheckBox;
  qryStudent: TADOQuery;
  rgWorkCompletion: TRadioGroup;
  rgHomeworkCompletion: TRadioGroup;
  rgWorkQuality: TRadioGroup;
  rgLessonInvolvement: TRadioGroup;
  rgBehaviour: TRadioGroup;
  lblText2: TLabel;
  rgHomeworkQuality: TRadioGroup;
  btnGenerate: TButton;
  lblErrorMessage: TLabel;

procedure btnGoBackClick(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure btnGenerateClick(Sender: TObject);

public
  procedure RegenerateReport; // A public procedure has been created for the
                           // next form to make use of, this is public because the other form must access the
                           // functionality of this form to create the report so it will call this function

private
```

```
firstName : string;
lastName : string;
yearGroup : string;
formGroup : string;
gender : string;
attendance : integer;
punctuality : integer;
behaviourPoints : integer;
achievementPoints : integer;

averageAttendance : integer;
averagePunctuality : integer;
averageBehaviourPoints : integer;
averageAchievementPoints : integer;

workingAtGrades : array of string;
subjects : array of string;

heShePronoun : string;
himHerPronoun : string;
hisHerPronoun : string;

workCompletion : integer;
workQuality : integer;
homeworkCompletion : integer;
homeworkQuality : integer;
behaviour : integer;
lessonInvolvement : integer;

overallWeight : Integer;

finalParagraph : string;
```

*// New procedures have been implemented for creating the sentences for the reports*

```
procedure SetAllSentences;
procedure GetStudentInformation;
procedure Grades;
procedure AttendancePunctuality;
procedure BehaviourAchievementPoints;
procedure SchoolYear;
procedure GenerateReport;
end;
```

```
var
```

```
frmReportGeneratorPage: TfrmReportGeneratorPage;  
  
implementation  
  
uses unitGenerateReport, unitFinalParagraphPage;  
  
{$R *.dfm}  
  
// Several different objects have been defined here each of type TSentences, for example introduction will have excellent, good.. and so on arrays of sentences, by creating a type I only need to store 5 arrays that each object will have using the dot function. Each of these variables is an instance of the TSentences class  
var  
    introductionSentences : TSentences;  
    conclusionSentences : TSentences;  
    workQualitySentences : TSentences;  
    workCompletionSentences : TSentences;  
    homeworkQualitySentences : TSentences;  
    homeworkCompletionSentences : TSentences;  
    behaviourSentences : TSentences;  
    lessonInvolvementSentences : TSentences;  
    HowToImproveSentences : TSentences;  
  
// The GetArray() function has been implemented here, since this function is a member of the TSentences class it has access to the 5 array variables, when an object calls this function these variables belong to that object  
function TSentences.GetArray(value : integer) : string;  
begin  
    Randomize;  
    // If the value entered by the user is 1, then it returns a random string from the causeOfConcern array, so it has returned a sentence to match the value of 1 but it has been randomised to make the report unique and varied, same processed repeated for each value  
    if value = 1 then begin  
        GetArray := causeOfConcern[Random(length(causeOfConcern))];  
    end  
    else if value = 2 then begin  
        GetArray := unsatisfactory[Random(length(unsatisfactory))];  
    end  
    else if value = 3 then begin  
        GetArray := improvements[Random(length(improvements))];  
    end  
    else if value = 4 then begin  
        GetArray := good[Random(length(good))];  
    end
```

```
else if value = 5 then begin
  GetArray := excellent[Random(length(excellent))];
end;
end;

procedure TfrmReportGeneratorPage.btnGoBackClick(Sender: TObject);
begin
  Hide;
  frmGenerateReportPage.Show;
end;

// When the form is shown it displays the student name so the user knows which student it is for
procedure TfrmReportGeneratorPage.FormShow(Sender: TObject);
begin
  GetStudentInformation;
  lblText1.Caption := 'Generate report for ' + firstName + ' ' + lastName;
  lblErrorMessage.Visible := false;
end;

procedure TfrmReportGeneratorPage.btnGenerateClick(Sender: TObject);
begin
  GenerateReport;
end;

procedure TfrmReportGeneratorPage.GetStudentInformation;
var
  sqlStatementStudent : string;
  sqlStatementPerformance : string;
  i : integer;
begin
  sqlStatementStudent := 'SELECT * FROM Students WHERE studentID = ' +
frmGenerateReportPage.StudentID;
  sqlStatementPerformance := 'SELECT * FROM Performance WHERE studentID =
' + frmGenerateReportPage.StudentID;

  with qryStudent do begin
    Close;
    SQL.Clear;
    SQL.Add(sqlStatementStudent);
    Open;
  end;

  firstName := FieldByName('firstName').AsString;
  lastName := FieldByName('lastName').AsString;
```

```
yearGroup := FieldByName('yearGroup').AsString;
formGroup := FieldByName('formGroup').AsString;
gender := FieldByName('gender').AsString;

if gender = 'M' then begin
    heShePronoun := 'he';
    himHerPronoun := 'him';
    hisHerPronoun := 'his';
end
else if gender = 'F' then begin
    heShePronoun := 'she';
    himHerPronoun := 'her';
    hisHerPronoun := 'her';
end;

Close;
SQL.Clear;
SQL.Add(sqlStatementPerformance);
Open;

attendance := FieldByName('attendance').AsInteger;
punctuality := FieldByName('punctuality').AsInteger;
behaviourPoints := FieldByName('behaviourPoints').AsInteger;
achievementPoints := FieldByName('achievementPoints').AsInteger;

Close;
SQL.Clear;
SQL.Add('SELECT AVG(attendance) FROM Performance');
Open;

averageAttendance := Fields[0].AsInteger;

Close;
SQL.Clear;
SQL.Add('SELECT AVG(punctuality) FROM Performance');
Open;

averagePunctuality := Fields[0].AsInteger;

Close;
SQL.Clear;
SQL.Add('SELECT AVG(behaviourPoints) FROM Performance');
Open;

averageBehaviourPoints := Fields[0].AsInteger;
```

```

Close;
SQL.Clear;
SQL.Add('SELECT AVG(achievementPoints) FROM Performance');
Open;

averageAchievementPoints := Fields[0].AsInteger;

Close;
SQL.Clear;
SQL.Add('SELECT subject, workingAt FROM Grades WHERE studentID = ' +
frmGenerateReportPage.StudentID);
Open;

SetLength(workingAtGrades, RecordCount);
SetLength(subjects, RecordCount);

First;
i := 0;
while not Eof do begin
  workingAtGrades[i] := Fields[1].AsString;
  subjects[i] := Fields[0].AsString;
  i := i + 1;
  Next;
end;
end;
end;

// This procedure sets all the sentences, it creates the instances of the TSentences class and sets the arrays, this is where the actual sentences of the report are created, it uses the variables that have been created earlier to write the report include the name of the student and anything else
procedure TfrmReportGeneratorPage.SetAllSentences;
begin
  introductionSentences := TSentences.Create;

  SetLength(introductionSentences.excellent, 2);
  introductionSentences.excellent[0] := firstName + ' is an excellent student.';
  introductionSentences.excellent[1] := firstName + ' is an excellent student, ' +
heShePronoun + ' is always interested in ' + hisHerPronoun + ' work and works very hard.';

  SetLength(introductionSentences.good, 2);
  introductionSentences.good[0] := firstName + ' has been good.';
  introductionSentences.good[1] := firstName + ' has been good.';

```

```
SetLength(introductionSentences.improvements, 2);
introductionSentences.improvements[0] := firstName + ' has room for
improvement.';
introductionSentences.improvements[1] := firstName + ' has room for
improvement.';

SetLength(introductionSentences.unsatisfactory, 2);
introductionSentences.unsatisfactory[0] := firstName + ' has been
unsatisfactory.';
introductionSentences.unsatisfactory[1] := firstName + ' has been
unsatisfactory.';

SetLength(introductionSentences.causeOfConcern, 2);
introductionSentences.causeOfConcern[0] := firstName + ' has been a cause of
concern.';
introductionSentences.causeOfConcern[1] := firstName + ' has been a cause of
concern.';

conclusionSentences := TSentences.Create;

SetLength(conclusionSentences.excellent, 2);
conclusionSentences.excellent[0] := 'To conclude, ' + firstName + ' has been
excellent.';
conclusionSentences.excellent[1] := 'To conclude, ' + firstName + ' has been
excellent.';

SetLength(conclusionSentences.good, 2);
conclusionSentences.good[0] := 'To conclude, ' + firstName + ' has been good.';
conclusionSentences.good[1] := 'To conclude, ' + firstName + ' has been good.';

SetLength(conclusionSentences.improvements, 2);
conclusionSentences.improvements[0] := 'To conclude, ' + firstName + ' has
been improvements.';
conclusionSentences.improvements[1] := 'To conclude, ' + firstName + ' has
been improvements.';

SetLength(conclusionSentences.unsatisfactory, 2);
conclusionSentences.unsatisfactory[0] := 'To conclude, ' + firstName + ' has
been unsatisfactory.';
conclusionSentences.unsatisfactory[1] := 'To conclude, ' + firstName + ' has
been unsatisfactory.';

SetLength(conclusionSentences.causeOfConcern, 2);
conclusionSentences.causeOfConcern[0] := 'To conclude, ' + firstName + ' has
```

been cause of concern.';

conclusionSentences.causeOfConcern[1] := 'To conclude, ' + firstName + ' has been cause of concern.';

workQualitySentences := TSentences.Create;

SetLength(workQualitySentences.excellent, 2);

workQualitySentences.excellent[0] := firstName + ' is a highly capable student who has performed consistently well since the beginning of term. The quality of ' + hisHerPronoun + ' work has been nothing short of excellent, ' + heShePronoun + ' shows a clear understanding of the concepts being taught and it''s great to see this.';

workQualitySentences.excellent[1] := firstName + ' is a hard working student, ' + heShePronoun + ' works well with friends in class and picks up new concepts and skills quickly, applying them confidently to exam questions. The quality of ' + hisHerPronoun + ' work is to a high standard which shows ' + firstName + ' is paying attention in class and working correctly.';

SetLength(workQualitySentences.good, 2);

workQualitySentences.good[0] := 'The work I''ve seen from ' + firstName + ' has been good. They write good notes and when it''s time to answer questions they show a great understanding.';

workQualitySentences.good[1] := firstName + ' works hard and does well, ' + heShePronoun + ' has a really good understanding of what''s begin taught and ' + heShePronoun + ' can get to the challenging questions quickly.';

SetLength(workQualitySentences.improvements, 2);

workQualitySentences.improvements[0] := 'The quality of work from ' + firstName + ' has been okay, however I feel there is room for improvement. I know if ' + firstName + ' pushes further ' + hisHerPronoun + ' ability to answer questions will improve and ' + hisHerPronoun + ' work will improve greatly.';

workQualitySentences.improvements[1] := firstName + ' needs to make some adjustments to ' + hisHerPronoun + ' work, I think ' + heShePronoun + ' has the capability of becoming a top student if ' + heShePronoun + ' spends more time revising and practising questions.';

SetLength(workQualitySentences.unsatisfactory, 2);

workQualitySentences.unsatisfactory[0] := 'The quality of work from ' + firstName + ' has been unsatisfactory, ' + heShePronoun + ' is not working as hard as ' + heShePronoun + ' could be, this is a problem and if ' + heShePronoun + ' is struggling remember ' + heShePronoun + ' can always come and talk to me for work advice.';

workQualitySentences.unsatisfactory[1] := firstName + ' has not been working well at all, ' + hisHerPronoun + ' understanding is weak and ' + heShePronoun + ' is not putting much time into learning and improving.';

SetLength(workQualitySentences.causeOfConcern, 2);  
workQualitySentences.causeOfConcern[0] := 'Painfully poor work from ' +  
firstName + '. The standard is very poor and not acceptable at all, ' +  
heShePronoun + ' really needs to get a grip and start focusing in lesson, making  
better notes and answering questions properly. This is really concerning and I am  
not impressed.';

workQualitySentences.causeOfConcern[1] := 'The quality of work from ' +  
firstName + ' has been a major cause of concern, it is really worrying seeing such  
poor understanding of the topics and level of answering questions. Something  
needs to change fast or it is not looking good for ' + firstName + ' ';

workCompletionSentences := TSentences.Create;

SetLength(workCompletionSentences.excellent, 2);

workCompletionSentences.excellent[0] := firstName + ' has excellent completion  
of work, ' + heShePronoun + ' completes all ' + hisHerPronoun + ' work in lesson  
and keeps up with what is being taught.';

workCompletionSentences.excellent[1] := 'The completion of work has been  
amazing, ' + firstName + ' always finishes the work I hand out and hardly ever  
needs help.';

SetLength(workCompletionSentences.good, 2);

workCompletionSentences.good[0] := firstName + ' has good completion of  
work. Any work that needs to be done during lesson gets done and I am happy  
with the amount ' + heShePronoun + ' completes in a lesson.';

workCompletionSentences.good[1] := 'Work has been finished every lesson  
which is great to see, unlike many students ' + firstName + ' does not sit around  
doing nothing and actually gets on with all ' + hisHerPronoun + ' work.';

SetLength(workCompletionSentences.improvements, 2);

workCompletionSentences.improvements[0] := firstName + ' has been  
improvements in work completion.';

workCompletionSentences.improvements[1] := firstName + ' has been  
improvements in work completion.';

SetLength(workCompletionSentences.unsatisfactory, 2);

workCompletionSentences.unsatisfactory[0] := firstName + ' has been  
unsatisfactory in work completion.';

workCompletionSentences.unsatisfactory[1] := firstName + ' has been  
unsatisfactory in work completion.';

SetLength(workCompletionSentences.causeOfConcern, 2);

workCompletionSentences.causeOfConcern[0] := firstName + ' has been  
causeOfConcern in work completion.';

