Name: Rayaan Uddin Candidate Number: 6250 Centre Number: 11022

OCR Computer Science Coursework

An implementation of a Report System



Introduction to the system

Contents

Section 1: Analysis 5
1.1 Features that make the problem solvable by computational methods 5
Problem Identification 5
Solving the Problem 5
System needs 5
1.2 Suitable stakeholders 5
Information Gathering 6
Planning 6
Interview – what the system needs (Mr Patel) [18/09/2022] 6
Survey 7
Email – Design of the system [Mr Wino] 9
Information on how student classes work in Graveney 9
Interview – Problem with the Existing System (Mr Wino) [1 to 1 on zoom at 16:30] 9
Graveney School Classes Format 10
1.3 Existing solutions 11
Solution A 11
Solution B 12
1.4 Features of the proposed computational solution 15
1.5 Limitations of the proposed solution 15
1.6 Requirements for the solution including hardware and software requirements 16
Hardware Requirements for the System 16
Software Requirements for the System 16
Features for the system 16
1.7 Success criteria 20
Section 2: Design 21
2.1 Decomposing the Problem 22
Illustration of the solution 22
2.2 Structure of the solution 23
Setting up the Entity Relationship Diagrams 23
Interface Design 24
Data dictionary 31
2.3 Algorithms 34
Query to get user details: 34
Query's to write user details 34
Query's to Update User details 34

Query to Create a Class: 34
Query to display users classes: 34
Query to Update a Class: 35
Query to Create Student: 35
Query to Update Student: 35
Query to Create Report: 35
Query to Get Report: 35
Login Form (Login Algorithm): 36
Signup Form (Create Account Algorithm): 36
Email Verification: 37
Create Class Form: 38
Delete Class Algorithm: 38
Create Student Form: 38
Delete Student Algorithm: 39
Create Report Form (Creation): 39
Create Report Form (Validation): 40
Report Comment Form: 40
2.4 Usability features 42
2.5 Key variables / data structures 43
Login form: TUserData Record 43
Login form 43
Signup form 43
Create Report form: Tquestion Record 44
Create Report form: Tquiz Class 44
Store Student (TStudentInfo): 44
Report Comment form 45
2.6 Test data during the iterative development 46
2.7 Further data to be used in the post development phase 60
System Testing 60
End User Testing 61
Section 3: Developing the coded solution 62
Stage 1: Database 62
Prototype 1 62
Prototype 2 64
Prototype 3 70
Stage 2: Implementing the “Login page” 73

Prototype 1 73
Prototype 2 80
Stage 3: Implementing the “Signup page” and “Main Menu” 91
Prototype 1 91
Prototype 2 113
Stage 4: Implementing the “Create Class Form” 115
Prototype 1 115
Prototype 2 122
Stage 5: Implementing the “Create Student Form” 125
Prototype 1 125
Prototype 2 133
Prototype 3 137
Stage 6: Implementing the “Update User Form” 139
Prototype 1 139
Prototype 2 156
Stage 7: Implementing the “Select Class Form” 158
Prototype 1 158
Prototype 2 173
Stage 8: Implementing the “Create Report Form” 176
Prototype 1 176
Prototype 2 200
Stage 9: Implementing the “Select Student Form” 209
Prototype 1 209
Stage 10: Hashing and Positioning 223
Prototype 1 223
Prototype 2 232
Section 4: Evaluation 235
4.1 System and End-user testing 235
Usability Testing 235
System testing 237
End-user testing 245
4.2 Evaluate the solution 248
Evaluating the solution to the success criteria… 248
Effectiveness of usability features 250
Further Development 252
Limitations and potential improvements 253

Maintenance Issues.....253

Section 1: Analysis

1.1 Features that make the problem solvable by computational methods

Problem Identification

Teachers in Graveney School are required to write reports. It is reported that it takes around 10 to 15 minutes per teacher, on a single student's report. This means a teacher spends an average of 5 hours, to write reports on a single class. Creating a system to write reports would reduce the time taken for a teacher to write reports. Having this problem solved means a reduction in stress for teachers. It will also save a lot of time, where they can be doing something else, such as creating test for students.

Solving the Problem

The system will take data of a student through generic multiple-choice questions, and automatically create reports and generate teacher's comments. Each multiple-choice question will have around four to five possible selections, with each being a phrase (two to four words), to make it as effortless as possible for the user. There will be a minimal amount of data needed to be input by the user to reduce time taken for a report. The aim is to create a report of an individual, using the selected phrases, for example:

E.g. how is {student name} effort in class?

1) Excellent
2) Good
3) Unsatisfactory
4) Very Poor

This would then be used to create the report

Working to Potential?	Effort in Lessons	Completion of Work
E	E	E

System needs

The features of a computer system that would be required for this solution would be a computer with hardware being able to run a suitable operating System for the software to run on.

The problem is suitable for a computational approach as reports comments are needed to be automatically generated through multiple choice questions. This can be done using an algorithm, which can me coded. Also the reports are electronically sent to the student's parents.

1.2 Suitable stakeholders

The stakeholders for this solution will be Alex Wino and Deepak Patel, both teachers of Graveney School. They will be a good representative of for the solution as they are required to write reports about their students, and the solutions main purpose is to reduce the time it takes them to write reports.

The system will be created for Graveney School. Mr Wino is a teacher of age, within Graveney School, who finds it hard learning how to use computer applications. He will fit the role of making sure the system is user friendly and the user interface is understandable.

Mr Patel is a computer science teacher and he will be one of the stakeholders. He will be managing what the program should do. He will also be discussing what data should be stored. Mr Patel also be the admin of the system, and have access to view the database. When a user forgets their password, Mr Patel will send the password to the email.

My stakeholders are going to represent the users of this system. They are going to show me what the user's needs are. All teachers of Graveney School will be able to use this solution, to create reports.

Information Gathering

Planning

I will be carrying out one to one interview, with my stakeholders, to see why my system is needed for teachers' problems. I will be asking one of the problems of the existing system, and the other what is needed from my system. Later on, I will be setting up a second interview with Mr Wino to analyse the current system in hand, and conclude the major problems with it.

Finally, I will be collecting the problems of the current system my stakeholders, Mr Wino, uses.

Interview – what the system needs (Mr Patel) [18/09/2022]

Why do you want this system?

It's hard to write reports. Sometimes I don't know what to write. I'm not great with English either. On top of that it's really time consuming to write reports, takes up a lot of time at home in which I could be relaxing or could be marking test on time for my students.

What is the main thing the system needs to be able to do?

Reduce the time for us to right reports which are actually good and aren't tedious to read. It should be able to process a well-written comment along with clearly stating the effort and behaviour in lessons for the student.

Is a questionnaire system good idea to write reports?

It is a good idea, however as I said before, it should be able to write student comments in detail. On top of that, there should be a minimum number of questions and they should be short questions so that they could be answered quickly by teachers.

Is the student's reports comments and grades required to be saved on the database (This would allow users to review the report of the student whenever they want.)

Without it being able to save data, it would be really hard to use the program, as it would require users to manually save it, therefore the system has to be able to store data somehow. It will also give teachers access to view the progress in a student's grade over time.

How do you think student data should be stored?

Teachers don't just use one classroom, therefore there has to be a database system that can save data so that it could be accessed on any computer, as if it was to be saved locally, most of the time we wouldn't be able to access the data of students that we have put into the system, and then the system would be useless.

Do you think it is useful for the system being able to create a finalised student report as a pdf?

As much as it would be great to have a system that would allow users to export the reports, the school already has a system which is able to create reports using comments and grades, and the same system is used for other purposes, for example it is used to register for attendance. The system is called Sims. It already has all the students and their parents' details stored in it and has no issues.

A problem that may occur would be that a minority of teachers may not like the system. If the school relied on the program to create the report that is sent straight to the parents, all teachers would be required to use the system so that their evaluation of the students could be included in the reports.

However, an option which is able to export the student's data, which has been inputted by teachers using the system, into a pdf file, could be a good idea as it could save time by having all the students report data needed to be put into sims in one single file. On top of that it could be safely stored elsewhere as a backup.

Should teachers be able to view students' progress in other classes, written by other teachers?

Yes, it will show teachers what subjects the student is strong at and if there is a change in behaviour and effort in other classes or not.

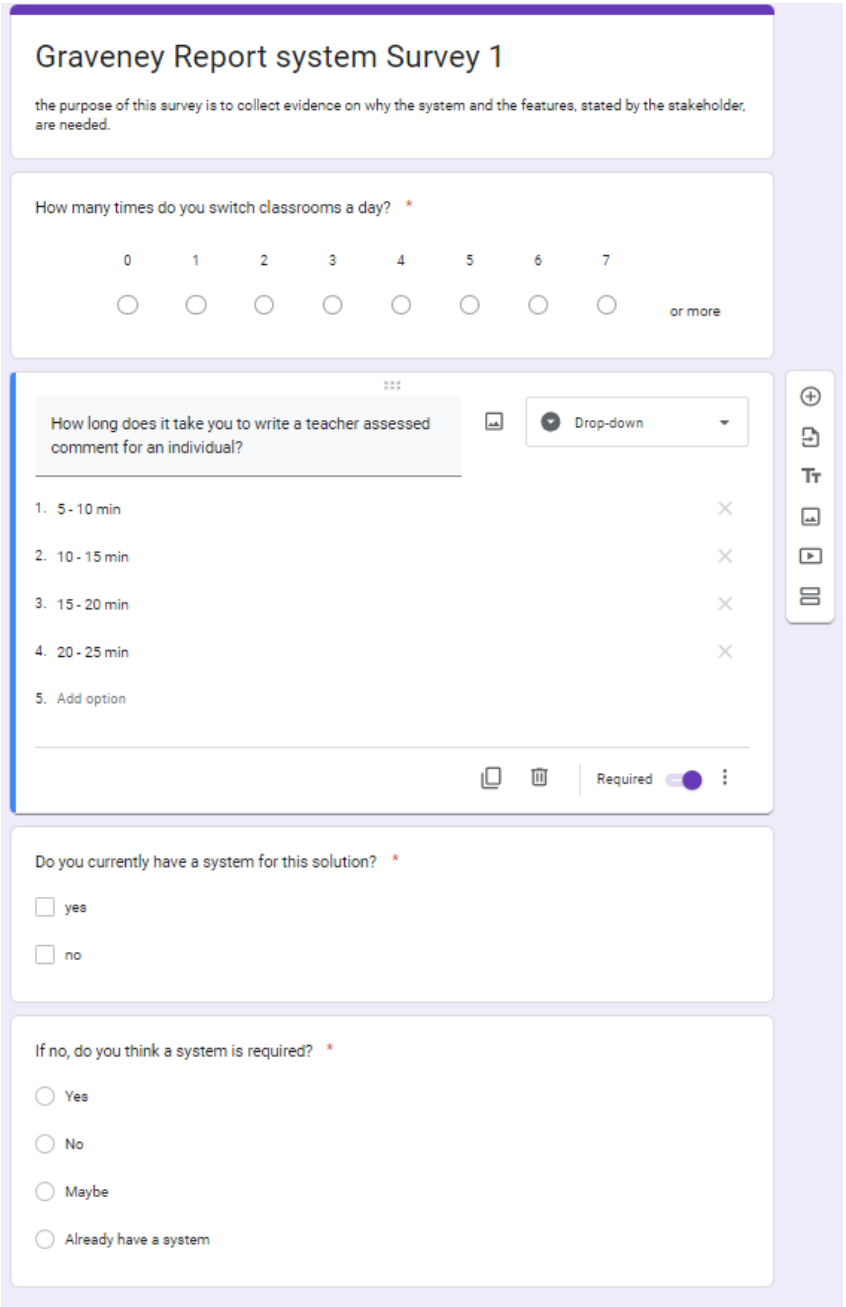
Summary: Program should be easy to use, should be able to generate report using questions and take grades. Users should be able to save data and export reports. Graveney School uses a system called Sims where the report comment will be copied from the system by the user.

Survey

Planning

I will create a small survey in google form, for 10 teachers to complete, to find out if the needed features stated by Mr Patel, is also a solution for other teachers.

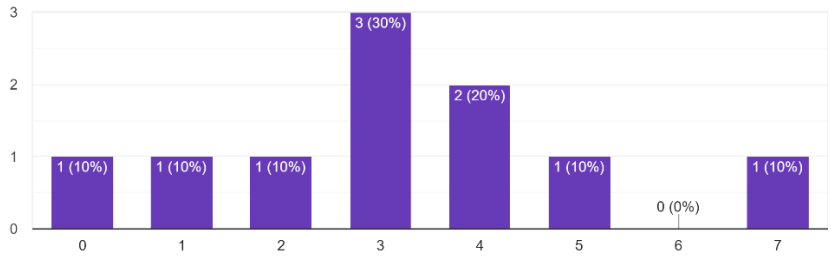
Survey Illustration



Survey results

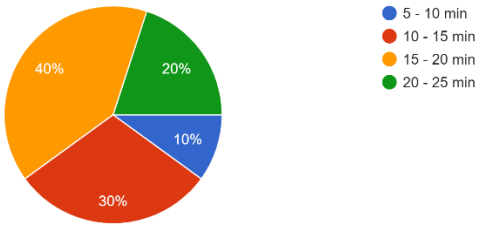
How many times do you switch classrooms a day?

10 responses



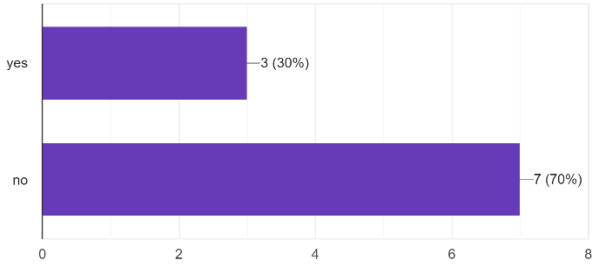
How long does it take you to write a teacher assessed comment for an individual?

10 responses



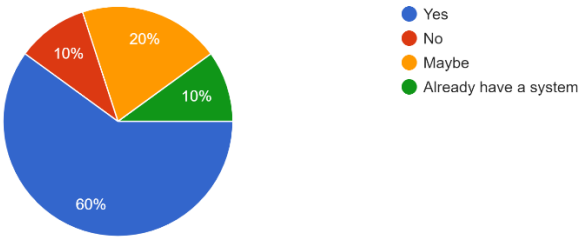
Do you currently have a system for this solution?

10 responses



If no, do you think a system is required?

10 responses



Summary

The results show that most teachers do switch rooms a lot during the day, meaning the feature of a database to store data, so that it can be accessed on any computer, is going to be necessary. They also show that it takes most people 15 – 20 min to write a teacher's assessed comment for a student. Moreover, at least one teacher, who already has a system, said that this system is still required, and overall, most people said it is required.

Email – Design of the system [Mr Wino]

Summary

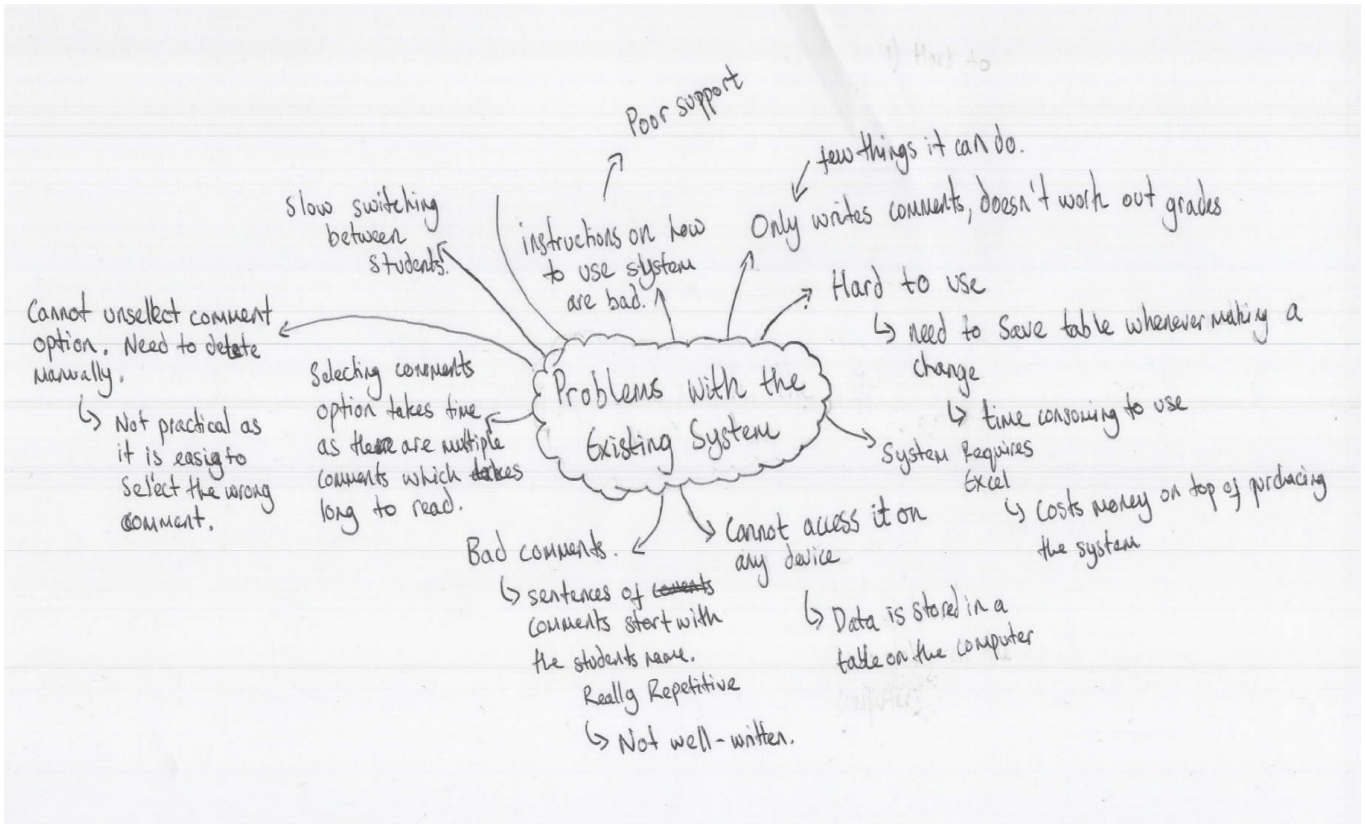
I and Mr Wino communicated via emails on how the system should look like. He requires that the system has a light colour and easy to read font sizes. He also stated that the system should show that it's made for Graveney School.

Information on how student classes work in Graveney

Each student is put into a single form class throughout year 7 to year 11, and year 12 to year 13. The student remains in the same form. Till year 11(if in secondary) or year 13(if in sixth-form). The form classes name's is a/ are alphabetical letter(s). Year 12 and 13's consists of three letters, whereas year 7 to 11's consists of only one letter. Form classes pupils are mixed up for lessons.

Interview – Problem with the Existing System (Mr Wino) [1 to 1 on zoom at 16:30]

These are some problems stated by Mr Wino about the current system. It shows some examples of what I should prevent when creating the system. One problem that I noticed using the system was you have to restart the system every time you save your data.



Graveney School Classes Format

The data of Class Name format was collected so that the system can validate it.

Year 7 to 11	Class Name consists of only one character
Year 12 and 13 (6th Form)	Class Name consists of three characters

Graveney School finishes at year 11 and year 13 for students. A form class of a student's stays the same throughout secondary year 7 – 11 and throughout 6th form year 12 – 13. The name of the class also stays the same throughout those years. In some occasions, the student may find themselves switching class between years 7 – 11.