```
workCompletionSentences.causeOfConcern[1] := firstName + ' has been
causeOfConcern in work completion.';

homeworkQualitySentences := TSentences.Create;

SetLength(homeworkQualitySentences.excellent, 2);
homeworkQualitySentences.excellent[0] := firstName + ' has been excellent in
homework quality.';
homeworkQualitySentences.excellent[1] := firstName + ' has been excellent in
homework quality.';

SetLength(homeworkQualitySentences.good, 2);
homeworkQualitySentences.good[0] := firstName + ' has been good in
homework quality.';
homeworkQualitySentences.good[1] := firstName + ' has been good in
homework quality.';

SetLength(homeworkQualitySentences.improvements, 2);
homeworkQualitySentences.improvements[0] := firstName + ' has been
improvements in homework quality.';
homeworkQualitySentences.improvements[1] := firstName + ' has been
improvements in homework quality.';

SetLength(homeworkQualitySentences.unsatisfactory, 2);
homeworkQualitySentences.unsatisfactory[0] := firstName + ' has been
unsatisfactory in homework quality.';
homeworkQualitySentences.unsatisfactory[1] := firstName + ' has been
unsatisfactory in homework quality.';

SetLength(homeworkQualitySentences.causeOfConcern, 2);
homeworkQualitySentences.causeOfConcern[0] := firstName + ' has been
causeOfConcern in homework quality.';
homeworkQualitySentences.causeOfConcern[1] := firstName + ' has been
causeOfConcern in homework quality.';

homeworkCompletionSentences := TSentences.Create;

SetLength(homeworkCompletionSentences.excellent, 2);
homeworkCompletionSentences.excellent[0] := firstName + ' has been excellent
in homework completion.';
homeworkCompletionSentences.excellent[1] := firstName + ' has been excellent
in homework completion.';

SetLength(homeworkCompletionSentences.good, 2);
homeworkCompletionSentences.good[0] := firstName + ' has been good in
```

```
homework completion.';

homeworkCompletionSentences.good[1] := firstName + ' has been good in
homework completion.';

SetLength(homeworkCompletionSentences.improvements, 2);
homeworkCompletionSentences.improvements[0] := firstName + ' has been
improvements in homework completion.';

homeworkCompletionSentences.improvements[1] := firstName + ' has been
improvements in homework completion.';

SetLength(homeworkCompletionSentences.unsatisfactory, 2);
homeworkCompletionSentences.unsatisfactory[0] := firstName + ' has been
unsatisfactory in homework completion.';

homeworkCompletionSentences.unsatisfactory[1] := firstName + ' has been
unsatisfactory in homework completion.';

SetLength(homeworkCompletionSentences.causeOfConcern, 2);
homeworkCompletionSentences.causeOfConcern[0] := firstName + ' has been
causeOfConcern in homework completion.';

homeworkCompletionSentences.causeOfConcern[1] := firstName + ' has been
causeOfConcern in homework completion.';

behaviourSentences := TSentences.Create;

SetLength(behaviourSentences.excellent, 2);
behaviourSentences.excellent[0] := firstName + ' has been excellent in
behaviour.';

behaviourSentences.excellent[1] := firstName + ' has been excellent in
behaviour.';

SetLength(behaviourSentences.good, 2);
behaviourSentences.good[0] := firstName + ' has been good in behaviour.';

behaviourSentences.good[1] := firstName + ' has been good in behaviour.';

SetLength(behaviourSentences.improvements, 2);
behaviourSentences.improvements[0] := firstName + ' has been improvements in
behaviour.';

behaviourSentences.improvements[1] := firstName + ' has been improvements in
behaviour.';

SetLength(behaviourSentences.unsatisfactory, 2);
behaviourSentences.unsatisfactory[0] := firstName + ' has been unsatisfactory in
behaviour.';

behaviourSentences.unsatisfactory[1] := firstName + ' has been unsatisfactory in
behaviour.';
```

```

SetLength(behaviourSentences.causeOfConcern, 2);
behaviourSentences.causeOfConcern[0] := firstName + ' has been
causeOfConcern in behaviour.';
behaviourSentences.causeOfConcern[1] := firstName + ' has been
causeOfConcern in behaviour.';

lessonInvolvementSentences := TSentences.Create;

SetLength(lessonInvolvementSentences.excellent, 2);
lessonInvolvementSentences.excellent[0] := firstName + ' has been excellent in
lesson involvement.';
lessonInvolvementSentences.excellent[1] := firstName + ' has been excellent in
lesson involvement.';

SetLength(lessonInvolvementSentences.good, 2);
lessonInvolvementSentences.good[0] := firstName + ' has been good in lesson
involvement.';
lessonInvolvementSentences.good[1] := firstName + ' has been good in lesson
involvement.';

SetLength(lessonInvolvementSentences.improvements, 2);
lessonInvolvementSentences.improvements[0] := firstName + ' has been
improvements in lesson involvement.';
lessonInvolvementSentences.improvements[1] := firstName + ' has been
improvements in lesson involvement.';

SetLength(lessonInvolvementSentences.unsatisfactory, 2);
lessonInvolvementSentences.unsatisfactory[0] := firstName + ' has been
unsatisfactory in lesson involvement.';
lessonInvolvementSentences.unsatisfactory[1] := firstName + ' has been
unsatisfactory in lesson involvement.';

SetLength(lessonInvolvementSentences.causeOfConcern, 2);
lessonInvolvementSentences.causeOfConcern[0] := firstName + ' has been
causeOfConcern in lesson involvement.';
lessonInvolvementSentences.causeOfConcern[1] := firstName + ' has been
causeOfConcern in lesson involvement.';

HowToImproveSentences := TSentences.Create;

SetLength(HowToImproveSentences.excellent, 2);
HowToImproveSentences.excellent[0] := firstName + ' needs to continue their
excellent revision and work ethic.';
HowToImproveSentences.excellent[1] := firstName + ' needs to continue their

```

```

excellent revision and work ethic.';

SetLength(HowToImproveSentences.good, 2);
HowToImproveSentences.good[0] := firstName + ' needs to use the resources
available to them to continue revising.';
HowToImproveSentences.good[1] := firstName + ' needs to use the resources
available to them to continue revising.';

SetLength(HowToImproveSentences.improvements, 2);
HowToImproveSentences.improvements[0] := 'I highly suggest ' + firstName +
uses the revision resources available as it will help them greatly.';
HowToImproveSentences.improvements[1] := 'I highly suggest ' + firstName +
uses the revision resources available as it will help them greatly.';

SetLength(HowToImproveSentences.unsatisfactory, 2);
HowToImproveSentences.unsatisfactory[0] := firstName + ' has been
unsatisfactory in how to improve.';
HowToImproveSentences.unsatisfactory[1] := firstName + ' has been
unsatisfactory in how to improve.';

SetLength(HowToImproveSentences.causeOfConcern, 2);
HowToImproveSentences.causeOfConcern[0] := firstName + ' has been
causeOfConcern in how to improve.';
HowToImproveSentences.causeOfConcern[1] := firstName + ' has been
causeOfConcern in how to improve.';

end;

// The grades procedure creates a message based on the grades the student has
procedure TfrmReportGeneratorPage.Grades;
var i : integer;
begin // Loops through the grades array and then adds on to the final paragraph if
the grade is good or bad
for i := 0 to length(workingAtGrades) - 1 do begin
  finalParagraph := finalParagraph + firstName + ' is working at ' +
  workingAtGrades[i] + ' in ' + subjects[i] + ' .';
  if (workingAtGrades[i] = 'A*') or (workingAtGrades[i] = 'A') then begin
    finalParagraph := finalParagraph + workingAtGrades[i] + ' is a really good
grade. ';
  end;
  if (workingAtGrades[i] = 'B') then begin
    finalParagraph := finalParagraph + workingAtGrades[i] + ' is a solid grade. ';
  end;

  if (workingAtGrades[i] = 'C') or (workingAtGrades[i] = 'D') or (workingAtGrades[i] =
  'E') then begin

```

```

finalParagraph := finalParagraph + workingAtGrades[i] + ' is a poor grade and '
+ heShePronoun + ' should try to improve this. ';
end;
end;
end;

// This procedure creates a message based on their attendance and punctuality, it
looks at the average attendance calculated earlier and compares which is greater
or smaller. It then adds to the final paragraph an appropriate message.
procedure TfrmReportGeneratorPage.AttendancePunctuality;
begin
  finalParagraph := finalParagraph + 'This year ' + firstName + ' has reached an
attendance of ' + IntToStr(attendance) + '%.';

  if attendance > averageAttendance then begin
    finalParagraph := finalParagraph + ' The average attendace is currently ' +
IntToStr(averageAttendance) + '% so ' + heShePronoun + ' is above average and
doing well.';
  end;

  if attendance = averageAttendance then begin
    finalParagraph := finalParagraph + ' The average attendace is currently ' +
IntToStr(averageAttendance) + '% so ' + heShePronoun + ' has equal attendance
which is good.';
  end;

  if attendance < averageAttendance then begin
    finalParagraph := finalParagraph + ' The average attendace is currently ' +
IntToStr(averageAttendance) + '% so ' + heShePronoun + ' is below average, this
is concerning and ' + heShePronoun + ' needs to make sure ' + heShePronoun + '
turns up more regularly./';
  end;
end;

// Same process is carried out for this page
procedure TfrmReportGeneratorPage.BehaviourAchievementPoints;
begin
  finalParagraph := finalParagraph + firstName + ' currently has ' +
IntToStr(achievementPoints) + ' achievement points and ' +
IntToStr(behaviourPoints) + ' behaviour points.';

  if achievementPoints > averageAchievementPoints then begin
    finalParagraph := finalParagraph + ' The average amount of achievement points
is currently ' + IntToStr(averageAchievementPoints) + ' so I''m happy to report ' +
firstName + ' is above average.'

```

```

end;

if achievementPoints = averageAchievementPoints then begin
    finalParagraph := finalParagraph + ' The average amount of achievement points
is currently ' + IntToStr(averageAchievementPoints) + ' so ' + firstName + ' has
exactly the average.'
end;

if achievementPoints < averageAchievementPoints then begin
    finalParagraph := finalParagraph + ' The average amount of achievement points
is currently ' + IntToStr(averageAchievementPoints) + ' which means ' + firstName
+ ' is below average and should try and improve this.'
end;

if behaviourPoints > averageBehaviourPoints then begin
    finalParagraph := finalParagraph + ' Furthermore, the average amount of
behaviour points is currently ' + IntToStr(averageBehaviourPoints) + ' so I'm
happy to report ' + firstName + ' is above average.'
end;

if behaviourPoints = averageBehaviourPoints then begin
    finalParagraph := finalParagraph + ' Furthermore, the average amount of
behaviour points is currently ' + IntToStr(averageBehaviourPoints) + ' so ' +
firstName + ' has exactly the average.'
end;

if behaviourPoints < averageBehaviourPoints then begin
    finalParagraph := finalParagraph + ' Furthermore, the average amount of
behaviour points is currently ' + IntToStr(averageBehaviourPoints) + ' which
means ' + firstName + ' is below average and should aim to increase this.'
end;
end;

// This looks at the current school year the student is in and then randomly selects
a sentence that talks about the school year, it then appends it to the final
paragraph.

procedure TfrmReportGeneratorPage.SchoolYear;
var
    year7Sentences : array of string;
    year8Sentences : array of string;
    year9Sentences : array of string;
    year10Sentences : array of string;
    year11Sentences : array of string;
    year12Sentences : array of string;
    year13Sentences : array of string;

```

begin

Randomize;

SetLength(year7Sentences, 2);

year7Sentences[0] := 'Settling into year 7 can be a challenge, the increased level of work and large number of new students can be intimidating. This is why I''m happy to say ' + firstName + ' has settled in extremely well.';

year7Sentences[1] := 'As this is the first year of secondary school for ' + firstName + ', it can be difficult and there is a lot to get used to. ' + firstName + ' has began the year exceedingly well which is great to see.';

SetLength(year8Sentences, 2);

year8Sentences[0] := firstName + ' is of course well into year 8, the level of work remains similar and ' + hisHerPronoun + ' classes remain the same.';

year8Sentences[1] := 'Year 8 students have a slight increase in the difficult of work so ' + firstName + ' needs to be aware of this. They must also learn to be good role models for the year 7''s.';

SetLength(year9Sentences, 2);

year9Sentences[0] := 'It is important to do well at this stage, with GCSE''s beginning next year they should focus in lessons and it will make revising far easier when the real exams come.';

year9Sentences[1] := 'Being in year 9, ' + heShePronoun + ' needs to be a good role model for the younger years. Also, GCSE''s are starting next year and the work they do now will come in handy later on.';

SetLength(year10Sentences, 2);

year10Sentences[0] := 'These two years happen very quickly, if students don''t revise consistently now they will struggle next year, I suggest ' + firstName + ' revises early to ensure a good GCSE performance.';

year10Sentences[1] := firstName + ' has now started their GCSE''s, the difficulty in the work increases significantly and many students struggle. If ' + heShePronoun + ' works hard I''m sure ' + heShePronoun + ' will do very well.';

SetLength(year11Sentences, 2);

year11Sentences[0] := 'Year 11 is a stressful year for many students. ' + firstName + ' has to take many subjects and it''s definately challenging, work as hard as you can.';

year11Sentences[1] := firstName + ' is now doing ' + hisHerPronoun + ' GCSE''s, they can be stressful for many students but if they focus they call all get through with a good result.';

SetLength(year12Sentences, 2);

year12Sentences[0] := 'Year 12 is very difficult and the jump from GCSE''s is huge. It is really important ' + firstName + ' works hard this year for A-Levels next

```

year.';

year12Sentences[1] := firstName + ' has of course started A-Levels now, they
are difficult and the work will be frequent so ' + heShePronoun + ' must keep on
top of it all.';

SetLength(year13Sentences, 2);
year13Sentences[0] := 'As you know ' + firstName + ' is in year 13, this year is
shorter and the final exams approach quickly. I hope ' + firstName + ' is revising
regularly and going over year 12 content.';

year13Sentences[1] := firstName + ' has made it to year 13, it is important to
start looking beyond at University or Apprenticeships but it is also important to focus
on the A-Level exams and work hard!';

if yearGroup = '07' then
  finalParagraph := finalParagraph +
year7Sentences[Random(Length(year7Sentences))];
  if yearGroup = '08' then
    finalParagraph := finalParagraph +
year8Sentences[Random(Length(year8Sentences))];
  if yearGroup = '09' then
    finalParagraph := finalParagraph +
year9Sentences[Random(Length(year9Sentences))];
  if yearGroup = '10' then
    finalParagraph := finalParagraph +
year10Sentences[Random(Length(year10Sentences))];
  if yearGroup = '11' then
    finalParagraph := finalParagraph +
year11Sentences[Random(Length(year11Sentences))];
  if yearGroup = '12' then
    finalParagraph := finalParagraph +
year12Sentences[Random(Length(year12Sentences))];
  if yearGroup = '13' then
    finalParagraph := finalParagraph +
year13Sentences[Random(Length(year13Sentences))];
end;

procedure TfrmReportGeneratorPage.GenerateReport;
begin
// If the radio groups have not been selected then an error message is displayed
using the error message label
  if (rgWorkCompletion.ItemIndex = -1) or (rgHomeworkCompletion.ItemIndex =
-1) or (rgWorkQuality.ItemIndex = -1)
  or (rgHomeworkQuality.ItemIndex = -1) or (rgLessonInvolvement.ItemIndex = -1)
  or (rgBehaviour.ItemIndex = -1) then begin

```

```
lblErrorMessage.Caption := 'You must fill in each section';
lblErrorMessage.Visible := true;
exit;
end;
lblErrorMessage.Visible := false;

// Initially sets the final paragraph string to empty

finalParagraph := "";

// Runs the set all sentences procedure so the sentences have been made, this cannot be done beforehand since it is not hard coded, it uses the name variables and other variables of the student to create the personalised report
SetAllSentences;

// assigns the values of the radio groups to these variables
workCompletion := rgWorkCompletion.ItemIndex + 1;
workQuality := rgWorkQuality.ItemIndex + 1;
homeworkCompletion := rgHomeworkCompletion.ItemIndex + 1;
homeworkQuality := rgHomeworkQuality.ItemIndex + 1;
behaviour := rgBehaviour.ItemIndex + 1;
lessonInvolvement := rgLessonInvolvement.ItemIndex + 1;

// calculates the overall weight by using the mean value of all the variables
overallWeight := (workCompletion + workQuality + homeworkCompletion +
homeworkQuality + behaviour + lessonInvolvement) DIV 6;

// Before adding the introduction sentences it checks if the check box is ticked
if chkIncludeIntroduction.Checked then begin
  finalParagraph := finalParagraph +
introductionSentences.GetArray(overallWeight); // now the introduction sentences object runs its get array function which returns an array with the overallWeight as input, if the overall weight was 5 it would return a sentence from the excellent array to show the student was excellent, this is then appended to the final paragraph string
end;

finalParagraph := finalParagraph + ' ';
finalParagraph := finalParagraph + workQualitySentences.GetArray(workQuality);
finalParagraph := finalParagraph + ' ';
finalParagraph := finalParagraph +
workCompletionSentences.GetArray(workCompletion);
finalParagraph := finalParagraph + ' ';
```

```

finalParagraph := finalParagraph +
homeworkQualitySentences.GetArray(homeworkQuality);
finalParagraph := finalParagraph + ' ';
finalParagraph := finalParagraph +
homeworkCompletionSentences.GetArray(homeworkCompletion);
finalParagraph := finalParagraph + ' ';
finalParagraph := finalParagraph + behaviourSentences.GetArray(behaviour);
finalParagraph := finalParagraph + ' ';
finalParagraph := finalParagraph +
lessonInvolvementSentences.GetArray(lessonInvolvement);
finalParagraph := finalParagraph + ' ';

if chkMentionGrades.Checked then begin
  if (Length(subjects) > 1) then begin
    Grades;
    finalParagraph := finalParagraph + ' ';
  end;
end;

if chkMentionAttendancePunctuality.Checked then begin
  AttendancePunctuality;
  finalParagraph := finalParagraph + ' ';
end;

if chkMentionAchievementBehaviour.Checked then begin
  BehaviourAchievementPoints;
  finalParagraph := finalParagraph + ' ';
end;

SchoolYear;
finalParagraph := finalParagraph + ' ';

finalParagraph := finalParagraph +
HowToImproveSentences.GetArray(workQuality);
finalParagraph := finalParagraph + ' ';

if chkIncludeConclusion.Checked then begin
  finalParagraph := finalParagraph +
conclusionSentences.GetArray(overallWeight);
end;

Hide;
frmFinalParagraphPage.Show;
frmFinalParagraphPage.redtFinalParagraph.Text := finalParagraph;

```

```
// It sets the final paragraph pages rich edit box to the final paragraph, this page
will be outlined in the next prototype
end;
```