1.3 Existing solutions

Solution A

Summary

Solution A is a system made in excel. It is a system that is currently used by Mr Wino. Mr Wino also stated the problems with this system in the interview.

How the system works?

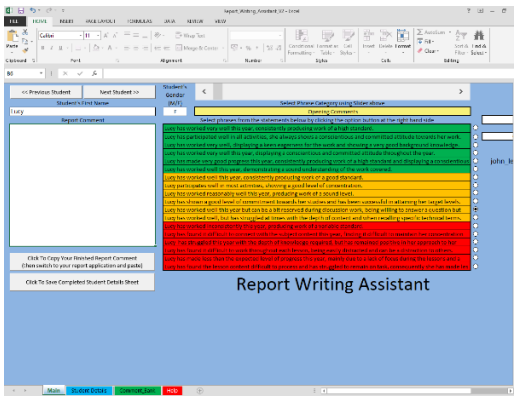
The system displays multiple sentences, for teachers to select and add to the student's comment. It is made to solve the same probem. Its purpose is to make it easier for teachers to write report comments and reduce time taken.

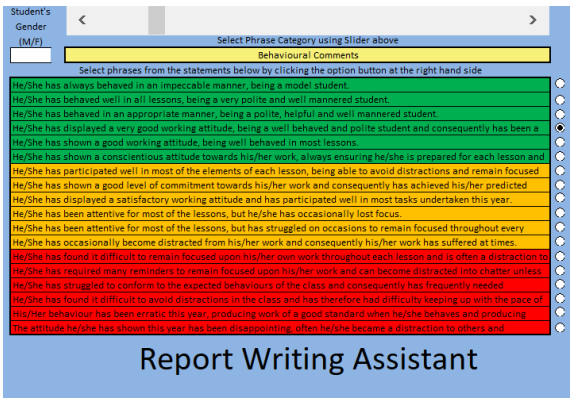
Parts I can apply to my solution

One of the features is that it substitutes the name and pronoun in the sentences as shown in Figure A_1. This is a feature that is required for my system, for the flow of the comment and to direct it to the student its written for.

Order of Comment Selection

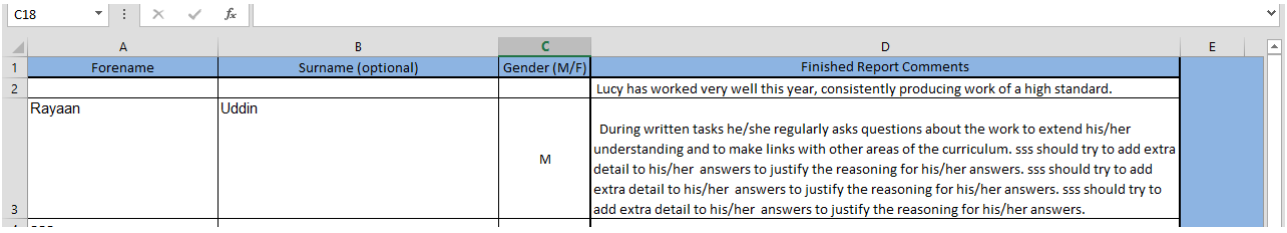
Opening, Behaviour, Progress, Strength, Weakness, Closing comment





Another feature is that it allows users to edit the students report comment. This is a good feature that is required for the solution, as there are some specif things that maybe needed to be added into the report comment, by the teacher, which the program may not be able to generate.

Another feature is that it allows the user to save the students name and comment, allowing them to access and alter it whenever they want to. This is required for the solution as the stakeholder, Mr Patel, stated the system should be implemented in such way that data should be able to be accessed from anywhere.



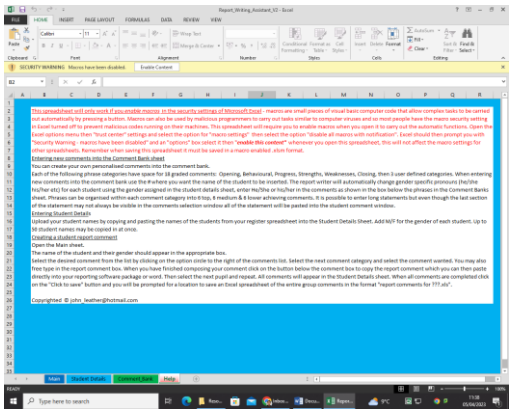
	A	B	C	D	E
1	Forename	Surname (optional)	Gender (M/F)	Finished Report Comments	
2				Lucy has worked very well this year, consistently producing work of a high standard.	
3	Rayaan	Uddin	M	During written tasks he/she regularly asks questions about the work to extend his/her understanding and to make links 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.	

Parts I cannot apply to my solution

One feature is that it sorts the comments using a scroll bar and through colour coordination. This is good feature for this system, however my system will not give an option to choose. Instead, my system in going to focus on generate the whole comment using the answers from each multiple choice question, since the issue is the amount of time it takes teachers to create reports, and this feature opposes the solution.

Another feature is that it has instructions for the users on how to use the system. This is a good feature however my system should be simple enough to understand how to use after one to two uses, since the main

goal is to allow users to create reports with ease and in time restrictions, therefore a help section will not be implemented.



Another feature is that it allows users to change the listed comments permenantly, to make the system more fitted for them. This is a good feature for this system, however as my system will be picking the comments for the user, depending on how they answer, the user cannot modify the comments in the comment bank as it may mean that the algorithm wont work. The user may answer something and the program may generate something totally different due to the user altering the comment. This is not a feature needed as my program will be speciallised for Graveney School, therefore will have all the comments needed for that school.

Solution B

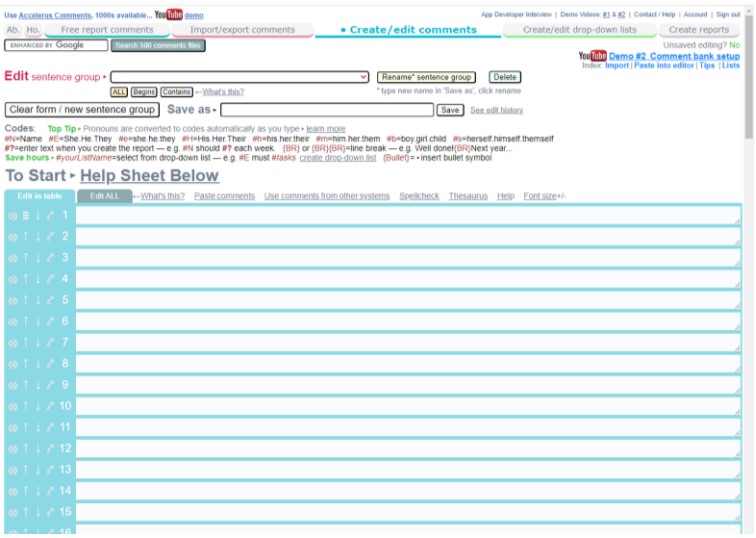
Summary

Solution B is a web application. It allows users to login/ create an account for free and write/ generate reports. It shows a few sentences (less than solution A) to select, but has a lot of drop-down lists to modify the comment. The web application offers you technical support in a variety of ways: video tutorials, written instructions, and a feedback and support option. It doesn't save any of your data.

Parts I can apply to my solution

One feature is that it allows users to replace the name for the pronoun, Vis versa. This is for the user to adjust the flow of the comment, making it easier or nicer to read. My system will aim to try have the generated comment not to have the name or pronoun repeated multiple times simultaneously. I will make use of the pronoun choices given by this solution, in my system.

Another feature is that the user can create an account and login to use the system. This is for the user's data for any personal settings they have chosen to be saved on their account. In this solution, an example of data being saved is personal comments written by the user. My solution will take a similar approach where it will allow users to create accounts and signing to the system, to save the users data and protect it from others accessing it.



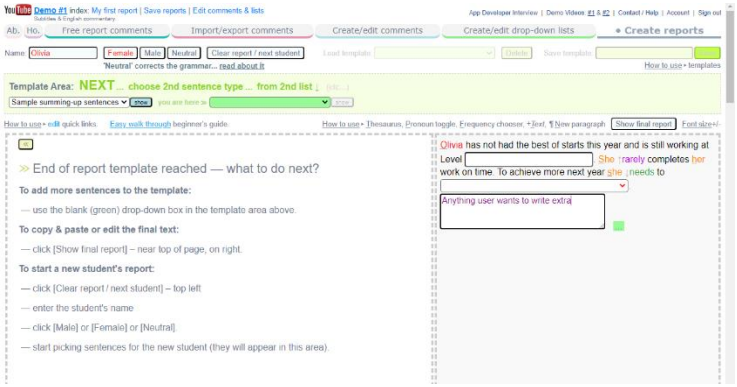
Another feature is that the user's login details can be saved. This is a great feature, which makes the system much easier and user friendly, as well as reducing the time it will take a user to create a report, which is the purpose of the solution. My solution will require this feature for this reason.

Parts I cannot apply to my solution

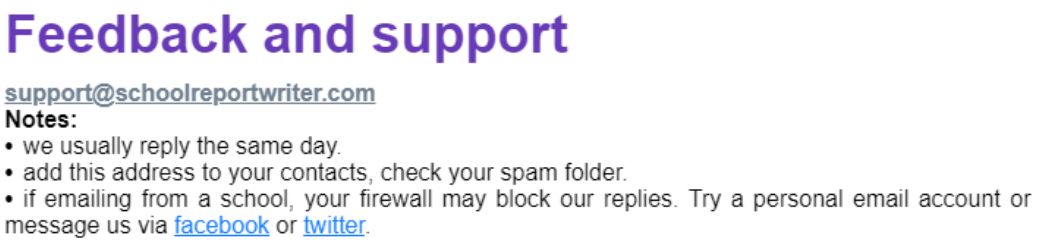
One of the features that it offers a drop-down list, in the report comment, to input things like the academic year the student is in. As solution B doesn't save students data, it is a good way to select key information about the student. My solution will not require this feature as users will be required to create students before writing their reports, meaning my system will know the information required, for the student, beforehand.



Another feature is that the user is required to write half of the comment. The system only gives the beginning and the end of a sentence, and allows the user to write anything about the student in the middle. This does not justify the solution as it then requires more time for the user to right the report. My system should auto generate the whole comment, and give an option to edit the comment.



Another feature is that it provides the user technical support by contacting them. My system will be made for one school, therefore technical support will be available by an IT technician working at the school, who will know on how to use the system, therefore a contact us section within the system will not be needed.



Feedback and support

support@schoolreportwriter.com
Notes:
- we usually reply the same day.
- add this address to your contacts, check your spam folder.
- if emailing from a school, your firewall may block our replies. Try a personal email account or message us via facebook or twitter.

Another feature is that it provides the user a thesaurus on the web application. As my system will generate the comment, I will not require a thesaurus for the user.

Another feature is that it provides the user technical support by contacting them. My system will be made for one school, therefore technical support will be available by an IT technician working at the school, who will know on how to use the system, therefore a contact us section within the system will not be needed.

1.4 Features of the proposed computational solution

The user will be able to register using their details and login to use the system (meeting the requirements 4.1 and 5.1). The system will have a simple design for the program unique to the school (requirement 6). The system has to allow users to register so they can use the system. Having an account is required. This is a feature adapted from solution B from the existing solutions.

The login form will make sure that the user has an account (requirement 4.2). When a user will create an account, their inputted data will be checked that it follows the standard form (requirement 5.2), and checked if the account doesn't already exist (requirement 5.3).

The user will be able to update its name, email and password (requirement 9)

The user will be able to create, delete or edit a student (requirement 11) on their account, once signed in. The user will have an option to read previous reports of the user and delete them if they want too (requirements 15.1 and 15.2). The user should also be able to export reports as PDF (requirement 15.3)

The user will be able to create, delete or edit a class (requirement 13) on their account, once signed in. The user should be able to view all students in an existing class.

The user will be able to create reports by completing a multiple choice quiz (requirement 14).

The user will be able to view a report after it is created (requirement 15.1). This is a feature seen in solution A which will be used in my system as well. The system must also display reports with correct pronouns as seen in solution A and B in existing solutions.

The user will be able to export the report as a pdf to share it to anyone through email, etc. (requirement 15.3). As seen in both Solution A and B from existing solutions, the user must be able to edit the report (requirement 14.3).

The user will be able to share students and its data to other users through the system efficiently (requirement 16).

1.5 Limitations of the proposed solution

One of the limitations is that the email system cannot be implemented as the Graveney School SMTP server could not be set up with the system. An email system would be recommended to verify a user, and make sure all data stored within the system is accurate, following the GDPR.

Another limitation is that users won't be able to have a pdf containing multiple reports, as it will increase the complexity of the program which would increase the time needed for the solution. Due to the time restraint, it will not be possible to implement this in my solution.

Another limitation is that there cannot be a student with the same first and last name in a class. This is because it will require teachers to enter more details about students, to prevent them from getting mixed up, which defeats the purpose of this solution (purpose: to generate reports in a shot given time and save them). Moreover, it is rarely seen, where two students have identical first and last names as each other, therefore it won't have an effect on the users experience of the system.

Another limitation is that the user's login details will not be able to be saved. This was a feature that was seen in solution B, however due to the complexity of the computational solution of this problem, along with the limited development time to meet, it will not be possible to meet this requirement for the proposed solution within the limited time. It is not a significantly important requirement for the solution of the problem, hence to complete the solution within the development time, it will not be met.

1.6 Requirements for the solution including hardware and software requirements

Hardware Requirements for the System

Number	Feature	Explanation and Justification
0	The system database must take as little storage as possible for the solution.	The school does not want the system database to take a lot of storage on the school server. 5GB is given as an average consumption on storage the system should take after a term worth of use.
1	A computer with i5 processor with basic I/O peripherals.	The school already has computers setup which use a keyboard and mouse. The school is going to using its current systems. Each computer has an Intel core i5 2nd generation with 8 GB ram. The system must be compiled for an x86-64 processor.
2	The system must work efficiently with a 720p and 1080p resolution monitor	The school consists of only 720p and 1080p monitors, therefore to make it easier for users, the system forms must be smaller than 720p as then users won't need to repeatedly scroll to use the system.

Software Requirements for the System

Number	Feature	Explanation and Justification
3.1	The solution must be able to run on Windows OS	The schools' computers run on windows operating system and all teachers know how to use it.
3.2	Microsoft access 2016 software and licence	The system must have Microsoft access to access the database, needed for the administration team. The school already has a license and all office applications installed on the server, available to be used by each computer system connected.
3.3	Delphi 2007	The school administrators understand Pascal programming language and the school has the license and software of Delphi 2007 installed.

Features for the system

Number	Feature	Explanation and Justification
4.1	Clients Should be able to login using an email and password	Login used to store client's data within database so that their data can be accessed from anywhere within the school as stated in the interview on 18th September. An email is used to identifying an individual, and password to verify.
4.2	System should inform the client if the entered email or password is incorrect, or if there are missing values.	This validation is to make sure that a user, without an account, cannot just enter random values and break into the program. This is to make it easier for user to identify what they have entered incorrectly.
5.1	Clients should be able to Create an account on the system.	User should be able to create an account to login and use the system. Their data should be stored within the database within the correct fields. The system only requires their first name and last name.
5.2	The system must check if the account already exists.	The system should be able to validate if the account already exists to stop data redundancy, and allow the system to function properly. The Client should be informed, so that they don't assume there is an issue with the system.
5.3	The system must tell the client if the account has been created.	A message to client will confirm to them that they are able to login and use the system.

5.4	The system must tell the client if the account has not been created with the reason.	This is to make sure the client is aware their account has not been created, so they don't assume it has been created and attempt to login. Giving the reason of why their account has not been created makes it easier for the client to correct any invalid data entered.
5.5	The system must make sure the password entered, when creating a new account, is valid.	The password must include at least one digit and have a length greater than 8 for security reasons. This reduces the chance of an account getting hacked.
5.6	The system must make sure the name entered, when creating a new account, is valid.	A name must include only characters of A-Z. This is to make sure the data stored in the database, of system users, is accurate.
6	The program must have a standard appearance with a font large enough to read.	The form should be a light formal color and will have a font, which is big enough for users with weak eyesight to read. The program should also show that it belongs and is created for Graveney School.
7	The system must verify the users email	When a user signs up, the system should be able to verify that the email is valid. This is to make sure valid data is stored within the database, so space is not wasted.
8.1	Once logged in, program must display the main menu.	Main menu will be a form linking multiple other forms with different purposes to solve the problem. The user has to be logged in to use the system.
8.2	Main Menu should display Client's name	This is to indicate that they have been logged into their account. Conformation to user that it's their account they are currently logged into.
8.3	Client should be able to open forms from main menu.	Forms will be linked to the main menu. Each form will meet requirements of the solution (requirements 9… 16).
9.1	Clients should be able to edit their account details	According to GDRP, users have a right to update any incorrect data, and data must be accurate. They may need to update email. User's need to have access to their email's, as if they forget their password, the admin will send the password to only that email address, for security reasons.
9.2	All signup requirements are to be applied to Update account	When updating account, the same validations need to be carried so that when the user does not update their account to an invalid email, name or password.
9.3	Clients should be able to delete their account	Legally the user has rights of what data is stored of them within the system. If the user does not require the system anymore, they should be offered an option to delete their data.
10.1	Clients should be able to create a student	There should be a form allowing users to enter student details and creating a student. This will allow them to create and save reports of students. It will also prevent them to enter the student details every time creating a report.
10.2	The system should inform the client when the student has been created	A response ensures to the user that the student has successfully been created and they can close the form, without losing any data.
10.3	The system should store the Name, Gender and Class of the student.	One of the details needed will be the gender of the student. This is so that the system can generate the report with the correct pronouns of the student.

		Other details needed will be what class the student belongs to, and the name of the student. Identifying a student to a class allows the user to find the student in the system much faster when creating the students report, by giving the user a filter option.
10.4.1	The system should check if the student already exists	The system should be able to check if the student already exists to stop data redundancy, and allow the system to function properly.
10.4.2	Clients should be informed if the student already exists	The Client should be informed, so they know why the student isn't being create, otherwise they may assume it's a problem within the system. It also makes them aware that the student already exists, so they can stop attempting to create it.
10.5.1	The system should validate the student's name.	A name must include only characters of A-Z. This is to make sure the data stored in the database, of system users, is accurate.
10.5.2	Clients should be informed if the name entered is invalid	Clients should be informed to make sure they are aware the name entered is invalid, and is preventing them creating the student, so they can correct the name entered.
10.6	The system should display existing classes, when creating a student.	The system should have a drop-down box, showing all of the existing classes of the user, when creating a student. This makes it easier for the user to create a student, as they can confirm if they have created the class that the student belongs in, rather than typing it in and receiving error relating to the class entered not being found.
10.7	When creating a student, Client should be able to filter class by year.	Allowing the user to filer the classes displayed by year will make it easier for the user to find the class.
11.1	Clients should be able to update student details	The system should have a form where clients can edit students' details. A reason this feature is required is to make sure the data stored is accurate. Students can change classes between year 7 – 11, and users will have to update the changes.
11.2	Clients should be able to delete students	Deleting students will also delete any reports related to the student, wiping all of the student's data within the database. This is to make sure there is no useless data stored, and users have control on their data.
12.1	Clients should be able to create a class.	The system allows users to sort their students in the student's correct class, making it easier for them to view the student, when creating or viewing a report, or updating the student.
12.2	Clients should be informed if the Class already exists.	The system should be able to check if the class already exists to stop data redundancy, and allow the system to function properly (will make it impossible for user to differ between the classes, and won't know which class has the student within it). The Client should be informed, so they know why the class isn't being create, otherwise they may assume it's a problem within the system.
12.3.1	The system should be able to validate Class Name.	Graveney School uses letters as names of class. For year 7-11, it consists of only one letter. For year 12 and 13s, it consists of 3 letters. The system must make sure a valid name is entered.
12.3.2	Clients should be informed if the name entered is invalid	Informing them the name is invalid in the form, will make them know what needs to be corrected. It saves time for the user, as it means they don't have to find the reason of why the class is not being created. It also indicates to them that the class has not been created.
12.4.1	System should make sure a year is selected	The class will not be viewable by the user if a year is not selected. Not having a year for a class will create errors within the system. Useless data should not be kept within the database which will have no use for the system.

12.4.2	Clients should be informed if form is not completed	Informing them will inform them that the class has not been created, and that they need to fill in the form fully to create the class.
13.1	The system should allow clients to update classes details	If they have entered the class details incorrectly, the system should allow them to edit the details rather than deleting the class, as when the class is deleted, so are all the students and reports of the students within the class. Reasons of them needing to update the details could be if the entered name was incorrect, or if a new academic year has been started.
13.2	The system should allow clients to delete classes	The system should delete the class, along with all students and their reports.
13.3	Clients should be given an option to increment the year of all classes by pressing a single button	The system should allow users to increment all class years by one, by pressing a single button, saving them time to update it individually at the beginning of every new academic year. All classes with year = 11 or 13 will be deleted as that's when they finish Graveney school.
14.1	The system should allow clients to create reports on students	The main purpose of the system is to be able to create a student report. It is required for the system to include the students comment, grade, attendance, punctuality, and grade, from the data collected by the teacher/ user, in the report.
14.2	The system should generate a teacher comment	The comment will be generated using the comments in the comment bank table within the database. The system will ask a series of multiple-choice questions to the user, about the student allowing it to generate the comment.
14.3	The system should allow clients to edit the comment after it is generated.	If the comment generated is too generic, and the user wants to write something specific about the student, they should be able to edit the comment.
14.4	The system should make sure all data entered is valid before creating the report	The system should be able to validate the data entered by the user. The attendance and punctuality values must be a valid percentage, and the grade must be a single character, either a digit or alphabet. No data should be null. The system should display a suitable error message if there is a null or invalid value, informing the user on how to correct it.
15.1	The system should allow clients to view the reports	The system should display the report to the user once created. It should also allow users to view any other reports created in the past.
15.2	The system should allow clients to delete reports	The system should make sure that any unused data should be removed from the database. Moreover, the system will not give an option to edit a report, as it will require users to do the questionnaire again. Instead, the system will allow users to delete the report and create a new one.
15.3	The system should allow clients to export students' data as a PDF file	One of the requirements stated in the interview was to allow users to export student's reports, into one PDF file, making it easier submitting the reports of a student into sims.
16	The system should allow users to share students	This will allow users to write reports, which can then be viewable by other users, that the user has shared the student too. This will also prevent the same student being created by separate users which will save data.
17	The system should hash the password	The system should store the hashed password in the database. One reason for this is to prevent data of users being stolen and accounts being hacked.

1.7 Success criteria

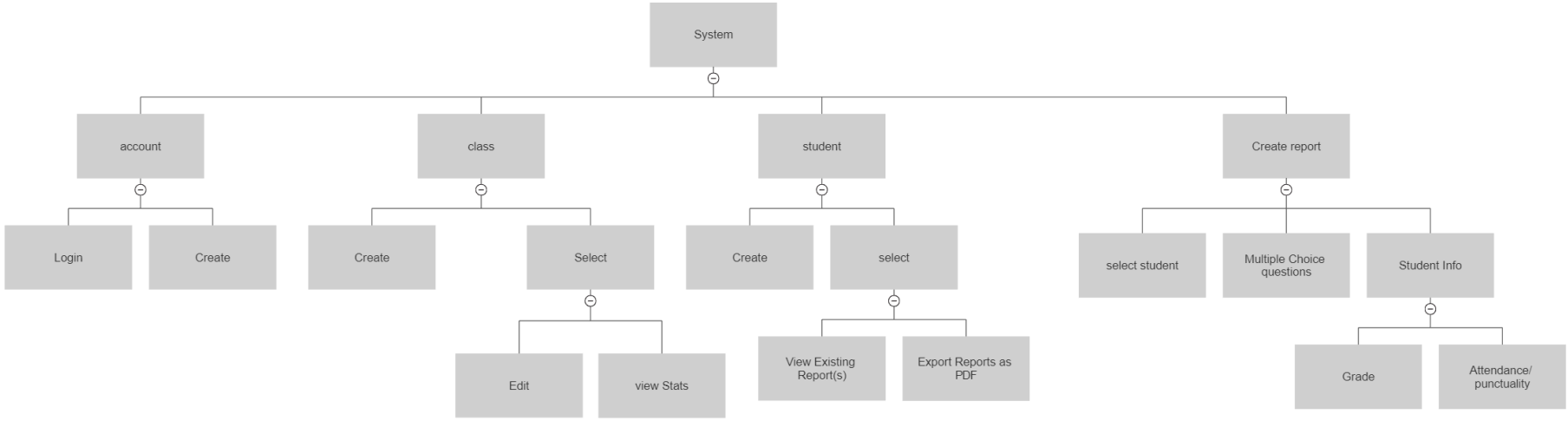
Number	Criteria	Justification
1	User is able to login with their email and password to access their account	Shows a login form for user to enter their details and open main menu. Requirements 4 and 8.1.
2	User must be able to create an account with name, email and password	Shows a Signup form for the user to enter their details for their account. Requirement 5.
3	User must be able to edit or delete their account details	Shows an update account form for the user to edit/ delete their details. Requirement 9. User can open form from main menu, requirement 8.3.
4	User must be able to create a school class with form name and year	User opens create class form, from main menu, where the user can create a class. Requirements 8.3 and 12.
5	User must be able to edit or delete any school class	User opens select class form, from main menu, where the user can edit/ delete a class's detail. Requirements 8.3, 13.1, 13.2.
6	User must be able to increment the year of all school class by pressing a button	User opens select class form, from main menu, where the user can increase all years by one, deleting current year 13 and year 11 classes. Requirement 13.3.
7	User must be able to create a student with student name and class.	User opens create student form, from main menu, where the user can create a student. Requirements 8.3 and 10.
8	User must be able to edit, share or delete any student	User opens select student form, from main menu, where the user can edit, share or delete a student. Requirements 8.3, 16 and 11.
9	User must be able to create a report of a student by completing a multiple choice quiz	User opens create report form from the main menu. The user selects a student and answers the multiple-choice questions. The user is able to create a report. Requirement 14.
10	User must get an option to export the report as a PDF or copy the comment	Users screen displays view report form. The user can either read, copy or export report. Requirement 15.3
11	User must be able to view, edit and delete reports after they have been created	User opens select student form and selects a student. The user clicks on a report and the report is displayed in view report form. The user can also delete that report in select student form. Requirement 15.1 and 15.2

Section 2: Design

2.1 Decomposing the Problem

Illustration of the solution

This is a hierarchy diagram illustrating the solution of the problem. It gives a draft of the key parts needed to solve the problem further broken down. Each rectangle is broken down to a sub problem.



In the account stage of the system, the user must be able to login or signup. The system will open on Login and will allow the user to switch to sign up. At login the user will be able to login to the system allowing the user to use the systems features, such as class, students and create report. At create, the user will be able to create an account to login and use the system.

In the class stage, the user will be able to create a class. Classes will be used to group students into different categories. In the select stage, the system should allow the user to edit the classes or view any statistics of a class required, like the students in a class.

In the student stage of the system, the user will be able to create a student. This will be used to save the reports to an individual student, for the solution. The select section, the user will be able to view reports created of the user in the past and export them as a pdf.

In the create report stage of the system, the user will be able to create reports. The system must auto generate the report and its comments when the user enters the data and completes the multiple choice quiz. There will be 6 multiple choice questions, each of the following topic: behaviour, effort, homework standards, understating the content, work at target, and contribution, generate the students report comment. These topics have been chosen by researching and evaluating solution A seen in section 1.3.

2.2 Structure of the solution

Setting up the Entity Relationship Diagrams

Staff (StfID, FirstName, LastName, LgnID)

Login (LgnID, Email, Password)

Student (SdtID, FirstName, LastName, classID)

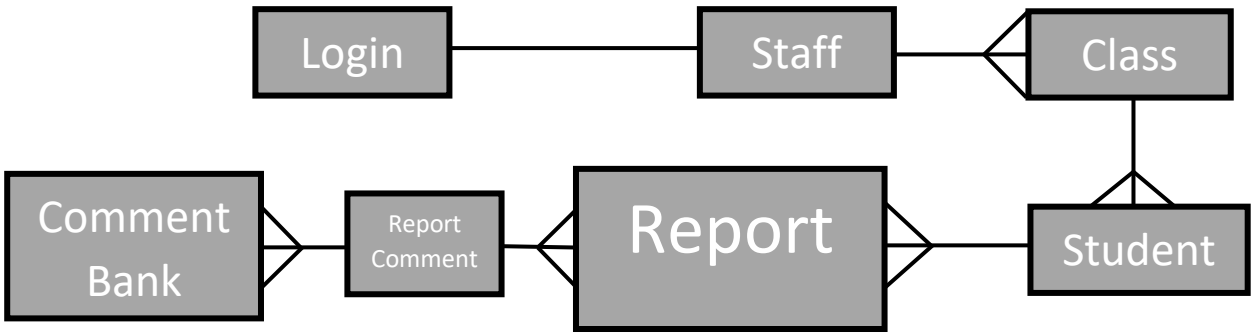
Report Info (RptInfoID, attendance, punctuality, grade, subject)

Class (ClassID, Name, year, StfID) //form class, a student only has one form class

Comment Bank (CmbID, type, comment, level)

Report Comments (RptID, CmbID) //stop other foreign keys from repeating over and over again

Report (RptID, attendance, punctuality, grade, subject,dateCreated, SdtID)



Here there is an illustration of the Relationship between each entity in the database. A user can only have one account hence a one-to-one relationship between Login and Staff. A user can create multiple classes, hence a one-to-many relationship between staff and class. A student can only be in one form class, and a class can have multiple students, hence a one-to-many relationship between class and student. A report is created on a single student, and a user will be able to create multiple reports on a student hence a one-to-many relationship between student and report. A report will consist of a comment. That comment will be made up of parts from the comment bank table. There is a many-to-many relationship between comment bank and report. Report comment table is a linking table.

To conclude

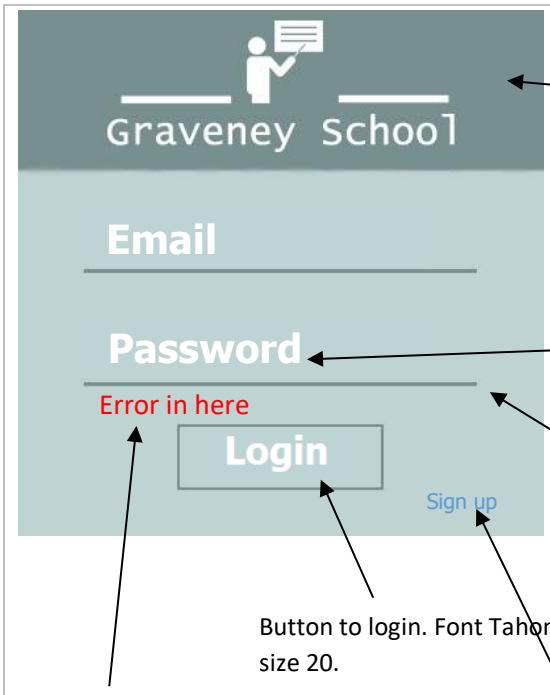
This section sets the structure of the database. It shows the entities of the database and its attributes. It also illustrates and explains the relationships between each entity in the database. This is an initial idea of the database I will be using to solve the problem. Changes may occur later on in implementation, however this sets a main idea of what my system should store and what it should do.

Interface Design

The aesthetics of the Graphical Interface is important as it will be what the user will be interacting with to use the system, therefore should be appealing.

Visualising the designs shows me how the application should look like, and give me a change to make any changes, before the development stage.

Login Form



The header color is #778f8f. Font Lucida Console size 30. System is used by one school.

Text boxes font Tahoma size 15. Both indicate what to input in each box.

The background color is #bed3d3. It's formal and simple. Resolution is 400 by 400. This meets requirement 2.

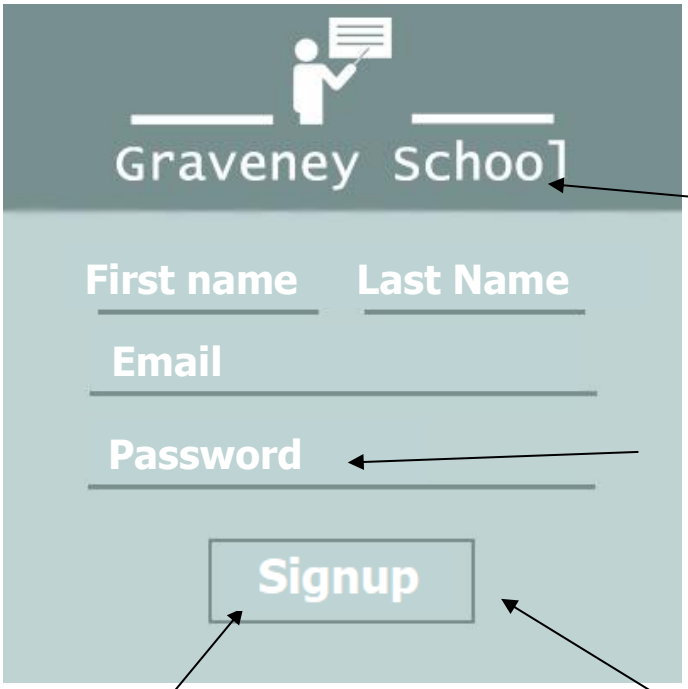
Button to login. Font Tahoma size 20.

Go to sign up page link. Font Tahoma size 8.

The error will be displayed here, e.g., incorrect password. This is to complete requirement 4.2.

The login form is needed to complete requirement 4.1. All components are labelled correctly to its function in the program, making it easier to use the system. It has a light colour and a big enough font to meet requirement 6.

Signup Form



The header color is #778f8f. Font Lucida Console size 30. System is used by one school.

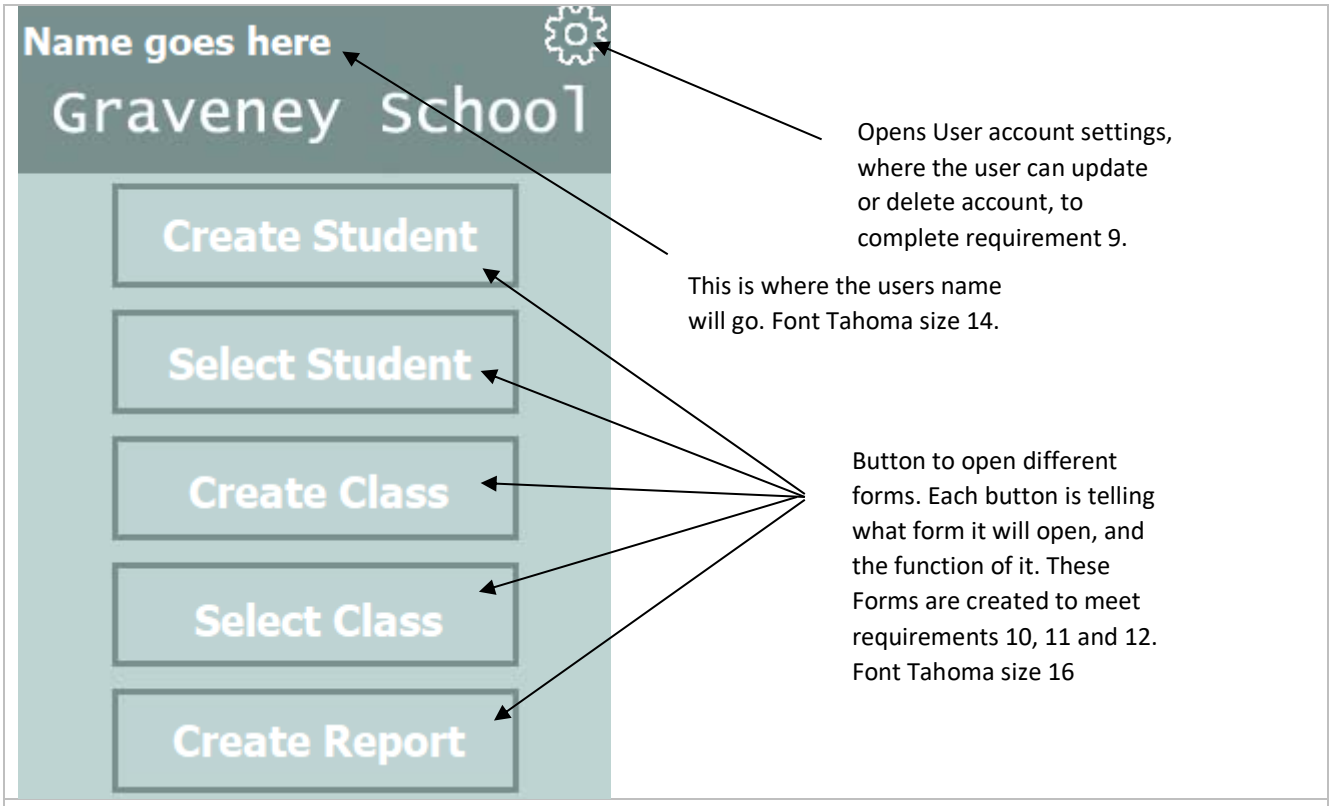
Text boxes font Tahoma size 15. Both indicate what to input in each box.

Button to Sign up. Font Tahoma size 20.

The background color is #bed3d3. It's formal and simple. Resolution is 400 by 400. This meets requirement 2.

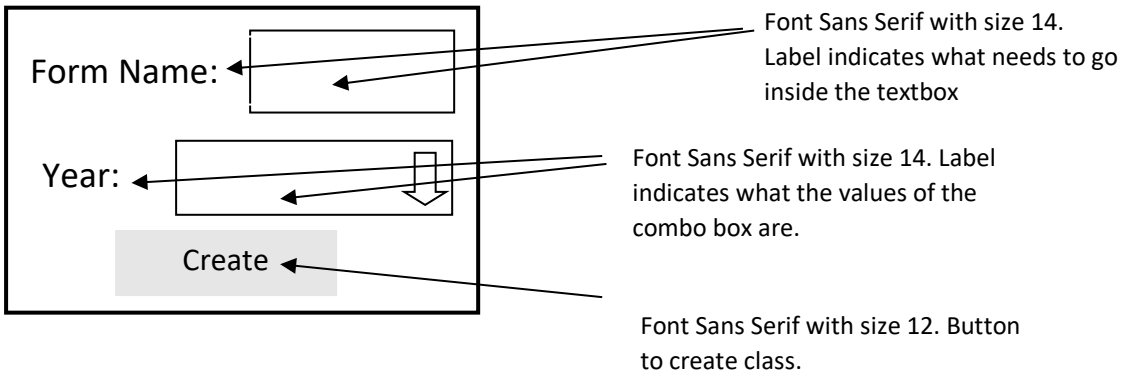
The signup form is needed to complete requirement 5.1. All components are labelled correctly to its function in the program, making it easier to use the system. It has a light colour and a big enough font to meet requirement 6.