```
// Public procedure RegenerateReport will be used in the final paragraph form that
has not been implemented, this procedure essentially acts a getter, hiding the
generate report function to other classes but makes the regenerate report visible.
procedure TfrmReportGeneratorPage.RegenerateReport;
begin
  GenerateReport;
end;
end.
```

## Requirements met

This has met requirements 11, 12 and 13.

Number	Feature	Explanation and justification
11.	Each form should have a dark blue background with contrasting white text.	The software should be easy to look at and have a clear layout, by having a dark blue background it creates a good look without being distracting. During the interview with Mr. Nicolston this was outlined as a key requirement.
12.	Every title, button and edit box should be clearly labelled.	Once again the software should be easy to use, the user should understand what everything does right away, this requirement will help with this.
13.	Verdana font should be used throughout each form. Verdana size 16 bold should be used for each button and size 16 regular for each edit box. Each title should be size 48 bold.	The verdana font is a clean professional look, it will improve the UI significantly if this font is used throughout the program with consistent sizing and bold or regular formats.

## Testing

Test number	Test purpose	Test data	Expected result	Result
10.3	Check if error message is displayed when radio groups have not been filled in	radio groups current index: -1	'You must fill in each section' error message should appear	Success

### 10.3 Results

Generate Report X

# GENERATE REPORT

GO BACK

Generate report for Peter Lara

(1 - Major Cause Of Concern 5 - Excellent)

**Include Sections**

- Include Introduction
- Include Conclusion
- Mention Grades
- Mention Attendance And Punctuality
- Mention Behaviour And Achievement Points

Work Completion

 1  2  3  4  5

Work Quality

 1  2  3  4  5

Homework Completion

 1  2  3  4  5

Homework Quality

 1  2  3  4  5

Behaviour

 1  2  3  4  5

Lesson Involvement

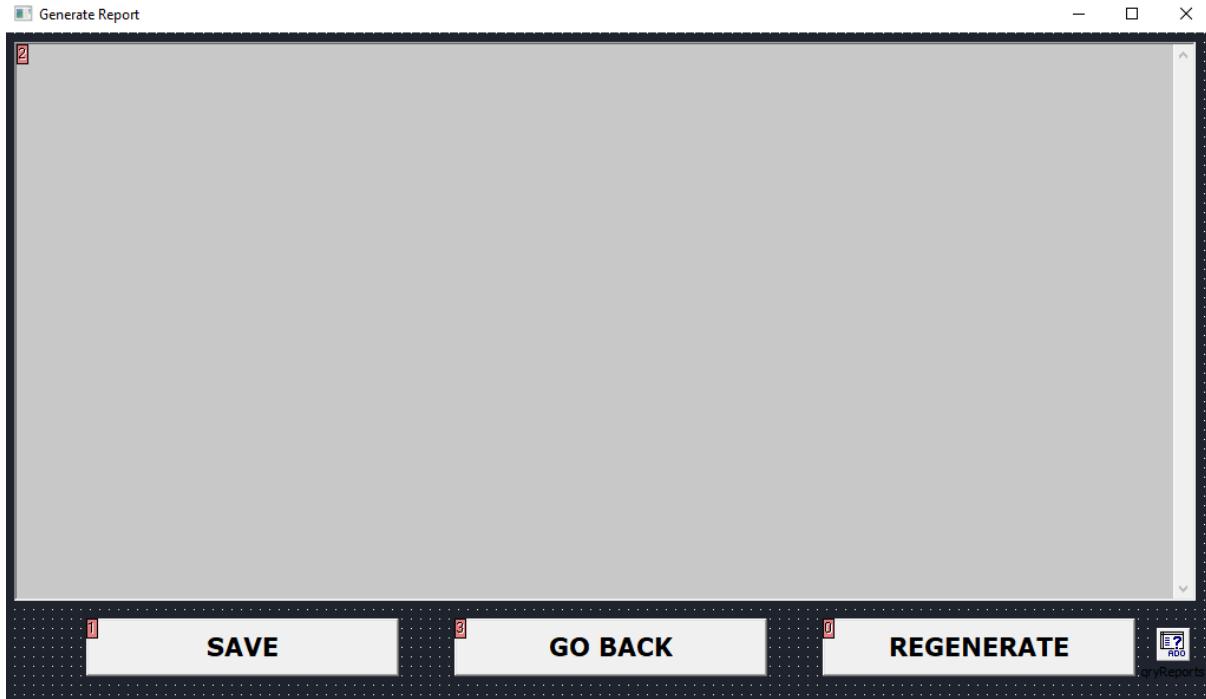
 1  2  3  4  5

You must fill in each section GENERATE

## 3.12 Stage 12 Final paragraph page

### 3.12.1 Prototype 1

Form screenshot



## Form accomplishments

The first prototype for the final paragraph page has been created, unlike the other forms that were at an early stage, this first prototype managed to contain an improved UI design which is consistent with the rest of the program. This form is smaller in size so more time was spent in the first prototype improving the design. This form contains a large rich text edit box in the centre of the page taking up a vast amount of space. This is to allow the user to see their reports in full. Below the edit box are 3 buttons: save, go back and regenerate. This form has a dark blue background like the other forms which improves the look, it uses verdana font and large clear buttons for good readability. The text displayed in the edit box is large and clear so the users can see their reports in full.

## Code

### unitFinalParagraphPage

unit unitFinalParagraphPage;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, StdCtrls, ComCtrls, DB, ADODB;

```
type
  TfrmFinalParagraphPage = class(TForm)
    btnRegenerate: TButton;
    btnSave: TButton;
    redtFinalParagraph: TRichEdit;
    btnGoBack: TButton;
    qryReports: TADOQuery;
    procedure btnGoBackClick(Sender: TObject);
    procedure btnSaveClick(Sender: TObject);
    procedure btnRegenerateClick(Sender: TObject);

  private
    procedure UpdateSQL;

  end;

var
  frmFinalParagraphPage: TfrmFinalParagraphPage;

implementation

uses unitReportGeneratorPage, unitLogin, unitGenerateReport;

{$R *.dfm}

// Returns to the report generator page if the staff wants to go back to change their settings
procedure TfrmFinalParagraphPage.btnGoBackClick(Sender: TObject);
begin
  Hide;
  frmReportGeneratorPage.Show;
end;

// Calls the regenerate report function which was created in the other form as a public function, this runs the process of creating the report again and it's instantly updated on this page
procedure TfrmFinalParagraphPage.btnRegenerateClick(Sender: TObject);
begin
  frmReportGeneratorPage.RegenerateReport;
end;

// Confirmation message when they click save, then it runs an SQL to update the report to the database
```

```

procedure TfrmFinalParagraphPage.btnSaveClick(Sender: TObject);
begin
  if MessageDlg('Are you sure you want to save this report? You can view it on the
reports page.', mtConfirmation, [mbYes, mbNo], 0) = mrYes then begin
    UpdateSQL;
    Hide;
    frmGenerateReportPage.Show;
  end;
end;

procedure TfrmFinalParagraphPage.UpdateSQL;
var
  sqlStatement : string;
begin // uses the staffID and studentID from various forms to update the reports
table
  sqlStatement := 'INSERT INTO Reports (reportDocument, dateModified,
studentID, staffID) VALUES (' + QuotedStr(redtFinalParagraph.Text) + ', ' +
QuotedStr(DateToStr(Now())) + ', ' +
QuotedStr(frmGenerateReportPage.StudentID) + ', ' + QuotedStr(frmLogin.StaffID)
+ ')';

  with qryReports do begin
    SQL.Clear;
    SQL.Add(sqlStatement);
    ExecSQL;
  end;
end;