Main Menu



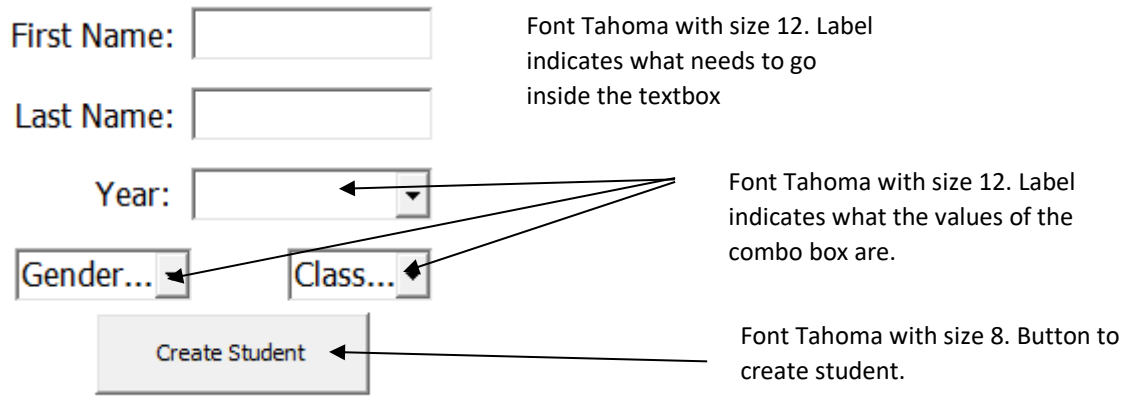
The main menu form is needed to complete multiple requirements, as it links multiple forms the user will need for the solution. It has a light colour and a big enough font to meet requirement 6.

Create Class Form



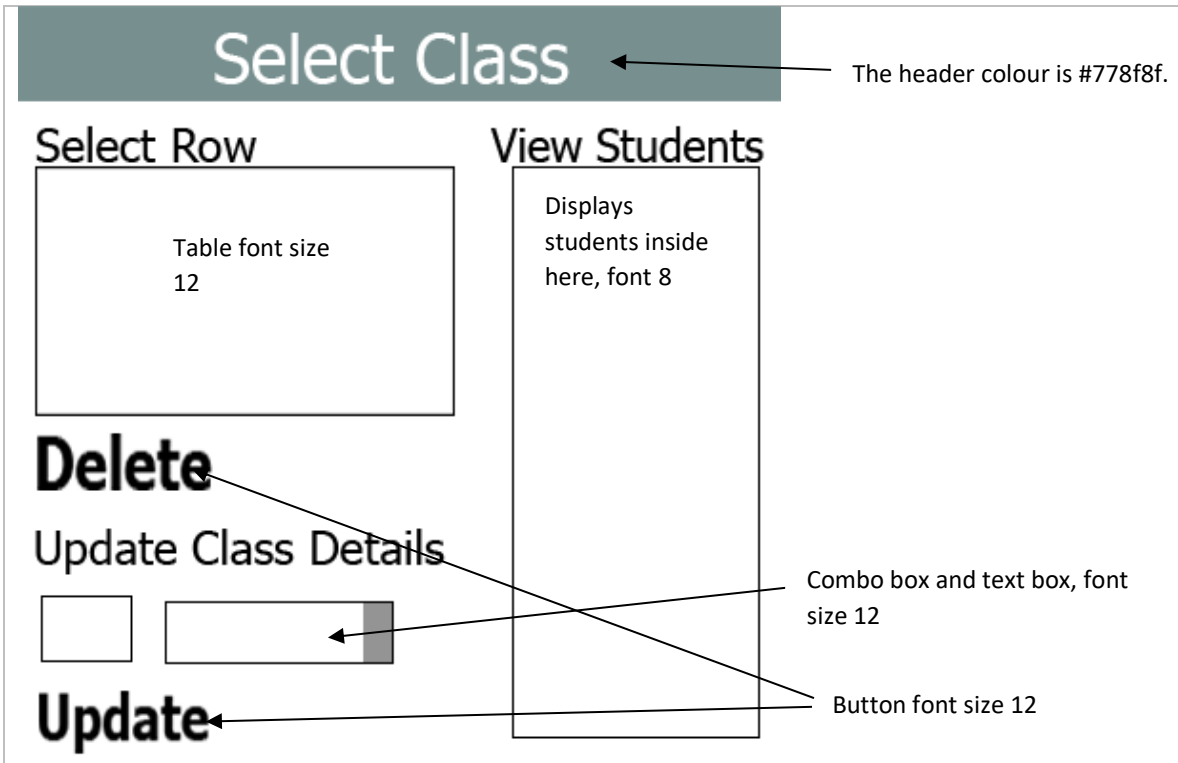
The create Class Form is needed to meet the requirement 12. It has a light colour and a big enough font to meet requirement 6. Resolution is 240 by 150, forms should at least fit a 720p monitor, hence meets requirement 2.

Create Student Form



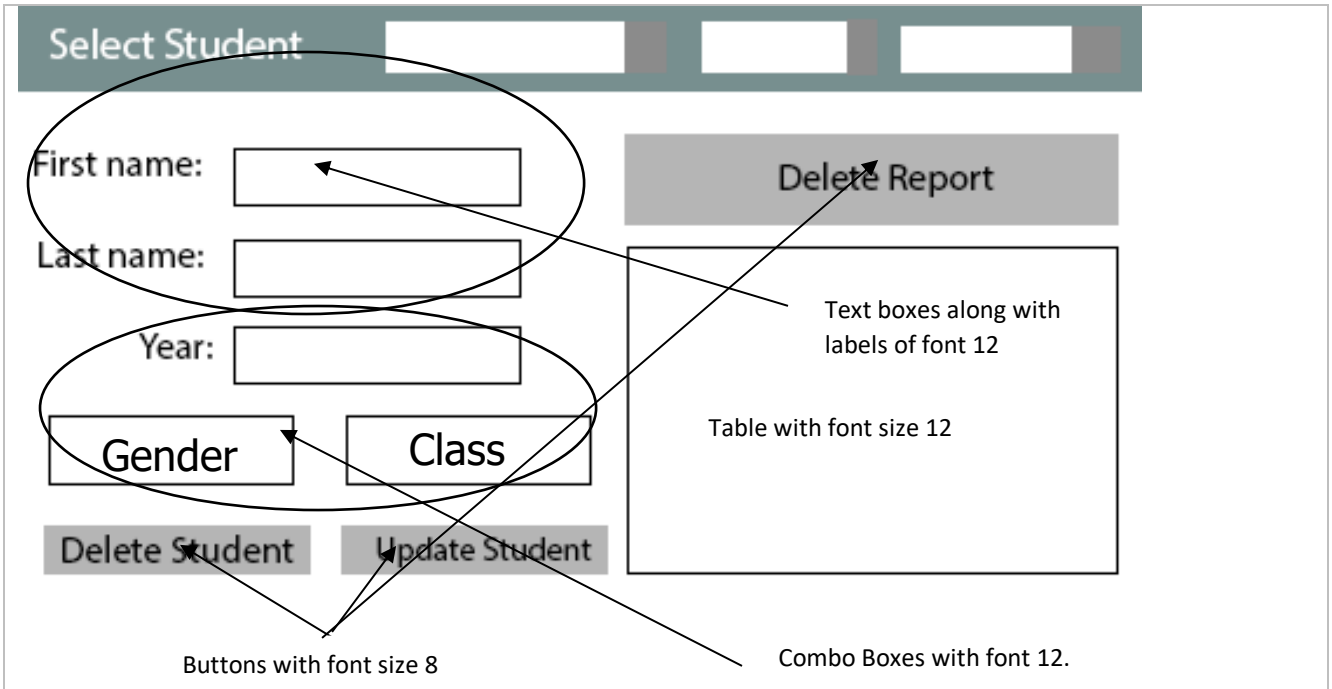
The create student Form is needed to meet the requirement 10. It has a light colour and a big enough font to meet requirement 6. Resolution is 245 by 220, forms should at least fit a 720p monitor, hence meets requirement 2.

Select Class Form



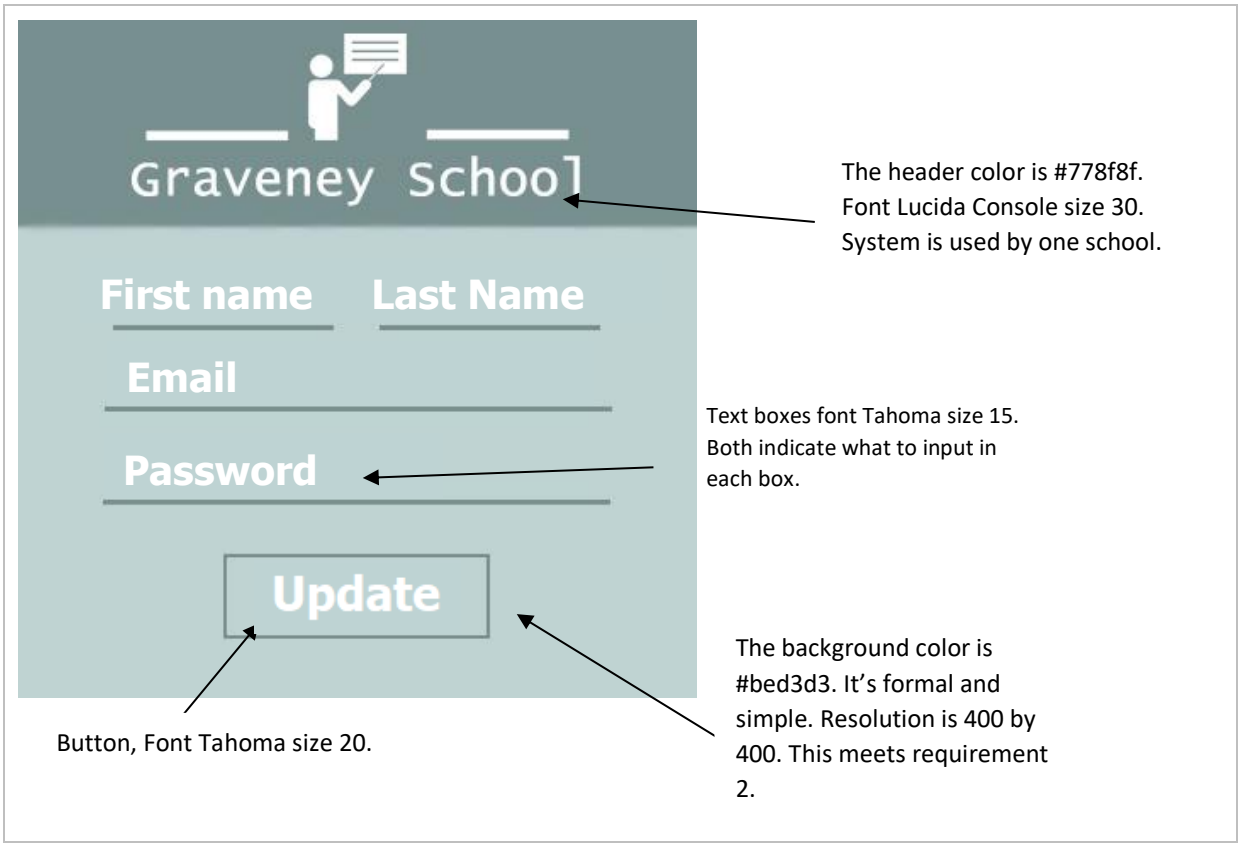
Resolution is 400 by 400, forms should at least fit a 720p monitor, hence meets requirement 2. All font styles are Tahoma. The titles are font 18. It has a light colour and a big enough font to meet requirement 6. The select class form is needed to meet requirements 13.1 and 13.2.

Select Student Form



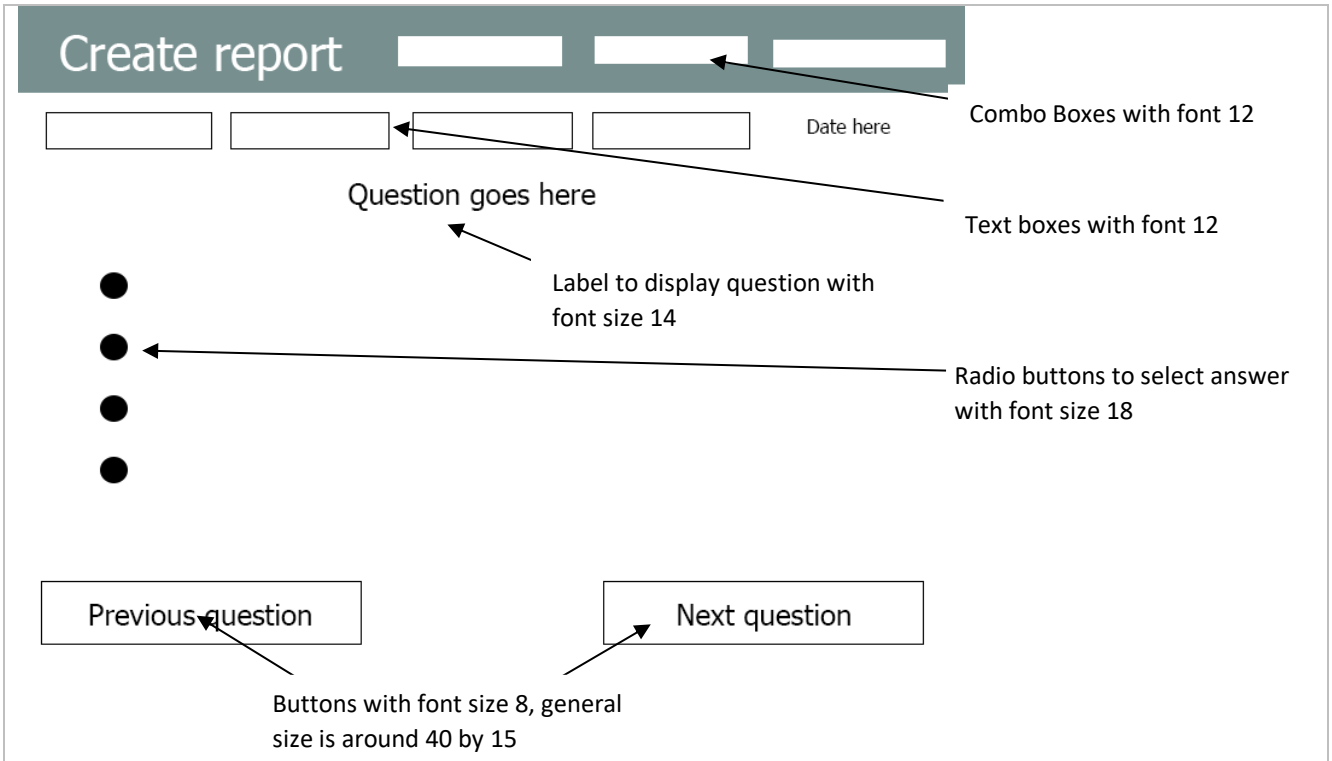
Resolution is 480 by 250, forms should at least fit a 720p monitor, hence meets requirement 2. The header colour is #778f8f. All font styles are Tahoma. The titles are font 18. It has a light colour and a big enough font to meet requirement 6. The select student form is needed to meet requirements 11 and 15.

Update Form



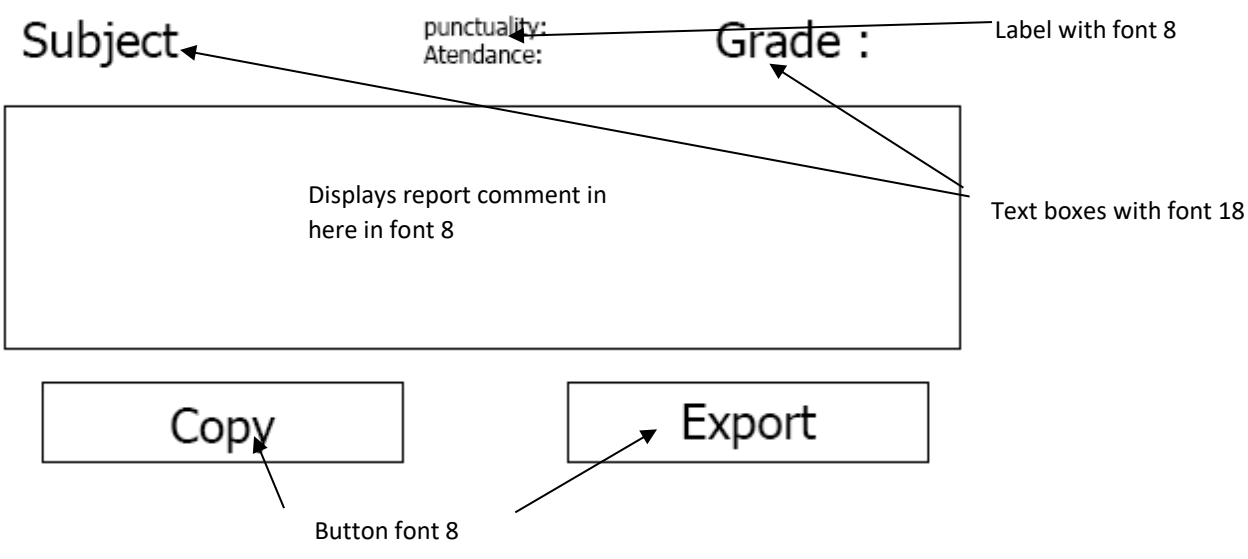
The update form is needed to complete requirement 9. All components are labelled correctly to its function in the program, making it easier to use the system. It has a light colour and a big enough font to meet requirement 6.

Create Report Form



Resolution is 785 by 565, forms should at least fit a 720p monitor, hence meets requirement 2. The header colour is #778f8f. All font styles are Tahoma. The title is font 36. It has a light colour and a big enough font to meet requirement 6. The create report form is needed to meet requirements 14.

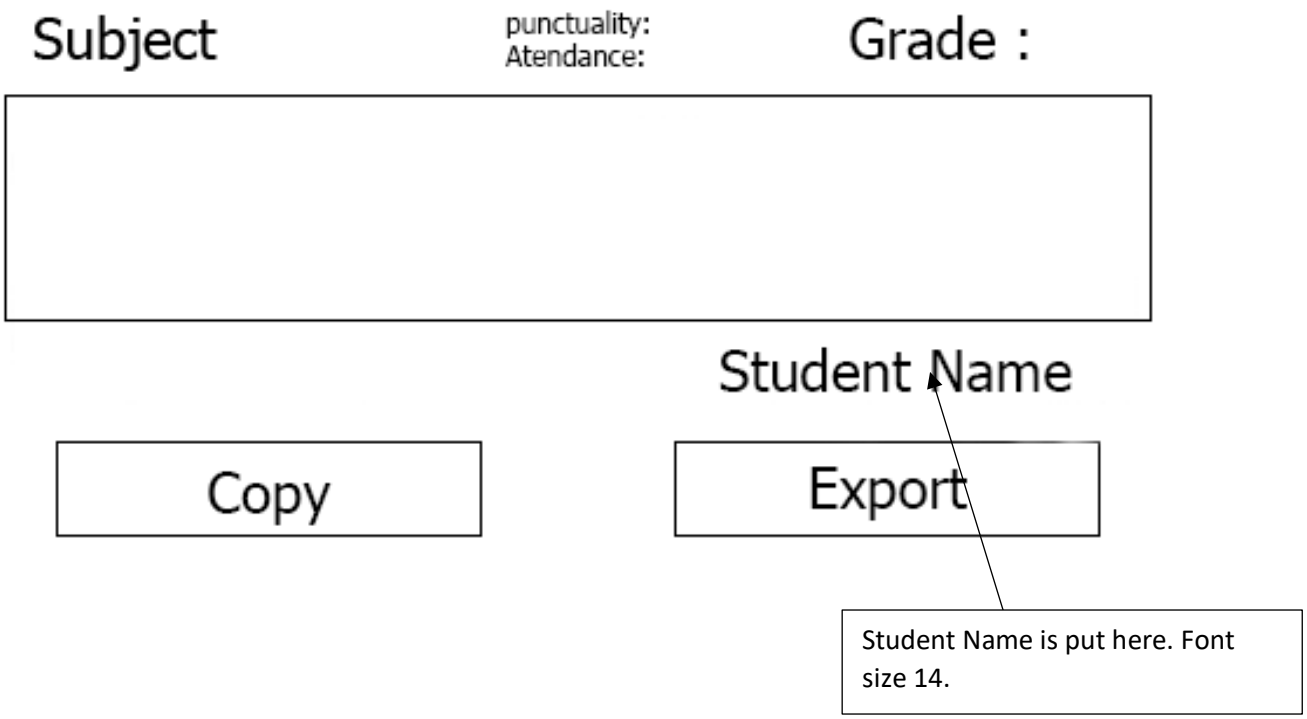
Report Comment Form



Resolution is 500 by 250, forms should at least fit a 720p monitor, hence meets requirement 2. The header colour is #778f8f. All font styles are Tahoma. It has a light colour and a big enough font to meet requirement 6. The report comment form is needed to meet requirements 15.1 and 15.3.

Conclusion

After creating my initial designs for the system, I showed it to my stakeholder, Mr Wino. He was pleased with the design and thought it is suitable for the systems purpose. He did want the report comment form to display the student's name. Here is an updated version of the Report Comment form…



Data dictionary

I am creating a data dictionary to outline the requirements of values in each of the fields, needed in the tables, in the database. This and the entity relationship diagrams set out in 2.2 will make it much easier for me to create the database.

Table name: login

Purpose: This will be used to store the user's login details so they are able to access the program.

Field Name	Required?	Data Type	Sample Data	Validation Rule	Validation Type	Description and key field
LgnID	Yes	AutoNumber	"1"	Must be unique. Must be an integer.	Type check. Presence check.	Primary key
Email	Yes	VarChar	"rayaanuddin@gmail.com"	Must be unique.	Presence check.	-
Password	Yes	VarChar	"Rayaan10"	Must be less or equal to 32 characters.	Length check. Presence check.	-

Table name: Staff

Purpose: This will be used to store the user's name.

Field Name	Required?	Data Type	Sample Data	Validation Rule	Validation Type	Description and key field
StfID	Yes	AutoNumber	"12"	Must be unique. Must be an integer.	Type check. Presence check.	Primary key
FirstName	Yes	VarChar	"Rayaan"	Must be unique.	Presence check.	-
LastName	Yes	VarChar	"Uddin"	Must be greater than 8 characters	Length check. Presence check.	-
LgnID	Yes	Integer	"1"	Must be unique. Must be an integer.	Type check. Presence check.	Foreign Key

Table name: Class

Purpose: This will be used to store the classes created by the user.

Field Name	Required?	Data Type	Sample Data	Validation Rule	Validation Type	Description and key field
classID	Yes	AutoNumber	"3"	Must be unique. Must be an integer.	Type check. Presence check.	Primary key
Name	Yes	VarChar	"B"	Length must be less than or equal to 3 Must only be alphabetical.	Presence check. Length check. Type check.	-
Year	Yes	Integer	"7"	Must be a single integer.	Length check. Presence check. Type check.	-
StfID	Yes	Integer	"12"	Must be unique. Must be an integer.	Type check. Presence check.	Foreign Key

Table name: Student

Purpose: This will be used to store the classes created by the user.

Field Name	Required?	Data Type	Sample Data	Validation Rule	Validation Type	Description and key field
SdtID	Yes	AutoNumber	"6"	Must be unique. Must be an integer.	Type check. Presence check.	Primary key
FirstName	Yes	VarChar	"Elena"	Must only be alphabetical.	Presence check. Length check. Type check.	-
LastName	Yes	VarChar	"Yale"	Must be a single integer.	Length check. Presence check. Type check.	-
Gender	Yes	VarChar	"Female"	Must only be alphabetical.	Type check. Presence check.	
classID	Yes	Integer	"3"	Must be unique. Must be an integer.	Type check. Presence check.	Foreign Key

Table name: Report

Purpose: This will be used to store the details of a report once created by the user.

Field Name	Required?	Data Type	Sample Data	Validation Rule	Validation Type	Description and key field
RptID	Yes	AutoNumber	"6"	Must be unique. Must be an integer.	Type check. Presence check.	Primary key
Attendance	Yes	Integer	"100"	Value must be less than or equal to 100.	Value check. Presence check. Length check. Type check.	-
Punctuality	Yes	Integer	"95"	Value must be less than or equal to 100.	Value check. Presence check. Type check.	-
Grade	Yes	VarChar	"8"	Length must be One.	Presence check. Length check. Type check.	
Gender	Yes	VarChar	"Female"	Must only be alphabetical.	Type check. Presence check.	
DateCreated	Yes	Date/Time	"08/06/2005"	Must be format DD/MM/YYYY	Type check. Presence check. Format Check	
SdtID	Yes	Integer	"6"	Must be unique. Must be an integer.	Type check. Presence check.	Foreign Key

Table name: Comment Bank

Purpose: This will be used to store the comments which the algorithm will select when generating the report.

Field Name	Required?	Data Type	Sample Data	Validation Rule	Validation Type	Description and key field
cmbID	Yes	AutoNumber	"1"	Must be unique. Must be an integer.	Type check. Presence check.	Primary key
type	Yes	VarChar	"Opening"	-	Type check. Presence check.	-
level	Yes	VarChar	"Green"	-	Presence check. Type check.	-
comment	Yes	VarChar	"{name} has made very good progress this year, consistently producing work of a high standard and displaying a conscientious attitude throughout each lesson."	-	Type check. Presence check.	-

Table name: Report Comments

Purpose: This will be used to link the comments to the reports, linking table.

Field Name	Required?	Data Type	Sample Data	Validation Rule	Validation Type	Description and key field
RptID	Yes	Integer	"6"	Must be unique. Must be an integer.	Type check. Presence check.	Composite key
CmbID	Yes	Integer	"1"	Must be unique. Must be an integer.	Type check. Presence check.	

2.3 Algorithms

This section portrays the key procedures and SQL query's, required for the solution, will make it much easier when developing the coded solution, giving me a foundation to build off.

Query to get user details:

SQL Query	Justification
`SELECT StfID, Email, Password, firstname, lastname FROM Login INNER JOIN staff ON (login.LgnID = staff.LgnID) WHERE email = :userEmail`	This will be used by the system to validate a user. When the user attempts to login, the email will be used to find the corresponding account in the database. This will allow users to login to their accounts. Requirements met: 4.1

Query's to write user details

SQL Query	Justification
`INSERT INTO Login(Email, Password) VALUES (:userEmail,:userPassword)`	This stores the users email and password to the table login in the database. It is executed in the signup phase.
`INSERT INTO staff(LgnID,Firstname,Lastname) VALUES (:loginID,:fname,:lname)`	This stores the users name into the staff table and references the record of the users email and password from table login. It is executed in the signup phase.
This allows the user to create an account. Requirements met: 5.1	

Query's to Update User details

SQL Query	Justification
`UPDATE Staff SET FirstName=:fName, LastName=:lName WHERE StfID=:staffID`	This updates the users first and last name in staff table. It is executed in update user detail phase.
`UPDATE Login SET Email= :userEmail, password=:userPassword WHERE LgnID=:loginID`	This updates the users email and password in table login. It is executed in update user detail phase.
This allows the user to update their account details. Requirements met: 9.1	

Query to Create a Class:

SQL Query	Justification
`INSERT INTO class(name, year, StfID VALUES (:cName, :cYear, :staffID)`	This will be used by the system to store the class details into the table class and reference the StfID of the user who created the class. This will allow users to create a class in the system Requirements met: 12.1

Query to display users classes:

SQL Query	Justification

`SELECT name FROM class WHERE year=:cYear and stfID=:staffID`	This will be used by the system to display users existing classes. This will be used in many forms in the system such as, select class, create report, create student form. It is shown in Interface Design most forms have a section where the user selects the class and student on the top. This will allow the system to assign data stored by the user to the correct student.

Query to Update a Class:

SQL Query	**Justification**
`UPDATE Class SET name = :cName, year = :cYear WHERE classID = :classID`	This will be used by the system to update the selected class's record in the table class. This will allow users to update a class's details in the system Requirements met: 13.1

Query to Create Student:

SQL Query	**Justification**
`INSERT INTO student(FirstName, LastName,Gender, ClassID) VALUES (:fName,:lName,:gender,:classID)`	This will be used by the system to add a student to the table student in the database. The class is referenced. This will allow users to create students on their accounts. Requirements met: 10.1

Query to Update Student:

SQL Query	**Justification**
`UPDATE Student SET FirstName = :FirstName, LastName = :LastName , Gender = :Gender, classID = :classID WHERE SdtID = :studentID`	This will be used by the system to update the selected student record in student table. This will allow users to update their student's details. Requirements met: 11.1

Query to Create Report:

SQL Query	**Justification**
`INSERT INTO Report(SdtID, Attendance, Grade, Punctuality, Subject, dateCreated) VALUES (:StudentID, :Attendance, :Grade, :Punctuality, :Subject, :dateCreated)`	This will be used by the system to store the report's details in the table report in the database.
`INSERT INTO [Report Comments](RptID, CmbID) values (:reportID, :reportNumber)`	This will be used by the system to assign the report comments to the report. This is done in the Report Comments table in the database.
This allows the user to create and save a report in the system Requirements met: 14.1	

Query to Get Report:

SQL Query	**Justification**

`SELECT FirstName, LastName, Gender, Attendance, Grade, Punctuality, Subject, DateCreated, Name, Year FROM Class INNER JOIN (Student INNER JOIN Report ON Student.SdtID = Report.SdtID) ON Class.classID = Student.classID WHERE report.RptID = :reportID`	This will be used by the system to get the report details by having the report id.
`SELECT comment FROM [Comment Bank] INNER JOIN [Report Comments] ON [Comment Bank].cmbID = [Report Comments].CmbID WHERE rptID = :reportID ORDER BY [Report Comments].CmbID`	This will be used by the system to get the report comments.
This will be needed in the report comment form, when displaying the report. This will allow users to view the student's reports; needed for the solution. Requirements met: 15.1	

Login Form (Login Algorithm):

```
PROCEDURE DatabaseLogin

  IF FormEmpty THEN
    ADD SQL SELECT StfID, Email, Password, firstname, lastname FROM Login INNER
JOIN staff ON (login.LgnID = staff.LgnID) WHERE email = :userEmail

    IF No Records Found THEN
      PRINT("EMAIL NOT FOUND")
    ELSEIF Password_entered == Password THEN
      Open MainMenuForm
    ELSE
      PRINT("INCORRECT PASSWORD!")
    ENDIF
  ELSE
    PRINT("COMPLETE THE FORM…")
  ENDIF

ENDPROCEDURE
```

This procedure will be used to validate the user, when logging in on login form. This will contribute to the overall solution since system uses accounts as a way to assign and store data individually to each user. Having this procedure allows the user to use the system and have its data secured.
Requirement met: 4

Signup Form (Create Account Algorithm):

```
PROCEDURE WriteAccount

  IF isValidEmail AND isValidName AND isValidPassword THEN

    ADD SQL SELECT LgnID from login where email = :userEmail

    IF No Records Found THEN
      ADD SQL INSERT INTO Login(Email, Password) VALUES (:userEmail,
:userPassword)
      SQL EXECUTE
```

```
      ADD SQL INSERT INTO staff(LgnID,Firstname,Lastname) VALUES
(:loginID,:fname,:lname)
      SQL EXECUTE

      PRINT("ACCOUNT CREATED!")

      CLOSE frmSignup
      OPEN frmLogin
    ELSE
      PRINT("ACCOUNT ALREADY EXISTS!")
  ENDIF

ENDPROCEDURE
```

This procedure will be used to store a user when account is created in signup form. This will contribute to the overall solution since to use the system the user has to have an account to login and store data on.
Requirement met: 5

Email Verification:

```
FUNTION isValidEmail(email)

  charslist = ['_', '-', '.', '0'..'9', 'a'..'z'];
  emailLen = email.LENGTH
  FoundAt = FALSE
  LastPoint = FALSE
  Valid = TRUE

  FOR i=1 TO emailLen
    c=email[i]
    IF c = "@" THEN
      IF FoundAt THEN
        Valid = FALSE
      ELSE
        FoundAt = TRUE
      ENDIF
    ELSEIF c = "." AND FoundAt THEN
      LastPoint = TRUE
    ELSEIF NOT(c in charslist) THEN
      Valid = FALSE
    ENDIF
  NEXT i

  RETURN Valid

ENDFUNCTION
```

This procedure will be used to validate the users email when creating account in signup form. This contributes to the overall solution since one of the requirement was data stored must be accurate.
Requirements met: 7 (updated)

Create Class Form:

```
PROCEDURE CreateClass

  IF FormEmpty THEN
    PRINT("FILL IN THE VALUES TO CREATE CLASS")
  ELSEIF isValidClassName THEN
    ADD SQL SELECT classID FROM class WHERE name = :Name AND stfID = :StaffID
AND year = :Year
    IF No Records Found THEN
      ADD SQL INSERT INTO class(name, year, StfID VALUES (:cName, :cYear,
:staffID)
      SQL EXECUTE
      PRINT("STUDENT CREATED")
    ELSE
      PRINT("CLASS EXISTS…")
    ENDIF
  ELSE
    PRINT("INVALID NAME…")
  ENDIF

ENDPROCEDURE
```

This procedure will be used by the system to create a class. The solution requires classes to store students in. It's a computational feature for the system, which organises students, making it easier for the user to search a student when needed to.
Requirement met: 12

Delete Class Algorithm:

```
PROCEDURE DeleteClass(ClassID)

  ADD SQL SELECT SdtID FROM student WHERE classID = :ClassID

  for i=1 to RecordCount
    DeleteStudentReports(SdtID)
    NEXT SQL
  NEXT i

  ADD SQL DELETE * FROM student WHERE classID = :ClassID
  SQL EXECUTE

  ADD SQL DELETE * FROM Class WHERE classID = :ClassID
  SQL EXECUTE
ENDPROCEDURE
```

This procedure will be used by the system to delete a class. The user must be able to delete its data.
Requirement met: 13.2

Create Student Form:

```
PROCEDURE CreateStudent

  IF FormEmpty THEN
    PRINT("FILL IN THE VALUES TO CREATE STUDENT")
  ELSEIF isValidName THEN
```

```
    ADD SQL SELECT FirstName, LastName FROM student WHERE classID = :ClassID AND
FirstName = :FName AND LastName = :LName AND Gender = :Gender
    IF No Records Found THEN
      ADD SQL INSERT INTO student(FirstName, LastName,Gender, ClassID) VALUES
(:fName,:lName,:gender,:classID)
      SQL EXECUTE
      PRINT("STUDENT CREATED")
    ELSE
      PRINT("STUDENT EXISTS…")
    ENDIF
  ELSE
    PRINT("INVALID NAME…")
  ENDIF

ENDPROCEDURE
```

This procedure will be used by the system to create a student. This contributes to the overall solution of the problem, since it then allows the user to create a report and on a student which is the purpose of the program.
Requirement met: 10

Delete Student Algorithm:

```
PROCEDURE DeleteStudent(StudentID)

  DeleteStudentReports(SdtID)

  ADD SQL DELETE * FROM Student WHERE sdtid = :StudentID
  SQL EXECUTE

ENDPROCEDURE
```

This procedure will be used by the system to delete a student. The user must be able to delete its data.
Requirement met: 11.2

Create Report Form (Creation):

```
PROCEDURE GenerateReport

  IF isReportFormValid THEN
    GenerateReportComment

    ADD SQL INSERT INTO Report(SdtID, Attendance, Grade, Punctuality, Subject,
dateCreated) VALUES (:StudentID, :Attendance, :Grade, :Punctuality, :Subject,
:dateCreated)
    SQL EXECUTE

    ADD SQL SELECT RptID FROM Report WHERE sdtID = :StudentID ORDER BY RptID
DESC

    FOR i=1 TO 6
      ADD SQL INSERT INTO [Report Comments](RptID, CmbID) values (:reportID,
:reportNumber)
      SQL EXECUTE
    NEXT i

    PRINT("REPORT CREATED")
  ENDIF
```

```
ENDPROCEDURE
```

This procedure will be used by the system to create a report. This contributes to the overall solution as it allows users to successfully create valid reports and have them saved in the database, so they can access it whenever they want.
Requirement met: 14

Create Report Form (Validation):

```
FUNCTION isReportFormValid

  IF isFormEmpty THEN
    isInvalid = TRUE
    PRINT("COMPLETE THE FORM…")
  ELSE
    IF Grade.LENGTH = 1 THEN
      IF  NOT(isAlpha(Grade)) THEN
        IF NOT(isInteger(Grade)) THEN
          PRINT("GRADE MUST BE ALPHABETICAL OR NUMERICAL")
          IsInvalid = TRUE
        ENDIF
      ENDIF
    ELSE
      PRINT("GRADE IS A SINGLE CHARACTER")
      IsInvalid = TRUE
    ENDIF

    IF NOT(isAlpha(Subject)) THEN
      PRINT("SUBJECT NAME MUST BE ALPHABETICAL WITH NO SPACES")
      IsInvalid = TRUE
    ELSE
      IF isInteger(Attendance) AND isInteger(Punctuality) THEN
        IF Attendance > 100 OR Attendance < 0 OR Punctuality > 100 OR
Punctuality < 0 THEN
          PRINT("PUNTUALITY OR ATTENDANCE IS INVALID")
          IsInvalid = TRUE
        ENDIF
      ELSE
        PRINT("PUNTUALITY OR ATTENDANCE IS INVALID")
        IsInvalid = TRUE
      ENDIF
    ENDIF
  ENDIF

  IsReportFormValid = NOT(IsInvalid)

ENDFUNCTION
```

This procedure will be used by the system to verify the report being created is valid. This is required for the overall solution of the system, making sure data stored is accurate.
Requirement met: 14.4

Report Comment Form:

```
PROCEDURE displayReport(reportID)

  ADD SQL SELECT FirstName, LastName, Gender, Attendance, Grade, Punctuality,
Subject, DateCreated, Name, Year FROM Class INNER JOIN (Student INNER JOIN
```

```
Report ON Student.SdtID = Report.SdtID) ON Class.classID = Student.classID WHERE
report.RptID = :reportID


  ADD SQL SELECT comment FROM [Comment Bank] INNER JOIN [Report Comments] ON
[Comment Bank].cmbID = [Report Comments].CmbID WHERE rptID = :reportID ORDER BY
[Report Comments].CmbID
  for i = 1 to Record Count
    displaycomment = displaycomment + comment
  end;
  ReplacePronoun(comment)

ENDPROCEDURE
```

This procedure will be used to display the report comment. This is required so the user can view the report once created or any existing stored ones in the database.
Requirement met: 15.1

2.4 Usability features

There are many features, which help the user's experience using the system. The system, when open, displays the login form. There will be a label on the login form, informing users to click and open the sign up form, if they are new to the program. This is an easy way for the user to get started straight away and use the system.