end.

```

## Requirements met

This has met requirements 7, 11, 12, 13, 51, 52, 53, 54, 55 and 56.

Number	Feature	Explanation and justification
7.	The main program forms must follow a 16:9 aspect ratio and be smaller than the monitor size.	The main forms must follow the standard monitor aspect ratio, they should also be smaller than the monitor to allow for a wider range of monitors. By doing this monitors with a 720p display or

		lower will still be able to run the program and view each form fully.
11.	Each form should have a dark blue background with contrasting white text.	The software should be easy to look at and have a clear layout, by having a dark blue background it creates a good look without being distracting. During the interview with Mr. Nicolston this was outlined as a key requirement.
12.	Every title, button and edit box should be clearly labelled.	Once again the software should be easy to use, the user should understand what everything does right away, this requirement will help with this.
13.	Verdana font should be used throughout each form. Verdana size 16 bold should be used for each button and size 16 regular for each edit box. Each title should be size 48 bold.	The verdana font is a clean professional look, it will improve the UI significantly if this font is used throughout the program with consistent sizing and bold or regular formats.
51.	Final paragraph page should display a large text box with the final report.	In order for the staff to see the report clearly and make changes easily, a large text box containing the final paragraph should be present.
52.	Final paragraph page should contain 3 buttons below the text box: save, go back and regenerate.	There should be 3 clear buttons below the report, these names will make it clear what each button does and having them at the bottom of the page makes it easily accessible.
53.	Program must contain a button to regenerate the report, creating a new random report but still based on the inputs the staff gave.	If a staff member is unhappy with the report they should be able to press the regenerate button, this will repeat the algorithm used to create the report and change it again.
54.	Program should allow the staff to make changes to the final report before saving it.	The staff will know more about the students than the program will, they are likely to want to add more

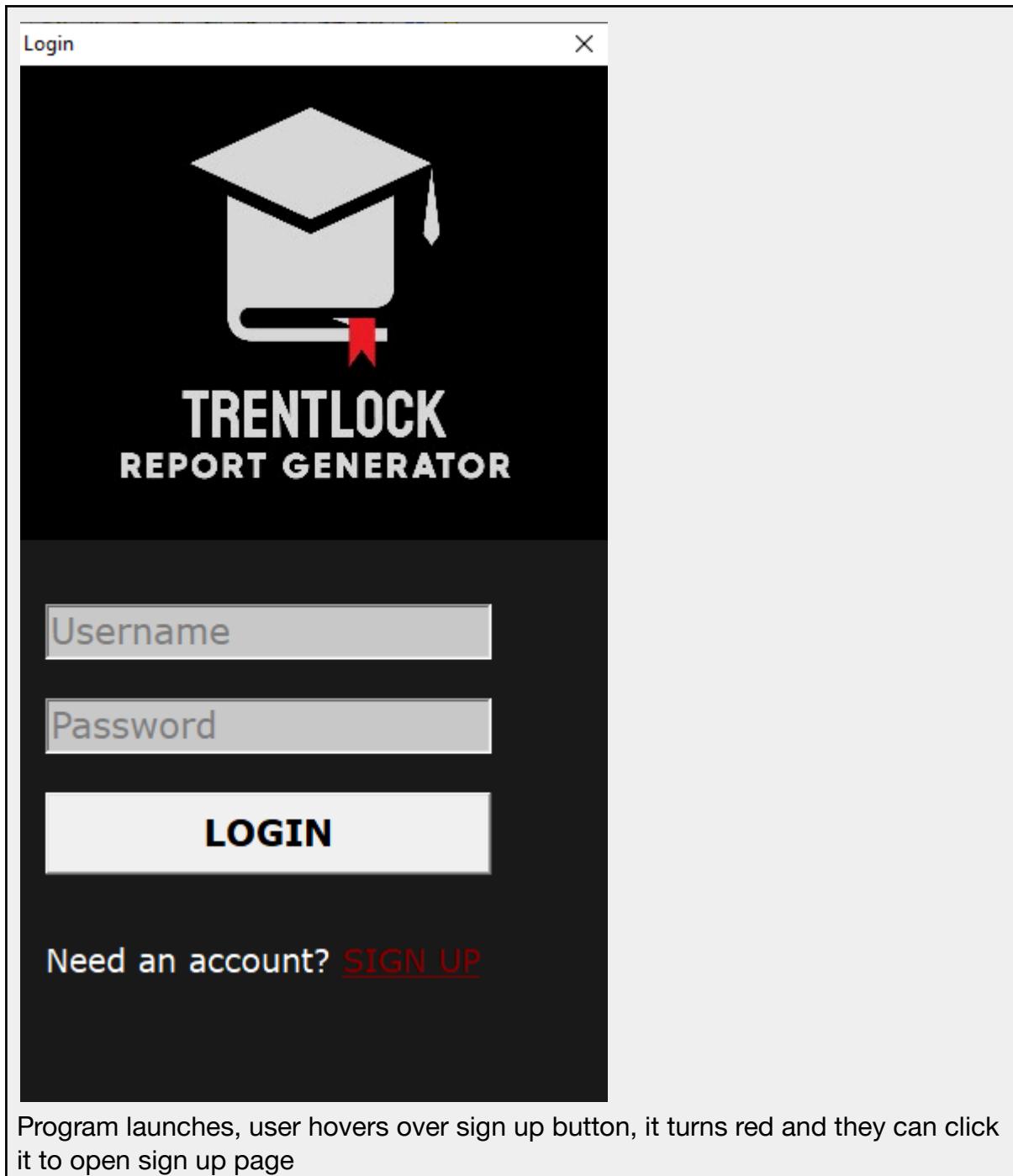
		to the report at the end or make changes to it. The text box should be editable so they can make changes before saving. This was a key requirement outlined during the planning section.
55.	Program should have an option to save the report to the database.	Once the user is happy with the report they should be able to save it to the database so it is stored there and accessible any time by going to the database table that was mentioned earlier as a requirement.
56.	Program should have an option to email the parents of the student with the report attached.	Once the staff has created the report they should be able to email the parents of the students with the parent email field from the database, this will mean the report can get to the parent quickly.

## 4. Evaluation

### 4.1 System testing & end user testing

#### 4.1.1 System testing

**Test 1 - Testing the process of creating an account, logging in and editing account details**



Program launches, user hovers over sign up button, it turns red and they can click it to open sign up page

Sign Up X

*First Name*  
Imran

*Last Name*  
Wadud

*Email*  
IWadud@trentlock.com

*Username*  
imranwadud

*Password*  
\*\*\*\*\*

I CONFIRM I WORK AT TRENTLOCK

**SIGN UP**

Have an account? [LOGIN](#)

Password must contain a special character

User enters first name, last name, email, username and password, they then click a confirm box outlining if they work at the school. Error message is displayed below if they didn't fill in a box properly. They then click sign up to sign up.

Sign Up X

*First Name*  
Imran

*Last Name*  
Wadud

*Email*  
IWadud@trentlock.com

Confirm X

Confirm Sign Up - Are you sure these details are correct?

 ?

I CONFIRM I WORK AT TRENTLOCK

**SIGN UP**

Have an account? [LOGIN](#)

Confirm box appears after clicking sign up asking if they are sure the details are correct.

Sign Up X

*First Name*  
Imran

*Last Name*  
Wadud

*Email*  
IWadud@trentlock.com

*User* Reportgenerator X

ii Congratulations, your account has been created

P \* OK

I CONFIRM I WORK AT TRENTLOCK

**SIGN UP**

Have an account? [LOGIN](#)

Once they click yes another confirmation box appears telling them their account has been made.

Login X

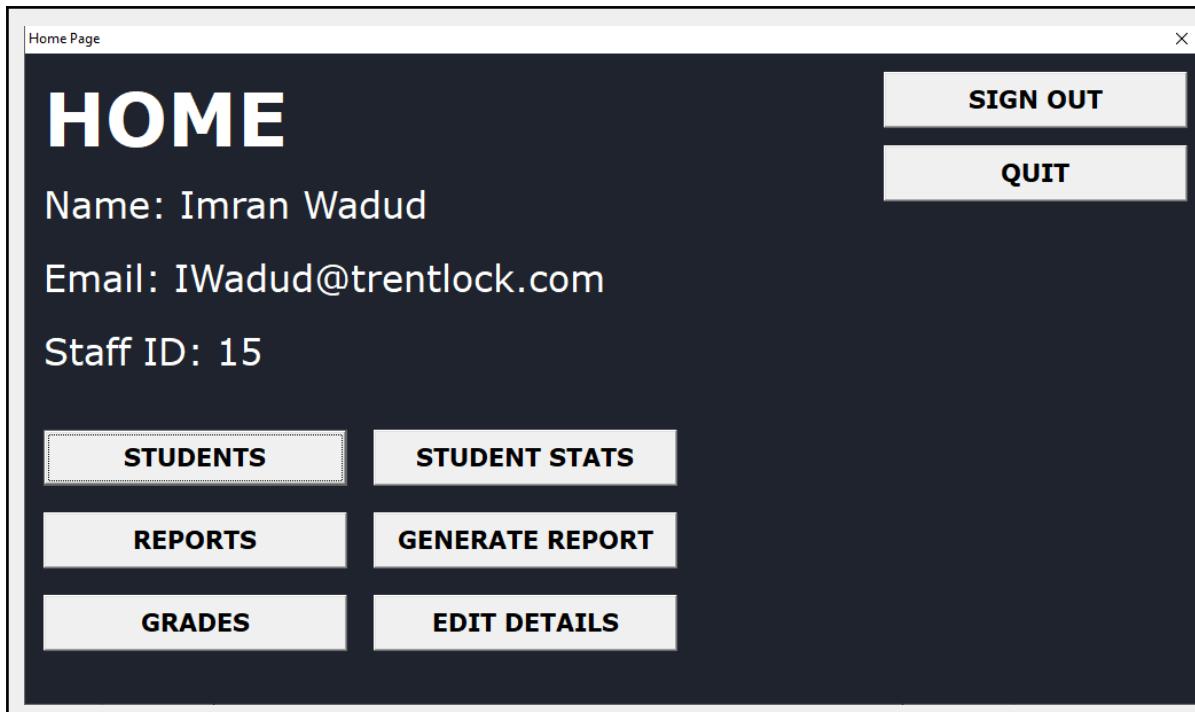


**TRENTLOCK**  
**REPORT GENERATOR**

**LOGIN**

Need an account? [SIGN UP](#)

It then returns them to the login page automatically so they can log in with their new details.



Once details are entered and they click login they are taken to the home page, this shows their name email and student ID

The 'Edit Details' page has a dark background. It features input fields for First Name ('Imran'), Last Name ('Wadud'), Email ('IWadud@trentlock.com'), Address, Phone Number, National Insurance Number, Username ('imranwadud'), and Password ('imranwadud!'). On the right side, there are 'GO BACK', 'SAVE', and 'CANCEL' buttons. The 'SAVE' button is highlighted with a dotted border.

Edit details page contains the details entered during sign up and other details they can save

Edit Details

# EDIT DETAILS

**GO BACK**

First Name	Last Name	<b>SAVE</b>
Imran	Wadud	
Email	IWadud@trentlock.com	<b>CANCEL</b>
Address	SW15 9QS	
Phone Number	071765422214	
<i>National Insurance Number</i>		
Username	Password	
imranwadud	imranwadud!	

New details have been entered into the boxes

Edit Details

# EDIT DETAILS

**GO BACK**

First Name	Last Name	<b>SAVE</b>
Imran	Wadud	
Email	IWadud@trentlock.com	<b>CANCEL</b>
Address	SW15 9QS	
Phone Number	071765422214	
<i>National Insurance Number</i>		
Username	Password	
imranwadud	imranwadud!	

Confirm

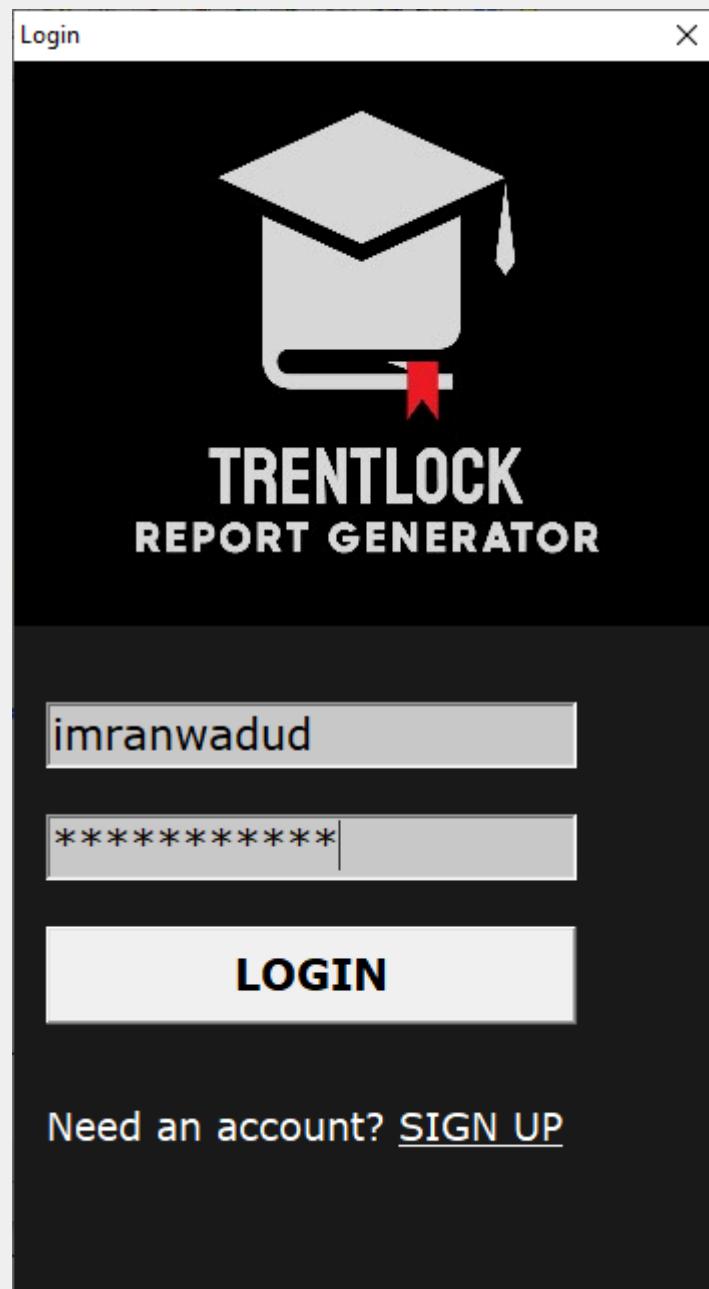
Are you sure you want to update your details? The changes are as follows:  
 Current Address: --> New Address: SW15 9QS  
 Current Phone Number: --> New Phone Number: 071765422214

Pressing the save button brings up a confirmation box listing the changes that are about to be made, if they click no they are taken back, if they click yes the details are saved to the database and officially updated

**Requirements met - 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 16, 17, 19, 20, 21, 23, 25, 26, 27, 28, 29, 31, 32, 33**

**Test success - 'We were able to launch the program, create an account, login with our new details, then go to edit our details in the edit details page and save the changes.'**

**Test 2 - Testing the process of a staff member logging in, viewing their students in the students page, viewing the student performance in the student performance page, viewing the grades in the grades page and filtering grades.**



User logs in with their username and password

Home Page

# HOME

Name: Imran Wadud

Email: IWadud@trentlock.com

Staff ID: 15

**STUDENTS**    **STUDENT STATS**

**REPORTS**    **GENERATE REPORT**

**GRADES**    **EDIT DETAILS**

**SIGN OUT**

**QUIT**

Home page displays users name, email and staff ID

Students

# YOUR STUDENTS

**(View Only)**

Student ID	First Name	Last Name	Year Group	Form Group	Student Email
10	Piper	Mckenzie	08	RRW	5057@trentloc
11	Landyn	Mcclain	12	SAA	5058@trentloc
12	Peter	Lara	13	SHP	5059@trentloc
13	Kassidy	Sims	11	SLH	5060@trentloc
14	Landen	Proctor	09	TJG	5061@trentloc
15	Amari	Mccarty	08	WJC	5062@trentloc
16	Uriah	Edwards	12	LYG	5063@trentloc
17	Jerry	Beasley	10	LAL	5064@trentloc
18	Lacey	Townsend	13	TJG	5065@trentloc
19	Kaitlin	Best	12	BJH	5066@trentloc

**GO BACK**

**SEARCH:**

Once they click the students page they are displayed a table with their students, the highlighted row is the one they have searched for in the search box. Here they have searched for Peter and it has automatically been highlighted

Student Performance

# YOUR STUDENTS

**GO BACK**

**Performance Overview (View Only)**

Student ID	First Name	Last Name	Attendance	Punctuality	Behaviour P
20	Abraham	Leon	46	86	
10	Piper	Mckenzie	8	91	

**SEARCH:** First Name    Last Name    10

After the students page they click go back to go to the home page and then click student stats to go to the performance page, this page again displays a table with the students performance. The user searches for a student with student ID of 10 and it is highlighted in the table

Grades

# ALL STUDENTS

**GO BACK**

**Grade Overview (View Only)**

**FILTER BY:** Working At Grade    Target Grade    Year  
                     Predicted Grade    EOY Grade    Subject

Student ID	First Name	Last Name	Year	Subject	Working A
1	Jamarion	Green	2022	Maths	B
1	Jamarion	Green	2022	English	C
1	Jamarion	Green	2022	History	A*
3	Barrett	Hays	2022	Chemistry	B
3	Barrett	Hays	2021	Chemisty	B

**SEARCH:** First Name    Last Name    Student ID

From the home page they click the grades button and are taken here, it displays the students grades with search and filter options

Grades

# ALL STUDENTS

**GO BACK**

**Grade Overview (View Only)**

**FILTER BY:** B Target Grade Year  
Predicted Grade EOY Grade Subject

Last Name	Year	Subject	Working At	Target	Predicated	End Of Y
Green	2022	Maths	B	A	A*	B
Hays	2022	Chemistry	B	D	B	B
Hays	2021	Chemistry	B	B	C	D

**SEARCH:** First Name Last Name Student ID

The working at grade is set to 'B' and it now shows all students working at a B showing the filter is working as intended

**Requirements met - 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 16, 24, 27, 28, 29, 35, 36, 37, 38, 39, 40, 41, 44**

**Test success - 'We were able to login, go to the students page and view our students, search for a student in the search box. Then we were able to go to the student stats page, look at our students statistics and search for a student by ID, then we were able to go to the grades page and filter students who are working at a B.'**

**Test 3 - Testing the process of a staff member logging in, going to the reports page and editing a report they have for a student**

Login X



**TRENTLOCK**  
**REPORT GENERATOR**

**LOGIN**

Need an account? [SIGN UP](#)

User logs in with username and password

Home Page

# HOME

Name: Imran Wadud

Email: IWadud@trentlock.com

Staff ID: 15

**STUDENTS**    **STUDENT STATS**

**REPORTS**    **GENERATE REPORT**

**GRADES**    **EDIT DETAILS**

**SIGN OUT**

**QUIT**

Home page is displayed with the name, email and staff ID after logging in

Reports

# YOUR REPORTS

(Edit Permissions Enabled)

S-ID	First Name	Last Name	Date Modified
5	Kaylie	Bartlett	17/01/2023
6	Zavier	Hoffman	17/01/2023
7	Zaniyah	Harrington	17/01/2023
8	Keith	Zimmerman	
9	Freddy	Merritt	17/01/2023
3	Barrett	Hays	20/03/2023
6	Zavier	Hoffman	21/03/2023
8	Keith	Zimmerman	21/03/2023
1	Jamarion	Green	28/03/2023

**GO BACK**

points. The average amount of achievement points is currently 34 which means Jamarion is below average and should try and improve this. Furthermore, the average amount of behaviour points is currently 143 which means Jamarion is below average and should aim to increase this. It is important to do well at this stage, with GCSE's beginning next year they should focus in lessons and it will make revising far easier when the real exams come. Jamarion needs to continue their

**SEARCH:** 16/01/2023  jamar

Clicking the reports button opens the reports page displaying all the reports for the staff's students. Staff has searched for the student name 'Jamarion', and the table is updated during each keystroke so Jamarion has been highlighted in the table before finishing the name. Report for Jamarion is displayed on the right

Reports X

# YOUR REPORTS

**(Edit Permissions Enabled)**

S-ID	First Name	Last Name	Date Modified
5	Kaylie	Bartlett	17/01/2023
6	Zavier	Hoffman	17/01/2023
7	Zaniyah	Harrington	17/01/2023
8	Keith	Zimmerman	
9	Freddy	Merritt	17/01/2023
3	Barrett	Hays	20/03/2023
6	Zavier	Hoffman	21/03/2023
8	Keith	Zimmerman	21/03/2023
1	Jamarion	Green	28/03/2023

SEARCH:

Furthermore, the average amount of behaviour points is currently 143 which means Jamarion is below average and should aim to increase this. It is important to do well at this stage, with GCSE's beginning next year they should focus in lessons and it will make revising far easier when the real exams come. Jamarion needs to continue their excellent revision and work ethic.

Further notes: I am really pleased with Jamarions maturity in lessons.

Staff has edited the report for Jamarion shown in the highlighted section, this has now been saved automatically