In many cases, the program will close a window when it's known the user does not need it anymore. An example of this will be when the user creates an account, the account window is closed and the login window is opened. This makes the application much more user friendly, as it guides them what to do next to use the system.

Whenever the user will create something, e.g. a student, the system will state it has been created, insuring to the user that the system is working as expected.

When the user goes about to delete an item, e.g. a class, the program will display a confirmation box, just in case of any accidental click. Furthermore, when deleting an item with multiple data stored within it, e.g. a class, which also will delete any students and their data, in the class, the program will inform the user that they are about to delete a x amount of … , e.g. Students.

All forms needed for the solution will be linked onto the main menu, which opens when the user logs in. All buttons in the system will be named after its functionality, making it easier for the user to use the system. All buttons will also be highly intractable, making sure the user knows that it is a button. They either become bigger when hovering on top, or will change colour, emphasising that it is clickable. This makes the system to user friendly and is a great feature for the user interface.

The system will have labels next to each text box informing the user what data is required. When the data entered is invalid, the system will inform the user know what is invalid and why in certain cases. The error message displayed will be written by me when implementing the solution and will be user friendly.

The system will have a consistent font style and theme. It will have the font style of mostly Tahoma, and a size ranging in between 10 and 20, depending on how much data is needed to be fit on the form, since the forms must be able to fit on a 720p screen, requirement 2. The font colour for inputting data will either be white or black, depending on the background colour. Any errors displayed on a form will be red. Each button which opens a form will be a colour changing intractable button, which will change to a dark teal colour from white. All the major forms will have a background colour of teal. Any other form will have a background colour of white. All forms will have a dark teal heading. This will make the system theme link together.

On login and signup, when the user inputs data, the hints of what should be entered inside the text box will disappear. This allows the user to see what they are entering in clearly instead of their view being partially blocked by default text. When the user inputs an invalid email or password on login, the one which is incorrect will go back to its default value, letting the user know again what needs to be inputted.

2.5 Key variables / data structures

Outlining the key variables needed for the program will make it much easier in development stage to know what each variable is used for and what it holds. It will also let me know what data structures need to be implemented into the coded solution.

Having variables and data structures written beforehand will also avoid any repeated code of creation of any classes, etc.

This selection will highlight all the key variables and data structures used by the system and its data types. It will also state if it's a local or global variable.

Login form: TUserData Record

Name	Data Structure/ properties	Global/ Local	Description & Justification
TUserData	Record	Global	This record is assigned to hold the users data. This is required as the system will need to consistently use the user's details in different forms.
TUserData.FirstName	String	Global	This will hold the users first name.
TUserData.LastName	String	Global	This will hold the users last name.
TUserData.Email	String	Global	This will hold the users email.
TUserData.Password	String	Global	This will hold the users password.
TUserData.StaffID	Integer	Global	This will hold the users staff ID, which is used to assign data to a specific user.

Login form

Name	Data Type	Global/ Local	Description & Justification
info	TUserData	Global	This object will hold the users details needed thought the program, once the user logs in. It is a global variable since it will be needed to be used by other units.

Signup form

Name	Data Type	Global/ Local	Description & Justification
Charslist	Char Array	Local	This array will hold all the characters a valid email can contain. This is required to check if email entered by user on signup phase is valid, to meet requirement 7.
FoundAt, LastPoint, Valid	Boolean	Local	These will be used to check for each requirement of an email when validating it. Required to meet requirement 7.

Create Report form: Tquestion Record

Name	Data Type	Global/ Local	Description & Justification
Tquestion	Record	Global	This record holds a question. This is required to be used to efficiently store the question and its number separately as different attributes in an object.
Tquestion.number	Integer	Global	This will hold the question number
Tquestion.text	String	Global	This will hold the question itself

Name	Data Structure	Global/ Local	Description & Justification
Tquestion	Record	Global	This record holds a question. This is required to be used to efficiently store the question and its number separately as different attributes in an object.

Create Report form: Tquiz Class

Name	Data Structure/ properties	Global/ Local	Description & Justification
Tquiz	Class	Global	This class will store the questions data. This is required as it will be the main data structure of a single question in the questionnaire.
Tquiz.question	Tquestion	Global	This will hold the question number and the actual question. The constructor will add it to the object. The constructor will set its default value to… Number is 0, Text is “”.
Tquiz.answerNumber	Integer	Global	This will hold the number of the selected answer i.e. number between 1 and 4. The constructor will add it to the object. The constructor will set its default value to 0.

Store Student (TStudentInfo):

Name	Data structure/ properties	Global/ Local	Description & Justification
TStudentInfo	Class	Global	This class will store the details of a selected student by the user. This will be when the user selects a student to edit, delete, view or create reports on.
TStudentInfo.fName	String	Global	This will hold the first name of the student selected. Will be needed when editing student or when outputting report.

TStudentInfo.lName	String	Global	This will hold the last name of the student selected. Saves time rather than reading it from database multiple time. Will be needed when editing student or when outputting report.
TStudentInfo.year	String	Global	This will hold the year of the student selected. Saves time rather than reading it from database multiple time. Will be needed when editing student.
TStudentInfo.gender	String	Global	This will hold the gender of the student selected. Will be needed when the system displays a report, need to be outputted in the right pronouns.
TStudentInfo.cName	String	Global	This will hold the class name of which the student is in. Saves time rather than reading it from database multiple time. Will be needed when editing student or when outputting report.
TStudentInfo.id	String	Global	This will hold the stdID from the table student in the database which matches to the student selected. This is required for the system to know which student to allocate/ read data to/ from.

Report Comment form

Name	Data Type	Global/ Local	Description & Justification
punctuality, attendance, grade, subject, studentName, dateCreated, classNameYr	string	Global	These variables will be set to hold the reports data which will be fetched in the report comment form. The reason these are global variables are since if the user decides to export the report as a PDF, these variables will be needed in the export report unit.

Name	Data Structure	Global/ Local	Description & Justification
pronoun	Array string	Local	This array will hold the pronouns for the student that the comment will be displayed for. The pronouns replace the place holders of the comments in the database, hence it is required for the solution. If gender is male then [he, his, him] If gender is female then [she, her, her]

2.6 Test data during the iterative development

Test Number	Test Purpose	Test Data	Expected Result	Test Type	Justification
1.1.1	Checking if database can store account details.	Login Table LgnID: 1 Email: adam@gmail.com Password: Adam10 Staff Table StfID: 1 FirstName: Adam LastName: Sandler LgnID: 1	It accepts the values and stores it within the database	Valid	This test is to make sure data can be add to the Table, and Staff Table can reference the Login table.
1.1.2	Checking if referential integrity is enforced between Login and Staff	Changing LgnID from 1 to 2 within staff table where StfID is 1. Note: Record with LgnID 2 doesn't exist	It displays error and doesn't change	Invalid	This test is to make sure referential integrity is enforced.
1.2.1	Checking if database can store class details	Class Table ClassID: 1 Name: F Year: 7 StfID: 1	It accepts the values and stores it within the database	Valid	This test is to make sure data can be add to the Table, and Class Table can reference the Staff table.
1.2.2	Checking if referential integrity is enforced between Staff and Class	Changing StfID from 1 to 2 within staff table where ClassID is 1. Note: Record with StfID 2 doesn't exist	It displays error and doesn't change	Invalid	This test is to make sure referential integrity is enforced.
1.3.1	Checking if database can store student details	Student Table SdtID: 1 FirstName: Josh LastName: Drake Gender: Male ClassID: 1	It accepts the values and stores it within the database	Valid	This test is to make sure data can be add to the Table, and Student Table can reference the Class table.
1.3.2	Checking if referential integrity is enforced between Student and Class	Changing ClassID from 1 to 2 within student table where StfID is 1. Note: Record with StfID 2 doesn't exist	It displays error and doesn't change	Invalid	This test is to make sure referential integrity is enforced.
1.4.1	Checking if database can store a report	Report Table RptID: 1 SdtID: 1 Attendance: 98 Grade: A Punctuality: 100 Subject: English DateCreated: 26/10/2022 Report Comments Store comments 1, 10, 15, 22, and 26 with RptID 1.	It accepts the values and stores it within the database	Valid	This test is to make sure data can be add to the Tables, and report comments is able to reference both report and comment bank.
2.1.1	Checking if system allows users to login	{click login button} Email: adam@gmail.com Password: Adam10	The program will display the "main	Valid	This test to make sure that the login form works by reading the

			menu" and "Login form" will hide.		accounts stored in the database. This is to make sure users are able to login to their accounts (requirement 4.1) and use the system.
2.1.2	Checking if "Login Form" can inform user if email is incorrect.	{click login button} Email: adam123@gm Password: Adam10	The "Login Form" will output 'email not found'.	Invalid	This test is required to make sure requirement 4.2 is met.
2.1.3	Checking if "Login Form" can inform user if password is incorrect.	{click login button} Email: adam@gmail.com Password: Adam10123	The "Login Form" will output 'incorrect password'.	Invalid	This test is required to make sure requirement 4.2 is met.
2.2.1	Checking if "Login Form" can inform user when all values entered are null.	{click login button} Email: Password:	The "Login Form" will output 'complete the form'.	Invalid	This test is to make sure the system informs client why they cannot login to the system, when they haven't filled in the form.
2.2.2	Checking if "Login Form" can inform user when email value entered is null.	{click login button} Email: Password: Adam10	The "Login Form" will output 'complete the form'.	Invalid	
2.2.3	Checking if "Login Form" can inform user when password value entered is null.	{click login button} Email: adam@gmail.com Password:	The "Login Form" will output 'complete the form'.	Invalid	
2.3	Testing button is interactive when mouse on top	Hover mouse on top of login button	When mouse is on top, colour is changed to RGB (119,143,143). When mouse is not on top, colour should be white.	Valid	This test is to make sure the button changes colour to indicate to the user that it's a button and can be clicked. This is a GUI test.
2.4	Checking if login form can open sign up form	Click 'don't have and account, sign up'.	Sign up form should appear and login form should close.	Valid	This test is to make sure that the system allows user to create an account. Without it, the system will be unusable. Needed for requirement 5.1 to be met.
2.5.1	Checking if the system can hide and show hints instead of user deleting it manually.	Click on email edit box Click on Password edit box Click on email edit box	When user clicks on email edit box, 'email' should disappear. When user clicks on password edit box, 'password' should disappear and 'email' should	Valid	This test is to make sure the system always gives a hint to users of what should be entered in the password and email edit boxes, when the value is null, and deletes the hint for the user, instead of having

			appear within email edit box. When user clicks on email edit box, 'email' should disappear, and 'password' should appear within password edit box.		the user deleting it manually.
2.5.2	Making sure it doesn't change the value of email edit box when the value is not null or not 'email' (the hint)	Type 'adam@gmail.com' in Email Edit box. Click on Password Edit box. Click on email edit box.	The value of the email edit box will not change when the user clicks on password edit box or email edit box	Valid	This test is to make sure the system does not change the values of the edit box when the user has entered the value in. If it does change the value, it will not allow the user to login to the system (requirement 4.1 will not be met).
2.5.3	Making sure it doesn't change the value of password edit box when the value is not null or not 'password' (the hint)	Type 'Adam10' in Password Edit box. Click on email edit box. Click on Password Edit box.	The value of the password edit box will not change when the user clicks on email edit box or password edit box.	Valid	
3.1	Testing when all values are valid, is user able to create account	TYPE First Name: Ryan Last Name: Reynold Email: RyanReynold@gmail.com Password: Reynold10 Click on signup button	The signup form should create the account and should display message of account created.	Valid	This test is to make sure the program is able to create an account, for a valid user, and inform them. This test is to check if requirements 5.1 and 5.3 are met.
3.2.1	Testing when all values are valid, apart from First Name	TYPE First Name: Ryan1 Last Name: Reynold Email: RyanRenold@gmail.com Password: Reynold10 Click on signup button	The signup form should not create the account and should display message invalid name.	Invalid	This test is to make sure the program doesn't create an account, when the values the user enters for name is invalid, and user is informed why the account is not created. This test is to make sure requirements 5.6 and 5.4 are met.
3.2.2	Testing when all values are valid, apart from Last Name	TYPE First Name: Ryan Last Name: Reynold##] Email: RyanRenold@gmail.com Password: Reynold10 Click on signup button			
3.3.1	Testing when all values are valid, apart from password, where password does not contain an integer but is greater than 8 characters	TYPE First Name: Ryan Last Name: Reynold Email: RyanRenold@gmail.com Password: ReynoldsTheBest Click on signup button	The signup form should not create the account and should display message invalid password.	Invalid	This test is to make sure the program doesn't create an account when the value the user enters for password is invalid, and user is informed why the account is not created. This test is to make sure
3.3.2	Testing when all values are valid, apart from	TYPE First Name: Ryan Last Name: Reynold			

	password, where password length is less than 8	Email: RyanRenold@gmail.com Password: Reyno1 Click on signup button			requirements 5.5 and 5.4 are met.
3.3.3	Testing when all values are valid, apart from password, where password length is 8	TYPE First Name: Ryan Last Name: Reynold Email: RyanRenold@gmail.com Password: Reynold1 Click on signup button	The signup form should create the account and should display message of account created.	Boundary	This is a boundary test to make sure password is valid when the length is on boundary
3.4.1	Testing Email Verification: When there is no '@' symbol	TYPE First Name: Ryan Last Name: Reynold Email: RyanReynoldgmail.com Password: Reynold10 Click on signup button	The signup form should not create the account and should display message invalid email.	Invalid	This is a test making sure an invalid email address is not accepted by the system. This test is to make sure the user is informed when the account is not created and told why. Requirement 7 met.
3.4.2	Testing Email Verification: When there is more than one '@' symbol	TYPE First Name: Ryan Last Name: Reynold Email: Ryan@Reynold@gmail.com Password: Reynold10 Click on signup button			
3.4.3	Testing Email Verification: When there is no '.' symbol	TYPE First Name: Ryan Last Name: Reynold Email: RyanReynold@gmailcom Password: Reynold10 Click on signup button			
3.4.4	Testing Email Verification: When there is no domain, but there is a full stop within the email	TYPE First Name: Ryan Last Name: Reynold Email: Ryan.Reynold@gmail Password: Reynold10 Click on signup button			
3.4.5	Testing Email Verification: When there is an invalid character	TYPE First Name: Ryan Last Name: Reynold Email: RyanReynold#@gmail.com Password: Reynold10 Click on signup button		Erroneous	
3.5	Checking if system allows user to create an account with a registered email	TYPE First Name: Thomas Last Name: Shelby Email: RyanReynold@gmail.com Password: Thomas10 Click on signup button	The signup form should not create the account and should display message informing user that the account already exists	Erroneous	This is a test to make sure the system does not create accounts of users who already exist. The test is also to make sure the user is informed that they

					already have an account. This test is to make sure requirement 5.2 is met.
3.6	Checking if users name displays on main menu	TYPE Email: RyanReynold@gmail.com Password: Reynold10 Click on Login button	The login form will successfully login with the valid account. It will hide itself and show main menu. The main menu will display “ryan reynold” on the top of the menu.	Valid	This test is to make sure the system is able to display the users name on main menu.
3.7	Testing buttons are interactive when mouse on top	Hover mouse on top of buttons on main menu	When mouse is on top, colour is changed to RGB (119,143,143). When mouse is not on top, colour should be white.	Valid	This test is to make sure the buttons change colour to indicate to the user that they are buttons and can be clicked. This is a GUI test.
4.1	Checking if user can open create class form	Clicks on button Create Class in Main menu	Opens Create Class form	Valid	This test is to make sure the main menu can open the create class form.
4.2.1	Checking if system allows users to create a class for year 7 – 11’s	TYPE Class Name: B Select Year 7 Click on Create Button	Create Class form should create the class and display a message informing the user that the class has been created	Valid	This test is to make sure the user is able to create a class using the system. This test is also to make sure that the class is stored properly in the database by the system. This test is also to make sure that classes can be created with the valid name length. This test is to check if requirement 12.1 is met.
4.2.2	Checking if system allows users to create a class for year 12 and 13’s	TYPE Class Name: BJH Select Year 13 Click on Create Button			
4.3.1	Checking if system can inform user when from is not completed, and does not create account	TYPE Class Name: NULL Does not select a year Click on Create Button	Create Class form should not create the class and display message, fill in the values to create class	Invalid	This test is to make sure that a class is not created with empty values. This is to make sure the system functions properly and classes can be used to sort students. This test is to check if requirement 12.4 is met.
4.3.2		TYPE Class Name: NULL Select year 7 Click on Create Button			
4.3.3		TYPE Class Name: B Does not select a year			

		Click on Create Button			
4.4.1	Checking if system can inform user when class name is invalid, where year 7 class name length is not one	TYPE Class Name: BL Select Year 7 Click on Create Button	Create Class form should not create the class and inform user class name is invalid.	Boundary	This test is to make sure the system does not create a class with an invalid name. The system is created for Graveney school and must follow the Graveney school procedure. The test is to check if requirement 12.3 is met.
4.4.2	Checking if system can inform user when class name is invalid, where year 11 class name length is not one	TYPE Class Name: BL Select Year 11 Click on Create Button			
4.4.3	Checking if system can inform user class name is invalid, where year 12 class name length is not three	TYPE Class Name: B Select Year 12 Click on Create Button			
4.4.4		TYPE Class Name: BLRT Select Year 12 Click on Create Button			
4.4.5	Checking if system can inform user class name is invalid, where year 13 class name length is not three	TYPE Class Name: B Select Year 13 Click on Create Button			
4.4.6		TYPE Class Name: BLRT Select Year 13 Click on Create Button			
4.4.7	Checking if system rejects a class name with an integer	TYPE Class Name: BL1 Select Year 13 Click on Create Button		Erroneous	
4.5	Checking if system rejects a class that already exists	TYPE Class Name: B Select Year 7 Click on Create Button	Create Class form should not create the class and inform user that the class already exists	Invalid	This test is to make sure the system can stop data redundancy. Having multiple same classes will make it impossible for users to know which student is in which class. This test is to check if requirement 12.2 is met.
5.1	Checking if form can add class names to combo box, corresponding to the correct year	1) Select Year 7, View Classes 2) Select Year 13, View Classes 3) Select Year 8, View Classes	1) Shows Class B 2) Shows Classes LMN and BJH 3) Shows No Classes	Valid	This test is to make sure the form correctly shows classes corresponding to the year selected by the user, using the database. The class is

					needed to be stored to meet requirement 10.3. The test is to check if requirements 10.6 and 10.7 are met.
5.2	Checking if user can create a student	TYPE First Name: Ben Last Name: Yale Select Year 7 Class B Select Gender Male Click on Create Student Button	Create Student form should create the student and inform user that the student has been created	Valid	This test is to make sure the system is able to accept valid values and create a student. This test is to make sure the system can store the student within the database. This test is to make sure the system can inform the user that the account has been created. This test is to check if requirements 10.1, 10.2, and 10.3 are met.
5.3	Checking if user can create a student, when student already exists	1)TYPE First Name: Ben Last Name: Yale Select Year 7 Class B Select Gender Male Click on Create Student Button 2)TYPE First Name: ben Last Name: yale Select Year 7 Class B Select Gender Male Click on Create Student Button	Create Student form should not create the student and inform user that the student already exists	Invalid	This test is to make sure the system can prevent redundancy. This test is to make sure the system can check if the student already exists in the database. This test is to make sure the system does not create the student when it already exists, and it can inform the user. This test is also to make sure that the Create Student form is not case sensitive. This test is to check if requirement 10.4 is met.
5.4.1	Checking if form is filled out	Click on Create Student Button	Create Student Form should not create the student and should inform user the form is not filled out.	Invalid	This test is to make sure the system collects all values it needs before creating the class, so the system can function properly. The test is to make sure the user is informed if the form is not fully filled out. This test is to ensure requirement 10.3 is met.
5.4.2	Checking if form is filled out, when first name is null	TYPE First Name: Last Name: Yale Select Year 7 Class B Select Gender Male Click on Create Student Button			
5.4.3	Checking if form is filled out, when last name is null	TYPE First Name: Ben Last Name: Select Year 7 Class B Select Gender Male			

		Click on Create Student Button			
5.4.4	Checking if form is filled out, when class is not selected	TYPE First Name: Ben Last Name: Yale Select Year 7 Select Gender Male Click on Create Student Button			
5.4.5	Checking if form is filled out, when year is not selected	TYPE First Name: Ben Last Name: Yale Select Gender Male Click on Create Student Button			
5.4.6	Checking if form is filled out, when gender is not selected	TYPE First Name: Ben Last Name: Yale Select Year 7 Class B Click on Create Student Button			
5.5.1	Testing when all values are valid, apart from First Name	TYPE First Name: Ben1 Last Name: Yale Select Year 7 Class B Click on Create Student Button	The Create Student form should not create the student and should display message invalid name.	Invalid	This test is to make sure the program doesn't create a student, when the values of the student's name is invalid, and user is informed why the account is not created. This test is to make sure requirement 10.5 is met.
5.5.2	Testing when all values are valid, apart from Last Name	TYPE First Name: Ben Last Name: Yale##] Select Year 7 Class B Click on Create Student Button			
5.6	Testing cboselectyear in system, when entering an alphabet	Enter Letter 'A' in cboselectyear	The Create Student form should just display select year	Valid	Whilst using the system, an error was found with the event of cboyear being changed. The system prompts an error whenever an alphabetical letter was entered within the cbo. This was not user friendly.
6.1	Testing main menu opening Update Setting Form	Click on setting button on main menu	Main menu should display the form with the users values displayed within the edit boxes, apart from password	Valid	This test is to make sure the main menu can open the update setting form.
6.2	Updating account when values are valid	Not changing any details Typing the original password: Adam1010 Click Update Button	The Update Student Form should update the email and change it within the database. It should	Valid	This test is to make sure the user is able to update their accounts when they are only

			then display a message to user regarding the changes taken place.		changing one detail, such as first name. This test is to make sure the validation of checking if the account already exist does not deny updating it to the user's email which he is logged into, as the user will not always change his email every time updating his account. This test can also verify that when the values are valid, the account can be updated. This test is to check if requirement 9.1 is met.
6.3.1	Updating user Email, where email is valid	Change Email value to Adam1@gmail.com Other values keep same… Name: Adam Sandler Type Password: Adam1010 Click Update Button	The Update Student Form should update the email and change it within the database. It should then display a message to user regarding the changes taken place.	Valid	This test is to make sure the system allows users to update their details. This test is to check if requirement 9.1 is met.
6.3.2	Updating user Email, where email is registered to another user	Change Email value to ryanrynold@gmail.com Other values keep same… Name: Adam Sandler Type Password: Adam1010 Click Update Button	The Update Student Form should not update the account. It should inform the user that the account already exists.	Valid	The test is to check if validation of checking if an account already exists to another user works. It is important as if two users have the same email, one of them will not be able to login. This test is to check if requirement 9.2 is met.
6.3.3	Updating user Email, where email is invalid	Change Email value to Adamgmail.com Other values keep same… Name: Adam Sandler Type Password: Adam1010 Click Update Button	The Update Student Form should not update the account. It should inform the user that the email entered is invalid.	Invalid	This test is to check the validation of email from signup is working. If the test is a success, it means the email validation is working, since the algorithm has already been tested in Stage 3. This test is to check if requirement 9.2 is met.
6.4.1	Updating Name, where First Name is invalid	Keep Values the same Name: Adam1 Sandler Type Password: Adam1010	The Update Student Form should not update the account. It should inform the	Invalid	This test is to check the validation of name from signup is working with update setting

		Click Update Button	user that the name entered is invalid.		form. If the test is a success, it means the name validation is working, since the algorithm has already been tested in Stage 3. This test is to check if requirement 9.2 is met.
6.4.2	Updating Name, where Last Name is invalid	Keep Values the same Name: Adam Sandler### Type Password: Adam1010 Click Update Button		Invalid	
6.5	Updating Password where password is invalid	Keep Values the same Type Password: Adam Click Update Button	The Update Student Form should not update the account. It should inform the user that the password entered is invalid.	Invalid	This test is to check the validation of password from signup is working with update setting form. If the test is a success, it means the password validation is working, since the algorithm has already been tested in Stage 3. This test is to check if requirement 9.2 is met.
6.6.1	Deleting an account	Click on delete button Then Yes	The Update Setting form should delete all data of the user. It should delete all reports, students, classes, and the users' staff record and login.	Valid	This test is to make sure the user is able to delete his account. This test is to make sure all records that contain the user's data are deleted from the database. This test is to check if requirement 9.3 is met.
6.6.2	Deleting an account, then cancelling it	Click on delete button Then No	The Update Setting form should not delete the account and shall just return back to the form.	Valid	This test is to make sure the system is able to prevent accidental deletion of an account. The purpose of the test is to confirm the No button does not delete the account.
6.7	Correct Casing for string	In signup form… First Name: jEfE Last Name: willIAMS Email: Jeff@gmail.com Password: Jeff1000 Press signup	The System should make sure the name in database is stored as Jefe Williams	Valid	This test is to make sure the system is able to correct invalid casing of strings entered by the user. This is to make the program look more professional. It is a requirement made by the stakeholder after testing stage 6.
7.1	Checking if Select Class Form can display students within the selected class	In Select Class form… Select class Name L Year 8	The Select Class form should display the 2 students, Holly and Andy	Valid	This test is to make sure the system is able to show the user which students are in the class. This feature is

			which are in the class.		needed as if user is deleting the class, he can check what data he is also deleting with it.
7.2	Checking if table can be filtered by year	In Select Class Form… Select filter by year 1) Select year 7 2) Select year 9	1) The Select Class form's table should display Class A 2) The Select Class form's table should display no classes	valid	This test is to make sure the form is able to change the query so that it only displays the year selected by the user. The purpose of this is to make the system as user friendly as possible.
7.3	Checking if deleting class functions properly	In Select Class Form… Select Class A Year 7 Press Delete Press yes	The Select Class form should delete the class along with all the students and their reports.	Valid	This test is to see if the Select Class form is able to allocate and delete all the data of the class, correctly. This test is to make sure requirement 13.2 is met.
7.4.1	Checking if class details can be updated with 2 different valid values	In Select Class Form… Select Class L Year 8 Class name K Class year 11 Update	The Select Class form should update the class and notify the user. It should then also refresh the table.	Valid	This test is to make sure the user is able to update the classes' values. This test is to make sure requirement 13.1 is met.
7.4.2	Checking if class details can be updated with year remaining the same	In Select Class Form… Select Class K Year 11 Class name L update			
7.4.3	Checking if class details can be updated with Name remaining the same	In Select Class Form… Select Class L Year 11 Class year 8 update			
7.5	Result when class name entered is invalid and user attempts to update it.	In Select Class Form… Select Class L Year 8 Class name K1 Class year 13 update	The select class form should not update the class, and inform the user.	Invalid	This test is to make sure the system does not allow any invalid data within the database. If the form is able to verify this correctly, then it will verify and find all invalid possible values, since the form uses the same procedure used by Create Class Form, which was already been tested.
7.6	Result when class is attempted to be updated with null value	In Select Class Form… Select Class L Year 8 Class name Class year Any Year update			
7.7	Checking if system will not update the class when the class already exists under the user	In Select Class Form… Select Class L Year 8 Class name: A Class year 8 update	The select class form should not update the class, and inform the user the class already exists	Invalid	This test is to make sure the system can prevent data redundancy. This is not only for the efficiency of the system, but also to make sure the

					solution (system) functions properly.
7.8	Making sure the data of one user isn't accessed by the other.	In Select Class Form… Select Class L Year 8 Class name: R Class year 11 update	The Select Class form should update the class and notify the user. It should then also refresh the table.	Valid	The test is to make sure the user is able to update their class to the same values of another user's class. The system must not search data of another user.
7.9	Checking if the btnIncrementClasses functions properly.	Pressing the button named btnIncrementClasses	The Update Setting form should delete all year 11s and 13 classes and increment the class year by 1.	valid	The purpose of this test is to make sure the user is able to update their data for a new academic year by pressing a single button. This test is to make sure requirement 13.3 is met.
8.1	Checking if the form can generate a valid report	Year 12|LAL|Mellad Bagbhan Grade B| Attendance 97| Punctuality 92|Computing 1a|2b|3b|4a|5a|6b	The create report form should be able to create a report by following the algorithm, and save it in the database.	Valid	The test is to make sure the user is able to create a report and the system can generate an accurate comment from the data collected. This test is to make sure requirement 14.1 and 14.2 is met.
8.2.1	Checking if the form makes sure from is complete before creating report	Year 9|A|[unselected] Other values selected randomly	The form should be able to inform the user that the form is not complete, by displaying a user-friendly error message. The form should not create a report.	Invalid	These tests are to make sure the form is complete and all values entered by the user are valid before creating the report. This test is to make sure a suitable error message is displayed to user if there is an issue with the data they have inputted. This test is to check if requirement 14.4 has been met.
8.2.2		Keep only grade empty Keep only attendance empty Keep only punctuality empty Keep only subject empty Keep question 6 empty			
8.3	Checking if the form is able to validate the attendance and punctuality	1) Having punctuality as 123 2) Having attendance as -12 3) Having attendance as A12	The form should display message to user, informing the value is invalid and not create the report	Invalid	
8.4	Checking if the form is able to validate the Grade entered	1) Inputting value 10 2) Inputting value AC 3) Inputting value #		Invalid	
8.5	Checking if the forms previous question button functions as intended too	Pressing previous button on form	The form should display the previous question	Valid	This test is to make sure the user is able to go back and forth when answering the multiple choice

					questions, without losing data or having any issues.
8.6	Checking if the form can display the report correctly	Year 12|LAL|Mellad Bagbhan Grade B| Attendance 97| Punctuality 92|Computing 1a|2b|3b|4a|5a|6b	The create report form should be able to create a report and open and display the form on report comment form.	Valid	This test is to make sure the system can display the students report once generated. This test is to make sure requirement 15.1 is met.
8.7.1	Checking if the form allows the user to copy the comment	Press Copy and paste it in Notepad	The report comment form should allow the user to copy the comment to clipboard	Valid	This test is to make sure the feature for users to copy the report comment to their computers clipboard functions as intended to.
8.7.2	Checking if the form allows the user to export the report as a pdf	Export report and open	The report comment form should allow the user to export the report.	Valid	The test is to make sure the user can export the report where ever they want to as a pdf file. This is to check if requirement 15.3 is met.
8.8	Checking if the form allows the user to input the grade A*	Create a report with A* as grade	The create report form should create the report.	Valid	The test is to make sure a report can contain the grade A* as requested from the stakeholders.
9.1	Checking if the form allows user to view and open students reports	Select Student Alex Rider and double click its existing report displayed	The select student form should display all the reports of a student when the user selects a student. It should allow user to open and view the form.	Valid	The test is to make sure the user can view a student's existing (already created) reports, which is needed to be implemented for requirement 15.1 to be met.
9.2.1	Checking if the form allows user to edit the student's values	Update Year 7 Class G Alex Rider to a Female with name Alexia Rider. Press the update button	The select student form should update the details of the students in the database and should show a message of success to the user.	Valid	The test is to make sure the user can Update students details using the select student form, with valid data. This is needed for requirement 11.1 to be met.
9.2.2	Checking result of attempting to update student values when form is not complete	1) Press Update student when no student is selected. 2) Select Year 8 class G Alex Rider, and update student with first name and last name null.	The Select Student form should output a message to the user indicating the form is not complete.	Invalid	This test is to make sure data of students is always accurately stored. This is needed for requirement 11.1 to be met.

		3) Select Year 8 class G Alex Rider, and update student with class and year unselected. 4) Select Year 8 class G Alex Rider, and update student with class unselected.			
9.2.3	Checking result of attempting to update to an existing student	Select Year 7, class G, Alex Rider. Attempt to update the class to Year 8 G, where there is another student by the name Alex Rider.	The select student form should display a message informing the user the student already exists, and not update the student's details	Invalid	This test is to avoid data redundancy in database. Having duplicates of students will cause logical errors in the system, hence this condition is made.
9.3.1	Checking if user is able to delete a report	Select Year 7, class G, Alex Rider and press delete report button with report selected	The select student form should delete the report of the student, and show a message when the report has been deleted. The table of reports being displayed should then be updated.	Valid	This test is to make sure the client can delete reports in select student form. Requirement 15.2.
9.3.2	Checking systems output when user attempts to delete a report, when no report is selected	Select Year 7, class G, Alex Rider and press delete report button with no report selected	The select student form should display a message informing the user that no reports are selected.	Invalid	This test is to make sure the system error is trapped and a user friendly message is displayed to the user.
9.4.1	Checking if user can delete student	Select student Alex Rider in Year 7, Class G and press delete student button	The select student form deletes the student and show a message to the user, informing them that the user has been deleted.	Valid	This test is to make sure the user is able to delete its students. Needed for requirement 11.2.
9.4.2	Checking systems output when user attempts to delete a student, when no student is selected	Press Delete Student button (with no student selected)	The select student form informs the student no student is selected.	Invalid	This test is to make sure the system error is trapped and a user friendly message is displayed to the user.
10.1	Checking if form positions in the centre	Open Login form Open Create Report Form	Both forms should display in the middle of the screen	Valid	This test is to make sure the system is able to work on any resolution monitor. Required for requirement 2 to be met.

2.7 Further data to be used in the post development phase

System Testing

Test Number	Test Purpose	Test Data	Expected Result	Justification
S1	Process of creating and logging into an account	Account Name: Rayaan Uddin Email: 6250@graveney.org Password: Rayaan10	Account is created. User can login with account.	This test is to make sure the user can use the account feature. This is a final test to make sure the system functions properly, taking perspective from client side.
S2	Process of updating an account	Update Email: 6250@graveney.com Password: RayaanU1	Account is updated.	
S3	Process of deleting an account	Pressing the delete account button	Account is deleted.	
S4	Process of setting up students and classes, and sharing students	Class Name: B Year: 7 Student Name: Nick Solo Class : (One created above) Gender: Male Share user to… 6250@graveney.org	Class and student are created. User shares student to another user.	This test is to make sure the user can use the student feature of the system. This is a final test to make sure the system functions properly, taking perspective from client side.
S5	Process of updating a class	Update class B: Year 8 Name to: C	Class is updated.	This test is to make sure the user can use the all class features of the system. This is a final test to make sure the system functions properly, taking perspective from client side.
S6	Process of creating and copying a report	Report Okay, Excellent, Okay, Some, At, Some Grade: 4 Attendance: 92 Punctuality: 86 Subject: English	Report is created. Comment is copied and pasted onto a word document.	This test is to make sure the user can create a report and copy the comment. This is a final test to make sure the system functions properly, taking perspective from client side.
S7	Process of updating a student and exporting a report	Student Name to: Nickoletta Solo Gender to: Female	Student is updated. The report is exported.	This test is to make sure the student selection feature is working. This is a final test to make sure the system functions properly, taking perspective from client side.
S8	Process of deleting a report	Pressing delete button on each form for each function	Report is deleted.	This test is to make sure the user is able to delete its data. This is a final test to make sure the system functions properly, taking perspective from client side.
S9	Process of deleting a student		Student is deleted.	

S10	Process of deleting a class		Class is deleted.	

End User Testing

This testing is required to check if the ned-user can use the program effectively. To carry this out, I am going to create a questionnaire for the stakeholders to complete. In the questionnaire I will get feedback about if the system. Any issues with the solution will be evaluated and will be resolved by an explanation of the next steps needed to be taken place for the system. In addition to collecting unsatisfactory results, it will also outline all the usability features.

The questions that will be asked are…

Does the login screen show up when the program is ran?
Does the login screen have an option to sign up?
When entered an invalid email, name, or password, does the system inform why the account is not being created, helping you out?
Does the program allow you to create your account?
Does the system allow you to create the same account?
Are you able to login with that account?
Does the program inform you if you got the password or email wrong when logging in?
Are you able to edit your details and login with the updated details?
Does the program allow you to open the create class window from the main menu?
Are you able to create a class?
Do you get a message when you have entered the wrong class name?
Does the program allow you to open the create student window from the main menu?
Are you able to create a student?
Does the program allow you to create the same student or class?
Does the program allow you to open the create report window from the main menu?
Does the program allow you to create a report and generate the comment for you?
Does the program inform you if you have entered any details incorrectly, and tell you which one you have?
Does the program display the report after it has been generated?
Does the program allow you to copy the report, and can you paste it anywhere on your computer?
Does the program allow you to export the report as a pdf?
Does the program display what location you want to export the report to?
Can you see all details of the report on the pdf file?
Does the program allow you to open the select class window from the main menu?
Does the program allow you to edit the class's details?
Does the program display all the students in the class?
Does the program allow you to open the select student window from the main menu?
Does the program allow you to edit a student's details?
Does the program allow you to open created reports?
Does the program allow you to increment all class years by one from the update settings window?
Do all windows open in the centre of your screen?
Does the system prevent you deleting any of you data accidentally?
Is the system designed to your like?
How easy is it to navigate the system on a scale of 1 to 10?
What would oppose you from using the system as a teacher?
Did you experience any faults in the system?

Section 3: Developing the coded solution

Stage 1: Database

The database structure, shown in 2.2, is being implemented.

Prototype 1

Here are the tables in design view created in MS Access.

Screenshot and Explanation of "Staff" _table

Field Name	Data Type	Description (Optional)
StfID	AutoNumber	Primary key
FirstName	Short Text	Users First name
LastName	Short Text	Users Last name
LgnID	Number	Foreign key

The table *Staff* stores the name of the user. It references the staff's login details (email and password). StfID is an important primary key for the system as it will be used to identify who the data in the database belongs to.

Screenshot and Explanation of "Login" _table

Field Name	Data Type	Description (Optional)
LgnID	AutoNumber	Primary Key
Email	Long Text	
Password	Short Text	

The table *Login* stores the user's email and password, which is to verify the user, allowing them to use the system.

Screenshot and Explanation of "Class" _table

Field Name	Data Type	Description (Optional)
classID	AutoNumber	Primary Key
Name	Short Text	Form Name
Year	Number	Academic Year
StfID	Number	Foreign Key

The table *Class* stores the all the school classes created by users. Users will not be able to create two of the same class. This is to prevent data redundancy. Year 7 to 11 Form name will be one character whereas Year 12 and 13, will be three characters. This is an example of a rule of Graveney School. Each class will only be viewable by the user who created it. StfID is used to identify whose data it is.

Screenshot and Explanation of "Student" _table

Field Name	Data Type	Description (Optional)
SdtID	AutoNumber	Primary Key
FirstName	Short Text	Student's First Name
LastName	Short Text	Student's Last Name
Gender	Short Text	Student's Gender
classID	Number	Student only belongs to single class

The table *Student* stores the all the students created by users. Users will not be able to create two of the same students in the same class. It references the class of which the student belongs to, which then links it to the user, who the data belongs to.