**Requirements met - 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 24, 25, 27, 28, 29, 41, 42, 43, 44**

**Test success - 'We were successfully able to login, go to the reports page and view all our reports, we were able to search for the student name and view the reports on the right, we were then able to edit the reports and have them automatically saved, updating the date we modified them.'**

**Test 4 - Testing the process of a staff member logging, generating a report for their student and saving it to the database.**

Login X



**TRENTLOCK**  
**REPORT GENERATOR**

**LOGIN**

Need an account? [SIGN UP](#)

User logs in with their username and password

Home Page

# HOME

Name: Imran Wadud

Email: IWadud@trentlock.com

Staff ID: 15

**STUDENTS**

**REPORTS**

**GRADES**

**STUDENT STATS**

**GENERATE REPORT**

**EDIT DETAILS**

**SIGN OUT**

**QUIT**

Home page displays name, email and staff ID of user who just logged in

Generate Report

X

## SELECT STUDENT

**GO BACK**

Select the student you want to write a report for

**SEARCH:** Barrett

Jamarion Green

Jaylan Trujillo

**Barrett Hays**

Skye Morales

Kaylie Bartlett

Zavier Hoffman

Zaniyah Harrington

Keith Zimmerman

Freddy Morris

Student ID	Year Group	Form Group	Candidate No.
3	09	BJD	5050

*Parent Email*

Evanescence@hotmail.co.uk

*Student Email*

5050@trentlock.org

**CONTINUE**

After clicking the generate report button page opens allowing staff to select a student, a list of students is displayed with boxes showing key information for students. Staff searches for the student name 'Barrett' and the selection is automatically updated in the list as shown.

Generate Report X

# GENERATE REPORT

**Generate report for Barrett Hays**

(1 - Major Cause Of Concern 5 - Excellent)

<b>Include Sections</b> <ul style="list-style-type: none"> <li><input type="checkbox"/> Include Introduction</li> <li><input type="checkbox"/> Include Conclusion</li> <li><input type="checkbox"/> Mention Grades</li> <li><input checked="" type="checkbox"/> Mention Attendance And Punctuality</li> <li><input checked="" type="checkbox"/> Mention Behaviour And Achievement Points</li> </ul>	<b>Work Completion</b> <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5	<b>Work Quality</b> <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5
	<b>Homework Completion</b> <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5	<b>Homework Quality</b> <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5
	<b>Behaviour</b> <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5	<b>Lesson Involvement</b> <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5

**GENERATE**

Once staff click continue a new page opens telling them to generate the report for the student name they just selected. Staff have 6 options to rank their students from 1 to 5. Each radio box has been filled in and certain section have been left out which is the preference of the staff

Generate Report X

Barrett works hard and does well, she has a really good understanding of what's begin taught and she can get to the challenging questions quickly. Barrett has good completion of work. Any work that needs to be done during lesson gets done and I am happy with the amount she completes in a lesson. The quality of homework is really good, each task I set is handed back and marked well. Her completion of homework is also great, every assignment is handed in on time and she has never missed an assignment. Barrett's behaviour is good, she listens well and doesn't mess around in lesson. Barrett has been excellent in lesson involvement, constantly raising her hand and asking or answering questions. This year Barrett has reached an attendance of 59%. The average attendance is currently 60% so she is below average, this is concerning and she needs to make sure she turns up more regularly. Barrett currently has 48 achievement points and 4 behaviour points. The average amount of achievement points is currently 34 so I'm happy to report Barrett is above average. Furthermore, the average amount of behaviour points is currently 143 which means Barrett is below average and should aim to increase this. It is important to do well at this stage, with GCSE's beginning next year they should focus in lessons and it will make revising far easier when the real exams come. Barrett needs to use the resources available to them to continue revising.

**SAVE**   **GO BACK**   **REGENERATE**

Once generate button is clicked this form opens containing the generated report, it is for the student they selected and matches the options the staff picked in the previous form

Generate Report

Barrett works hard and does well, she has a really good understanding of what's been taught and she can get to the challenging questions quickly. Barrett has good completion of work. Any work that needs to be done during lesson gets done and I am happy with the amount she completes in a lesson. The quality of homework is really good, each task I set is handed back and marked well. Her completion of homework is also great, every assignment is handed in on time and she has never missed an assignment. Barrett's behaviour is good, she listens well and doesn't mess around in lesson. Barrett has been excellent in lesson involvement, constantly raising her hand and asking or answering questions. This year Barrett has reached an attendance of 59%. The average attendance is currently 60% so she is below average. This is concerning and she needs to make sure she turns up more regularly. Barrett currently has an average achievement point of 3.5. This means she is above average and should aim to maintain this level. Furthermore, the average achievement point for Barrett's year group is 3.5. This means Barrett is below average, with GCSE's becoming easier when the real exams come. Barrett needs to use the resources available to them to continue revising.

Confirm

Are you sure you want to save this report? You can view it on the reports page.

**SAVE**    **GO BACK**    **REGENERATE**

Clicking the save button will bring up a confirmation box to confirm the changes

Generate Report

# SELECT STUDENT

Select the student you want to write a report for

**SEARCH:**

Jamarion Green  
Jaylan Trujillo  
Barrett Hays  
Skye Morales  
Kaylie Bartlett  
Zavier Hoffman  
Zaniyah Harrington  
Keith Zimmerman  
Freddy Morris

Student ID	Year Group	Form Group	Candidate No.
1	09	ADL	5048

*Parent Email*  *Student Email*  **CONTINUE**

Once the staff have confirmed they are taken back to the select student page where they can select another student to write a report for

Reports X

# YOUR REPORTS

**(Edit Permissions Enabled)**

S-ID	First Name	Last Name	Date Modified
6	Zavier	Hoffman	17/01/2023
7	Zaniyah	Harrington	17/01/2023
8	Keith	Zimmerman	
9	Freddy	Merritt	17/01/2023
3	Barrett	Hays	20/03/2023
6	Zavier	Hoffman	21/03/2023
8	Keith	Zimmerman	21/03/2023
1	Jamarion	Green	28/03/2023
3	Barrett	Hays	28/03/2023

Barrett works hard and does well, she has a really good understanding of what's been taught and she can get to the challenging questions quickly. Barrett has good completion of work. Any work that needs to be done during lesson gets done and I am happy with the amount she completes in a lesson. The quality of homework is really good, each task I set is handed back and marked well. Her completion of homework is also great, every assignment is handed in on time and she

**SEARCH:**

Reports page now shows the report that was just generated, proving it has been saved to the database and accessible any time

**Requirements met - 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 24, 26, 27, 28, 29, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55**

**Test success - 'We were able to login, select a student we wanted to write a report for, rank the student based on the different categories and then click generate to create the report. We were able to view and edit the report and the click save to save it to the database. We were then able to view the new report in the reports page.'**

## 4.1.2 End user testing

This section will contain the response from the planned questionnaire given to Mr. Wadud, these questions were outlined in section 2.7.2.

### 1. Does the login screen open when the program is launched?

'Hi, thanks for reaching out. Yep, I am able to launch the program and it shows me a login page.'

### 2. Does the sign up button open the sign up page?

'Yes, it did open the sign up page for me.'

**3. How easy is it to sign up?**

'The process was really simple, I had to fill in a few boxes and then click confirm.'

**4. Is the UI clear and easy to understand?**

'The interface is really nice, so far there have been no problems for us navigating the program, everything looks good and is clear.'

**4. Does the program inform you if you enter an incorrect field?**

'Yes, if I got one wrong it would highlight red and tell me the reason which was helpful.'

**5. Does the program save your details after signing up?**

'After I signed up there was a confirmation message telling me the process was successful which gave me confidence the process worked.'

**6. Can you successfully log in?**

'Yep, I entered my username and password and it took me right to the home page'

**7. Do you understand where each button takes you and what the purpose of each form is?**

'Yes, I liked how the home page clearly told me what each button did. Each page also had a title at the top so it was easy to tell what everything was for.'

**8. How easy is it to navigate the system from 1 to 10?**

'The program is really easy to navigate, I would say 10, everything is a click or two away and the ability to press the escape key to go back is useful.'

**9. Can you easily edit your details?**

'The edit details page was laid out clearly, I could see all my details in one page and I liked the confirmation box showing me what I was about to change. However I think there should have been a box containing the current details and the new details beside instead of combining them into one box. Also having a confirm

password box would have helped me ensure my new password was what I inputted.'

#### **10. Does the students table contain all the required information?**

'The students table is really thorough, if I'm unclear on anything about a student I can go there to see. This is a much needed improvement compared to our previous methods.'

#### **11. Are the tables easy to understand and use?**

'The tables are large and clear which was useful for us, the search functionality is particularly useful since some of us have a large number of students and it is hard to scroll through. It was very useful being able to sort by clicking the column title. '

#### **12. Can you clearly see your reports in the reports page?**

'Yes I could, the reports page showed all the reports I had, this was a huge improvement compared to beforehand. I also liked the search by date modified functionality so I could find reports from a certain date. However the text box which displayed the reports could have been bigger as I couldn't see the whole report at once.'

#### **13. How easy is it to search for a student or report?**

'This part was really easy to use, I could just click on the search box and start typing and the table would automatically update the selected student, we again found this really useful compared to beforehand.'

#### **14. How easy is it to select a student and generate a report for them?**

'It was really easy to select a student, I could scroll through the list of students and it would show me key information on the student I picked. I then had to click continue and it would take me to a page where I rank the student in different categories. Once I was happy with my options I clicked the generate button and a large report was made for me, easy to see and make changes to.'

#### **15. How easy is it to save the report to the database?**

'Once I had generated the report I could click the save button, a confirmation message would appear and I could click yes. Going back to the reports page I could see the report I just created for the student which was a nice feature.'

#### **16. How easy is it to create a new report for a new student or regenerate your current report?**

'This process was really simple, once I had saved my report the program would automatically take me to the select student page so I could quickly make another report for another student. There was also a button to regenerate the report which instantly updates the report, this was useful to get some variety.'

#### **17. Have you encountered any bugs or errors?**

'We haven't been able to find any bugs at all, this is great news and gives us confidence to continue using the program.'

#### **Evaluation**

Overall Imran, the stakeholder, was happy with the solution and found it useful. The UI was clear and easy to navigate and the process of creating a report was fast and useful. Despite this some functionality could have been improved. Increasing the size of the reports box in the reports page would have helped Imran read the reports in full instead of scrolling down. Changing the layout of the edit details page would have improved the usability, helping Imran tell which details are old and which are new more easily. In the future more client testing will be included during the development of each prototype.

## **4.2 Evaluate the solution**

This section will evaluate the program against the success criteria outlined in section 1.8.

#### **Success criteria 1 - Staff must be able to successfully login to their account**

This criterion has been successfully met. This is demonstrated by test number 1 and it has met requirement 14.

**Success criteria 2** - Staff must be informed if they entered the wrong username or password so they can correct it

This criterion has been met successfully. This is demonstrated by test numbers 1.4 and 1.5. This has successfully met requirements 15.

**Success criteria 3** - Staff must be able to successfully create an account using their username, password, full name and email

This criterion has been successfully met. Test number 2 has demonstrated this. This has also met requirements 16 and 17.

**Success criteria 4** - If the staff enter invalid or empty data they must be informed

This criterion has also been met successfully. Testing numbers 2.1, 2.2, 2.3 and 2.4 have demonstrated this. This has met requirements 15.

**Success criteria 5** - Staff must be informed if the username they entered already exists

This criterion has now been successfully met. This has been proven by testing number 2.15. Requirements number 22 has been met.

**Success criteria 6** - Staff should be able to sign up and then automatically return to the login page successfully

This criterion has been successfully met. This is demonstrated by test number 2.16 and it has met requirements 23.

**Success criteria 7** - Staff should successfully be displayed a confirmation message

This criterion has been successfully met. This is demonstrated by test number 2.14 and it has met requirement 25.

**Success criteria 8** - Staff must be able to successfully update their details

This criterion has been successfully met. This is demonstrated by all of testing number 8. This meets requirements 33.

**Success criteria 9** - Staff must be informed if their username already exists

This criterion has been successfully met. This is demonstrated by all of testing number 8. This meets requirements 34.

**Success criteria 10** - Staff must successfully be able to view their students in a database grid from the students table

This criterion has been successfully met. This is demonstrated by testing 4.2 all of testing number 8. This meets requirements 36.

**Success criteria 11** - Staff must be able to search the students full name or student ID successfully

This criterion has been successfully met. This is demonstrated by testing 4.3, 4.4, 4.5 and 4.6. This meets requirements 37.

**Success criteria 12** - Staff should successfully be able to view various tables in the database. This would include grades, attendance, achievementPoints

This criterion has been successfully met. This is demonstrated by all of testing 5 and 6. Requirements number 41 has been met.

**Success criteria 13** - Staff must be able to view their previous reports for their students successfully

This criterion has been successfully met. This is demonstrated by all of testing 7 and 6. Requirements number 43 has been met.

**Success criteria 14** - Staff must be able to successfully select a student to write a report for from a list of students

This criterion has been successfully met. This is demonstrated by all of testing 9 and 6. Requirements number 48 has been met.

**Success criteria 15** - Staff must be able to successfully select different radio buttons for different sections and select various checkboxes to determine how the report will be formed

This criterion has been successfully met. This is demonstrated by all of testing 10. Requirements number 49 and 50 have been met.

**Success criteria 16** - Staff must be displayed a large text box containing the final report and be able to edit it successfully

This criterion has been successfully met. This has met requirement 51.

**Success criteria 17** - Staff must be able to successfully regenerate the report, creating a new random paragraph from the inputs they gave

This criterion has been successfully met. This has met requirement 53.

**Success criteria 18** - Staff must be able to successfully save the report to the database once they are happy with it

This criterion has been successfully met. This has met requirement 54.

**Success criteria 19** - Staff must be able to successfully email the parent of the student they wrote a report for

This criterion has not been met, unfortunately due to time constraints this request by the stakeholders did not come to fruition.

## 4.3 Further development

The solution has met the majority of the requirements, however there are a few that have unfortunately not been implemented into the solution. This section will cover any requirements that were not implemented and how they could be addressed during further development.

**Requirement 10** - The login form must contain a ‘forgot password’ button so the user can reset their password.

This feature was not implemented due to the time constraints, it would have required a lengthy process of implementing an email system that emails the user to confirm they are trying to reset their account and then send another email with the new password. If further development takes place this could be implemented and would be a welcome feature as users could reset their password if they ever forget instead of creating a new account.

**Requirement 30** - Home page should contain a profile picture of the staff logged in at the corner of the page

This feature was not implemented due to the complexity of connecting delphi to the existing school systems to access photographs. Delphi currently does not have a relationship to the software used by the school which stores the ID photographs of staff members so it cannot automatically display a profile picture in the program. The other method would be manually adding the photographs which would require more work for the school to manage. In the future this could be a feature to look into implementing, however with the current school systems this is not feasible.

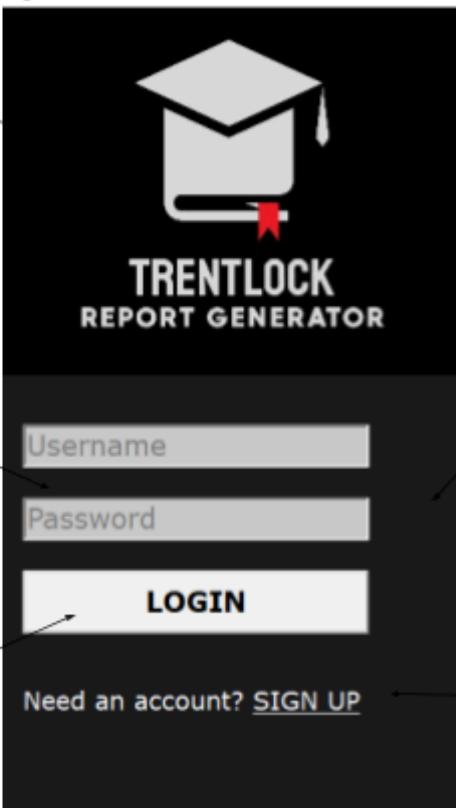
**Requirement 56** - Program should have an option to email the parents of the student with the report attached.

This feature could not be implemented because of the time constraints, it would have made use of the parent email field in the database and delphi's SMTP email system. The report would have been attached to the email with a custom message the staff wants to write. This feature should be looked into during further development as it could be a powerful tool to quickly send emails to the parents of the students.

## 4.4 Effectiveness of usability features

In section 2.4 a range of usability features were outlined, planning how easy it would be to navigate the program and use its functionalities. This section will provide evidence of the usability features planned in 2.4 showing how they came to fruition.

**Form name** - Login



The image shows a mobile-style login screen titled "Login". At the top is a logo of a graduation cap resting on a book with a red ribbon. Below the logo, the text "TRENTLOCK REPORT GENERATOR" is displayed in a bold, sans-serif font. The background is dark grey. There are two input fields: "Username" and "Password", both with placeholder text in a light grey font. Below these is a large, prominent "LOGIN" button with the word "LOGIN" in bold black capital letters. At the bottom of the screen, the text "Need an account? SIGN UP" is displayed, where "SIGN UP" is underlined to indicate it is a link.

Simple large logo, with text displaying name of school and program

Large clear edit boxes with grey colour which is less distracting compared to white

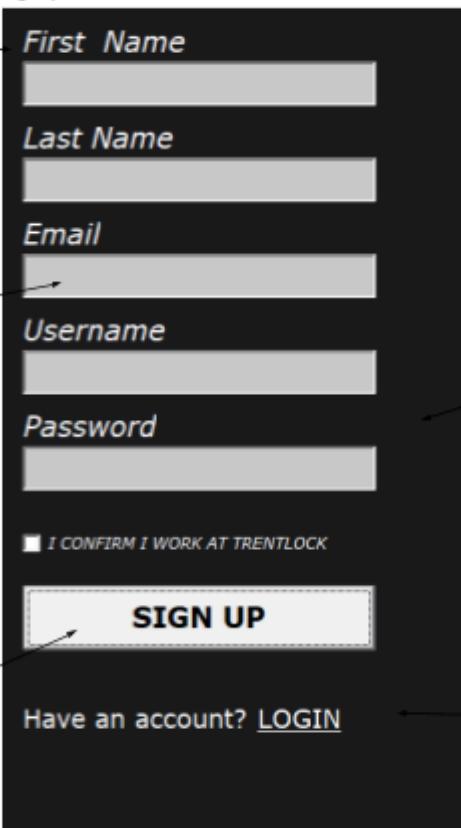
Large login button to easily login

Dark grey background, not distracting and looks good

Underlined sign up button that indicated is can be clicked

The design of this form has clearly met requirements 6, 8, 11, 12 and 13. The response from the usability tests show the stakeholder was satisfied with the design and functionality of this form and so this usability feature was successfully met.

**Form name** - Sign up



The form is titled "Sign Up" and features a dark grey background. It includes five input fields labeled "First Name", "Last Name", "Email", "Username", and "Password", each with a grey input box. Below these is a checkbox labeled "I CONFIRM I WORK AT TRENTLOCK". A large, prominent "SIGN UP" button is centered at the bottom, and a "Have an account? LOGIN" link is located below it.

- Clearly labelled edit boxes
- Large clear edit boxes with grey colour which is less distracting compared to white
- Dark grey background, not distracting and looks good
- Large sign up button to easily sign up
- Underlined login button that indicated is can be clicked

The design of this form has clearly met requirements 6, 11, 12 and 13. The response from the usability tests show the stakeholder was satisfied with the design and functionality of this form and so this usability feature was successfully met.

#### Form name - Home Page

Clear and large title at top of page indicating purpose of page

Dark blue background, not distracting and minimal

Large clear buttons to sign out or quit, far away from other buttons as they serve a different purpose

Staff information clearly displayed in large text

Large clear buttons to navigate to different forms

The design of this form has clearly met requirements 7, 8, 11, 12, 13 and 26. The response from the usability tests show the stakeholder was satisfied with the design and functionality of this form and so this usability feature was successfully met.

**Form name** - Students

Large table which matches colour of program, clear headers and rows with alternating colours to visually differentiate between each row

Students

Dark blue background, not distracting and minimal

Clear and large title at top of page indicating purpose of page

Large go back button placed at top right of each form for consistent experience

**YOUR STUDENTS**

(View Only)

Student ID	First Name	Last Name	Year Group	Form Group	Student Email
1	Jamarion	Green	09	ADL	5048@trentloc
2	Jaylan	Trujillo	10	AXM	5049@trentloc
3	Barrett	Hays	09	BJD	5050@trentloc
4	Skye	Morales	12	BJH	5051@trentloc
5	Kaylie	Bartlett	10	DGS	5052@trentloc
6	Zavier	Hoffman	10	EXV	5053@trentloc
7	Zaniyah	Harrington	11	JBS	5054@trentloc
8	Keith	Zimmerman	07	LAL	5055@trentloc
9	Freddy	Merritt	12	LYG	5056@trentloc

SEARCH:  First Name  Last Name  Student ID

Clearly labelled search boxes, simple grey colour which is less distracting compared to white

The design of this form has clearly met requirements 7, 8, 11, 12, 13 and 26. The response from the usability tests show the stakeholder was satisfied with the design and functionality of this form and so this usability feature was successfully met.

### Form name - Student Performance

Large table which matches colour of program, clear headers and rows with alternating colours to visually differentiate between each row

Dark blue background, not distracting and minimal

Large go back button placed at top right of each form for consistent experience

Clear and large title at top of page indicating purpose of page

**YOUR STUDENTS**

Performance Overview (View Only)

Student ID	First Name	Last Name	Attendance	Punctuality	Behaviour	Progress
1	Jamarion	Green	80	96	88	On Track
2	Jaylan	Trujillo	79	6	68	At Risk
3	Barrett	Hays	59	68	56	At Risk
4	Skye	Morales	97	88	88	On Track
5	Kaylie	Bartlett	98	88	88	On Track
6	Zavier	Hoffman	100	68	100	On Track
7	Zaniyah	Harrington	45	65	65	At Risk
8	Keith	Zimmerman	32	100	100	At Risk
9	Freddy	Merritt	12	54	54	At Risk

SEARCH:  First Name  Last Name  Student ID

Clearly labelled search boxes, simple grey colour which is less distracting compared to white

The design of this form has clearly met requirements 7, 8, 11, 12, 13 and 26. The response from the usability tests show the stakeholder was satisfied with the design and functionality of this form and so this usability feature was successfully met.

### Form name - Grades

Large table which matches colour of program, clear headers and rows with alternating colours to visually differentiate between each row

Dark blue background, not distracting and minimal

Large go back button placed at top right of each form for consistent experience

Clear and large title at top of page indicating purpose of page

Drop down boxes placed at top of form and clearly labelled

Clearly labelled search boxes, simple grey colour which is less distracting compared to white

Student ID	First Name	Last Name	Year	Subject	Working A
1	Jamarion	Green	2022	Maths	B
1	Jamarion	Green	2022	English	C
1	Jamarion	Green	2022	History	A*
3	Barrett	Hays	2022	Chemistry	B
3	Barrett	Hays	2021	Chemistry	B

The design of this form has clearly met requirements 7, 8, 11, 12, 13 and 26. The response from the usability tests show the stakeholder was satisfied with the design and functionality of this form and so this usability feature was successfully met.

### Form name - Reports

Large table which matches colour of program, clear headers and rows with alternating colours to visually differentiate between each row

Clear and large title at top of page indicating purpose of page

Dark blue background, not distracting and minimal

Large go back button placed at top right of each form for consistent experience

The screenshot shows a web application interface. At the top, there is a navigation bar with a 'Reports' link and a 'GO BACK' button. Below the title 'YOUR REPORTS' and a note '(Edit Permissions Enabled)', there is a table with columns: S-ID, First Name, Last Name, and Date Modified. The table contains 10 rows of student data. To the right of the table, a large text box displays a detailed report for a student named Barrett, with a grey background. At the bottom, there is a search bar with the placeholder 'SEARCH: 16/01/2023' and two input fields for 'Full Name' and 'Student ID'.

S-ID	First Name	Last Name	Date Modified
6	Zavier	Hoffman	17/01/2023
7	Zaniyah	Harrington	17/01/2023
8	Keith	Zimmerman	
9	Freddy	Merritt	17/01/2023
3	Barrett	Hays	20/03/2023
6	Zavier	Hoffman	21/03/2023
8	Keith	Zimmerman	21/03/2023
1	Jamarion	Green	28/03/2023
3	Barrett	Hays	28/03/2023

Clearly labelled search boxes, simple grey colour which is less distracting compared to white

Large text box displayed on right side, with large text display the reports. Grey background which is not distracting and easy on the eyes

The design of this form has clearly met requirements 7, 8, 11, 12, 13 and 26. The response from the usability tests show the stakeholder was satisfied with the design and functionality of this form and so this usability feature was successfully met.

#### Form name - Edit details

The form is titled "EDIT DETAILS" in large, bold, white capital letters at the top center. In the top left corner, there is a small link labeled "Edit Details". The top right corner features a large, light blue "GO BACK" button with a white "X" icon. The background of the form is a dark blue color.

First Name	Last Name
John	Smith

**Email:** johnsmith@hotmail.co.uk

**Address:** SW12 345

**Phone Number:** 07123456789

**National Insurance Number:** QQ 123456 C

**Username:** J

**Password:** J

**Buttons:** A large "SAVE" button and a large "CANCEL" button are positioned on the right side of the form.

**Annotations:**

- "Clear and large title at top of page indicating purpose of page" points to the title "EDIT DETAILS".
- "Dark blue background, not distracting and minimal" points to the dark blue background of the form.
- "Large go back button placed at top right of each form for consistent experience" points to the "GO BACK" button.
- "Differently sized text boxes to indicate length of input required" points to the "Username" and "Password" fields, which have different widths.
- "Clearly labelled edit boxes, simple grey colour which is less distracting compared to white" points to the "Email" field, which has a clear label and a grey background.
- "Large save and cancel buttons, easy to click and clear on purpose" points to the "SAVE" and "CANCEL" buttons.

The design of this form has clearly met requirements 7, 8, 11, 12, 13 and 26. The response from the usability tests show the stakeholder was satisfied with the design and functionality of this form and so this usability feature was successfully met.

**Form name** - Generate report

Generate Report

# SELECT STUDENT

Select the student you want to write a report for

**SEARCH:** Full Name

Jamarion Green
<b>Jaylen Trujillo</b>
Barrett Hays
Skye Morales
Kaylie Bartlett
Zavier Hoffman
Zaniyah Harrington
Keith Zimmerman
Candidate Number

Student ID	Year Group	Form Group	Candidate No.
2	10	AXM	5049

Parent Email: Defiant@hotmail.co.uk      Student Email: 5049@trentlock.org

**GO BACK**

**CONTINUE**

The design of this form has clearly met requirements 7, 8, 11, 12, 13 and 26. The response from the usability tests show the stakeholder was satisfied with the design and functionality of this form and so this usability feature was successfully met.

### Form name - Reports Generator

Clear and large title at top of page indicating purpose of page

Dark blue background, not distracting and minimal

Large go back button placed at top right of each form for consistent experience

Name of student displayed at top of form for no confusion

Well labelled check boxes

Well titled radio groups with clear radio buttons, outlined to show distinction between other radio groups

Large, clearly labelled generate button at bottom of page telling the user they should press this to create the report

The design of this form has clearly met requirements 7, 8, 11, 12, 13 and 26. The response from the usability tests show the stakeholder was satisfied with the design and functionality of this form and so this usability feature was successfully met.

#### **Form name - Final Paragraph**

Large text box with grey background for minimal less distracting look, large text displayed in text box to clearly show report in full

Generate Report X

Jaylan is an excellent student. Jaylan is a highly capable student who has performed consistently well since the beginning of term. The quality of his work has been nothing short of excellent, he shows a clear understanding of the concepts being taught and it's great to see this. The completion of work has been amazing, Jaylan always finishes the work I hand out and hardly ever needs help. Jaylan has been excellent in homework quality. Jaylan has been excellent in homework completion. Jaylan has been excellent in behaviour. Jaylan has been excellent in lesson involvement. This year Jaylan has reached an attendance of 79%. The average attendance is currently 60% so he is above average and doing well. Jaylan currently has 75 achievement points and 345 behaviour points. The average amount of achievement points is currently 34 so I'm happy to report Jaylan is above average. Furthermore, the average amount of behaviour points is currently 143 so I'm happy to report Jaylan is above average. These two years happen very quickly, if students don't revise consistently now they will struggle next year, I suggest Jaylan revises early to ensure a good GCSE performance. Jaylan needs to continue their excellent revision and work ethic. To conclude, Jaylan has been excellent.

SAVE    GO BACK    REGENERATE

3 clear large buttons at bottom of page, each titled with the purpose of each and easily accessible

The design of this form has clearly met requirements 7, 8, 11, 12 and 13. The response from the usability tests show the stakeholder was satisfied with the design and functionality of this form and so this usability feature was successfully met.

## 4.5 Limitations and potential improvements

One of the main limitations with the project was the time constraints. The lack of time meant I was unable to implement all the features that the stakeholders originally wanted. Certain tasks took longer than expected and more time was spent focusing on those crucial tasks to get the project to a working, completed state. Before implementing I knew time limitations would be a problem because the

stakeholders had given a set time span to complete the project, this is why in section 1.6 I outlined potential limitations of the solution. One of these limitations was the ability to display student photographs when clicking on a student. This feature would have required extensive programming to connect Delphi with the existing school systems which contain the student photographs. With an extended time frame this feature could be implemented and improve the overall solution. The existing database would have linked to the schools one and have a field storing the image file of the student, then through SQL queries these images could be displayed on the delphi pages.

The second main limitation with the project was Delphi 7 functionality, this IDE is powerful for creating software applications however there are many features it did not include making it impossible to meet some of the features the stakeholders wanted. One of the constraints of Delphi 7 was printer formatting functionality and printing functionality which was also outlined in section 1.6 as a potential limitation. This feature was not achievable due to there being no such feature in delphi. If Delphi had this feature this limitation could be resolved, the reports would have to be compatible with A4 size paper and then a nearby printer would have to be connected to the program so it could print the report to that printer. This would have improved the solution but unfortunately this has not become a feature.

The final limitation was finance. Researching a wider range of existing solutions would have improved the quality of the research and the results. Getting a more accurate understanding of all the existing solutions and their pros and cons would have helped significantly when designing, however many existing solutions require a digital purchase as they are paid pieces of software, this means they were inaccessible and the results of the research were not as strong as they could have been.

## 4.6 Maintenance issues

The program needs to be well maintained for future use to ensure it runs smoothly and as intended for staff members. However there are certain issues that could complicate the maintenance process, this section will address these issues and how they could be handled.

One maintenance issue that could exist is the process of system back-up. This process is important because if the database is not backed up it risks losing a large amount of data after a possible corruption or error. If there were to be corruption the

database would have to be built again and all missing information like staff, students and reports would be permanently lost. To resolve this issue frequent backups of the school database should take place. The school has access to a large hard drive where they can backup the database, to back up the database the scheduling feature can be used on Microsoft Access which allows backups to take place to the external hard drive at a selected time interval. This would provide safety and a level of insurance to the staff members working at Trentlock because they won't have to worry about losing information from the database.

Another maintenance issue that could exist is the process of upgrading the program. A new version of windows could be installed onto the school computers which changes the way the program behaves. Similarly new hardware could be installed at the school that would change how the user interacts with the program. The version of delphi that was used to create the program could also be updated providing new features and functionality that could be used to further develop the program. All these changes would require the program to be upgraded in some way, the main solution is for staff to regularly check the program to ensure it functions properly. If a new version of Delphi is released they should port the program to it, this would likely increase the stability of the program, increase performance and ensure it works on newer hardware devices or new software updates.

A third maintenance issue that could exist is the portability of the solution, it is highly unlikely that Tentlock school will change location so this should not be a concern, however looking at this from a smaller scale the problem of the program being able to run on multiple computers and laptops throughout the school should be considered. When creating the solution I made use of the structure of the school servers, the database and application file is stored on a drive that is accessible on all computers and laptops connected to the school network, this allows anyone to run the application. However if these systems break or no longer work for a particular reason a hard copy of the application and database is stored on 5 computers throughout the school and their main backup hard drive. This ensures the program is accessible everywhere and can be accessed whether the school is online or offline.