Screenshot and Explanation of "Comment Bank" _table

Field Name	Data Type	Description (Optional)
cmbID	AutoNumber	Primary Key
type	Short Text	Opening, Behaviour,Progress, Strength, Weaknesses, Closing Statements
level	Short Text	Green=Good, Amber=Okay, Red=Bad
comment	Long Text	Text

The table *Comment Bank* stores the all the sub-comments which will be used to generate the student's full comment. It saves data by preventing these phrases being stored multiple times.

Screenshot and Explanation of "Report" _table

Field Name	Data Type	Description (Optional)
RptID	AutoNumber	Primary Key
SdtID	Number	Foriegn key
Attendance	Number	
Grade	Short Text	
Punctuality	Number	
Subject	Short Text	
DateCreated	Date/Time	

The table *Report* stores the all the reports that the user creates.

Screenshot and Explanation of "Report Comments" _table

Field Name	Data Type
RptID	Number
CmbID	Number

The table Report *Comments* stores the comment of the report by linking the report to each comment it has from the *comment bank* table.

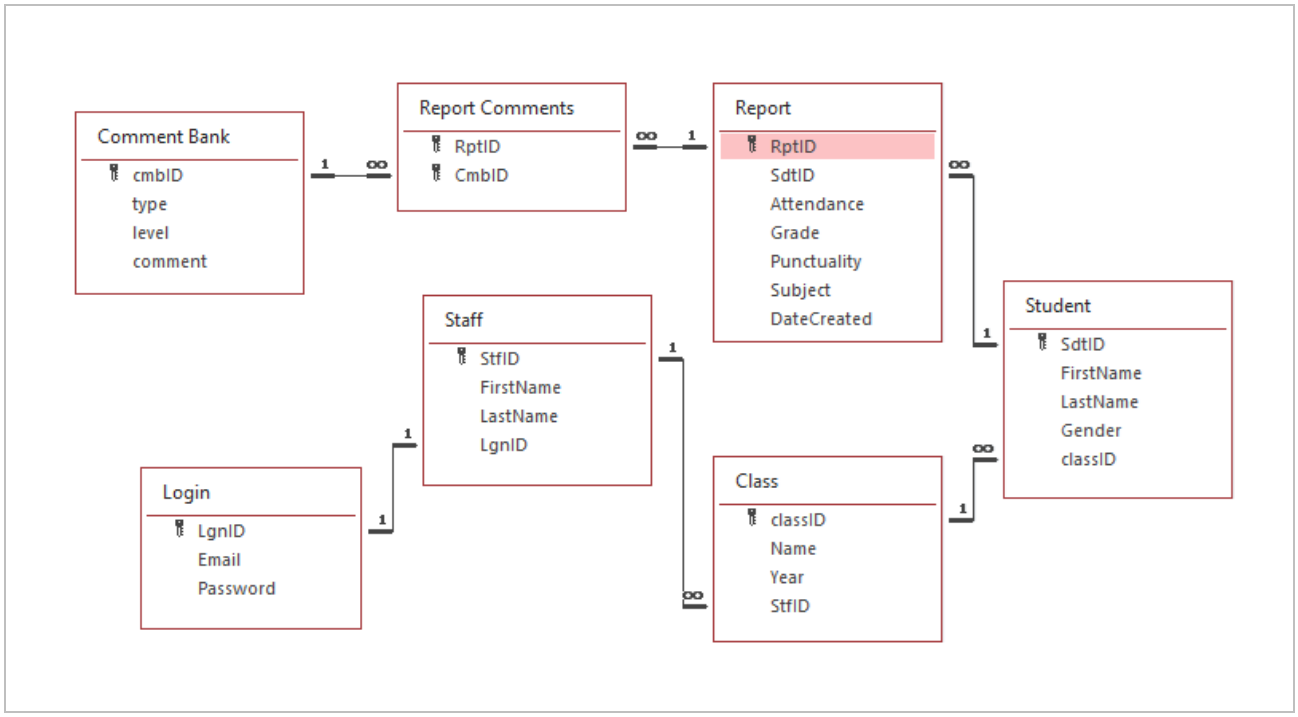
What has been done

The database has been created. Within the database, the tables have been created, with the correct fields (stated in 2.2 Structure of Solution). The Primary Keys, for each table, and data types, for each field, has been identified. This partially meets criteria's 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, since all requirements stated depend on a database to store data within and access.

Prototype 2

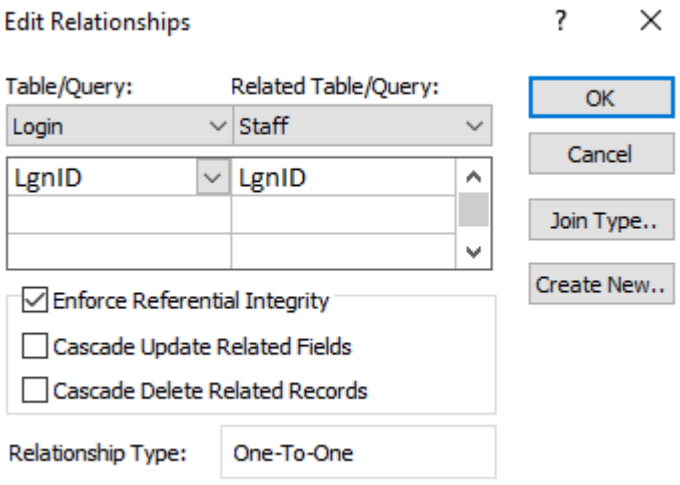
Relationships between each table will now be set up. In prototype1, fields were created to be foreign keys.

Screenshot of Relationship



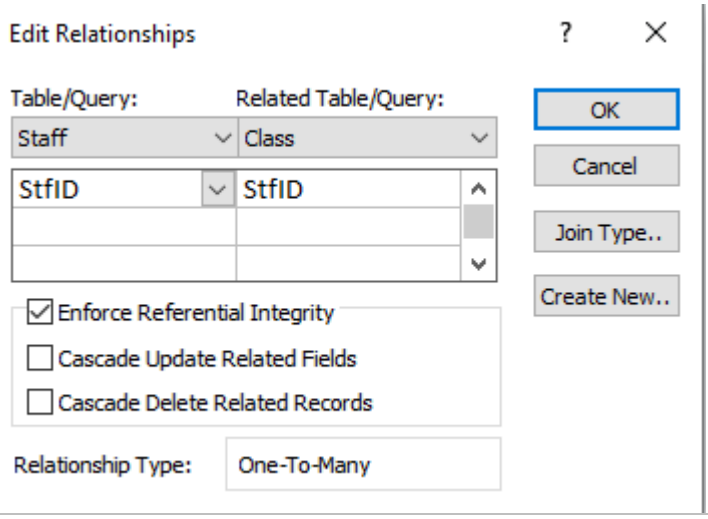
Explanation of Relationship between "Login" and "Staff"

There is a One-to-One relationship between the two tables. This is because a user (Staff) can only have one account.



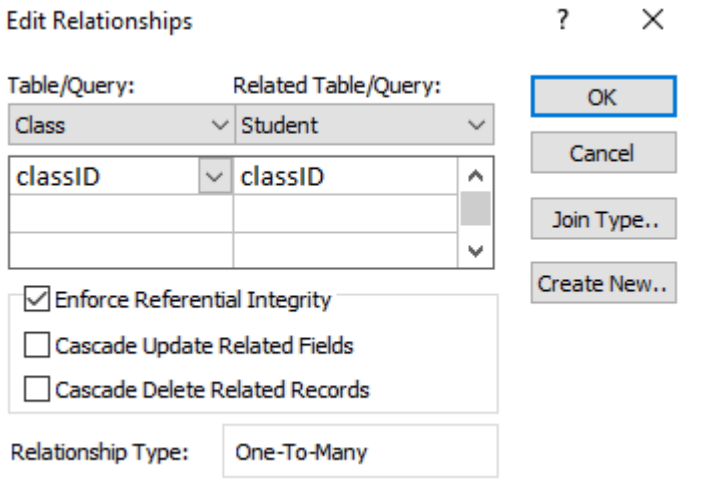
Explanation of Relationship between "Staff" and "Class"

There is a One-to-Many relationship between the two tables. This is because a user (Staff) can create as many classes as they wish to, as a Graveney teacher teaches more than one class. Each class belongs to a single teacher, as data is not shared between users.



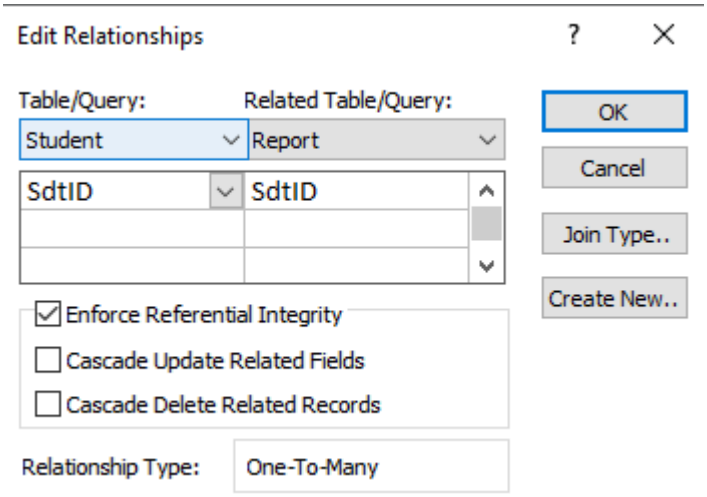
Explanation of Relationship between "Class" and "Student"

There is a One-to-Many relationship between the two tables. This is because a student of Graveney School can only be part of a single Form class, and a single class consists of multiple Students.



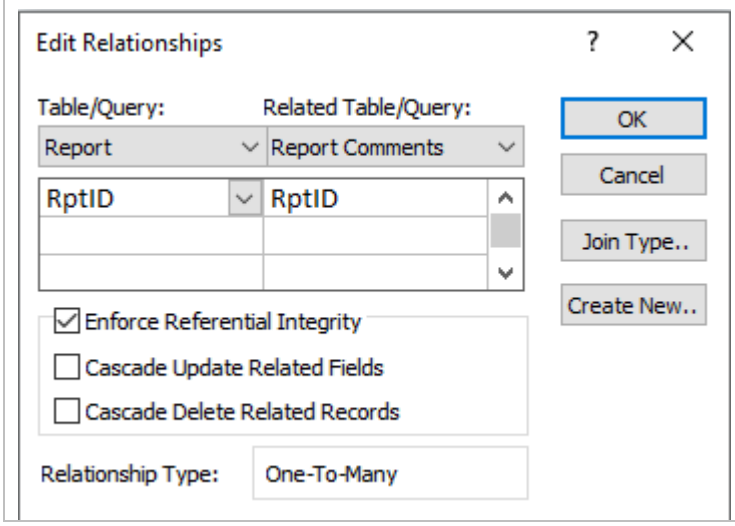
Explanation of Relationship between "Student" and "Report"

There is a One-to-Many relationship between the two tables. This is because Users (Staff) can write multiple reports on a single student. Each report will only be about an individual student.

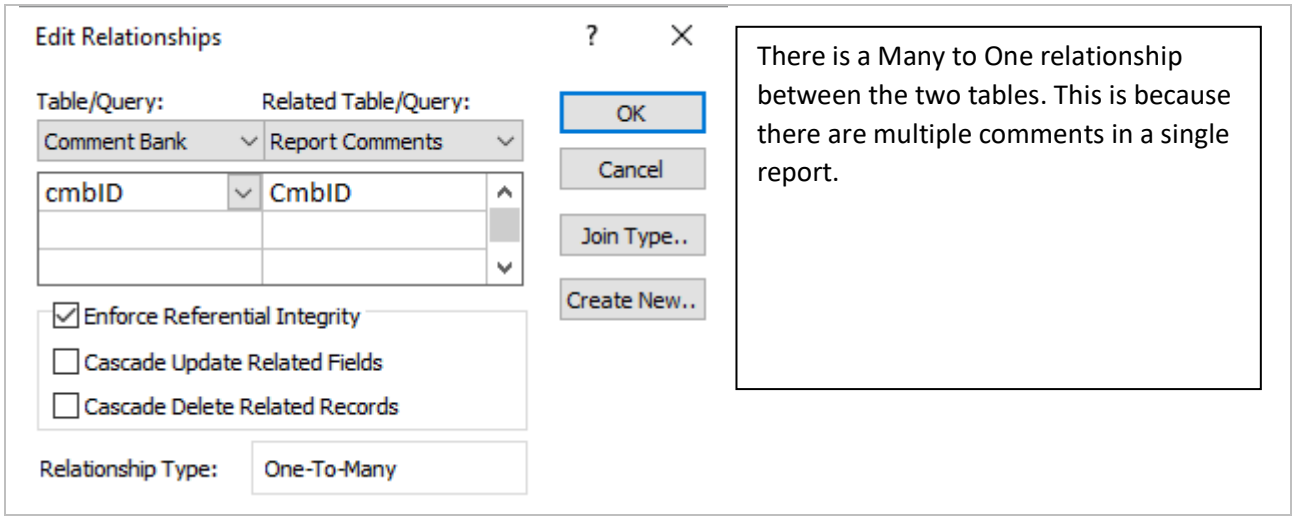


Explanation of Relationship between "Report" and "Report Comments"

There is a One-to-Many relationship between the two tables. This is because there are multiple comments in a single report.



Explanation of Relationship between "Report Comments" and "Comment Bank"



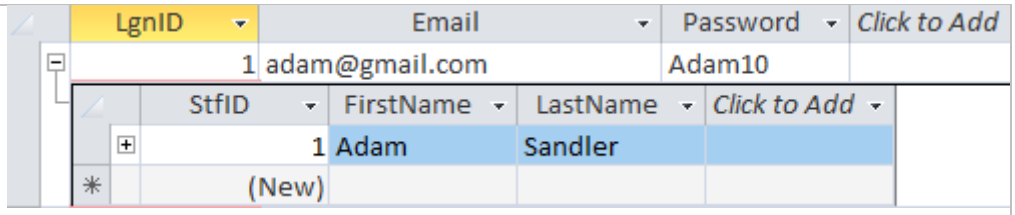
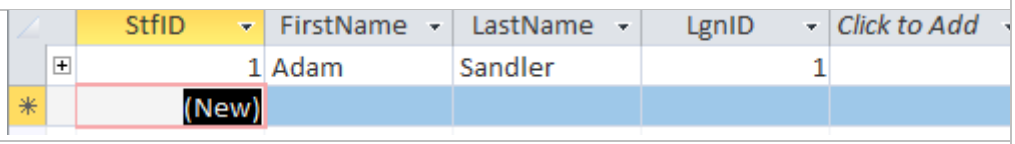
What has been done

Relationships between each table has been created with enforced referential integrity. This means that the records must reference an existing record. This is to keep data accurate and to prevent system from data being reference elsewhere.

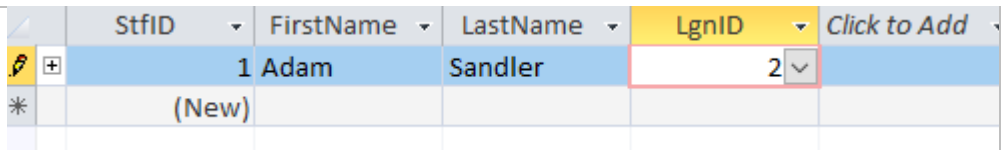
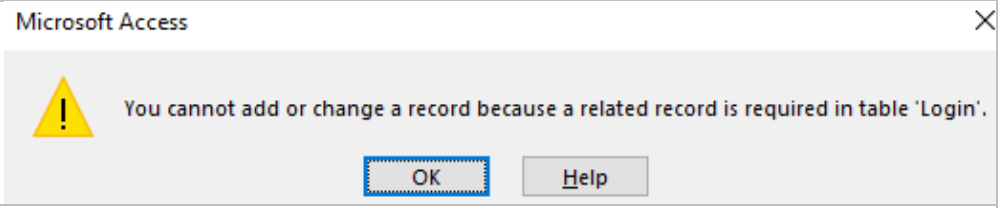
Testing of solution

Test carried out are 1.1, 1.2, 1.2, 1.3.

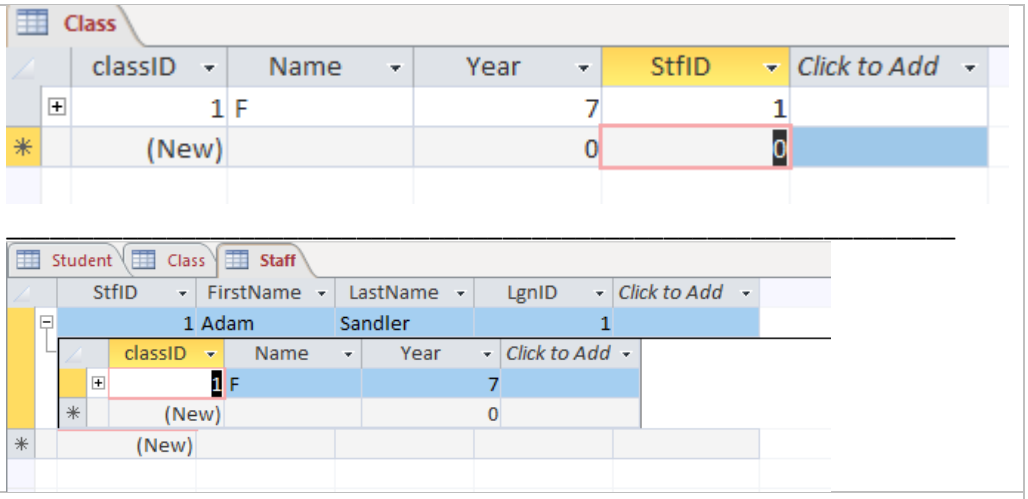
Test No 1.1.1 result: checking database can store account details…

Step 1) Inputting the test data values for Login table	
Step 2) Inputting the test data values for Signup table	
Summary: Test Successful – The database is able to store account data and has a relationship between Login and signup	

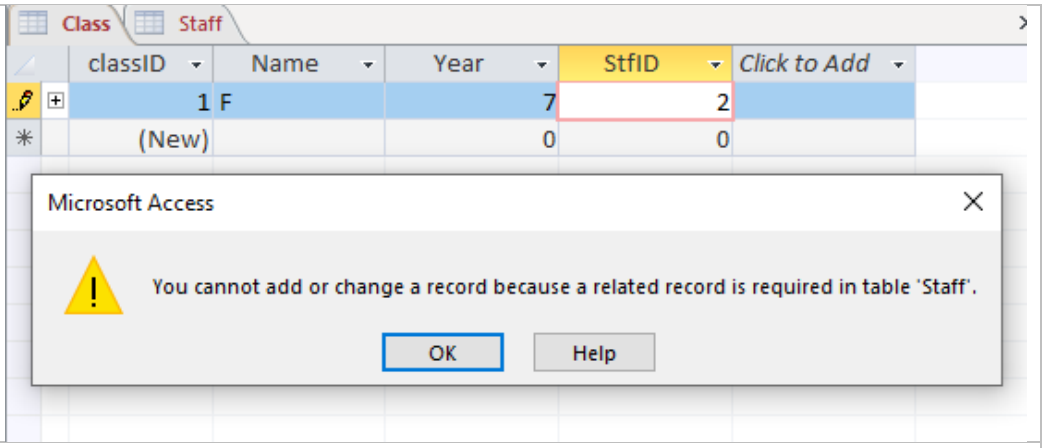
Test No 1.1.2 result: checking if referential integrity is enforced between Login and Staff…

Step 1) changing the value to 2	
Step 2) result…	
Summary: Test Successful – referential integrity is enforced.	

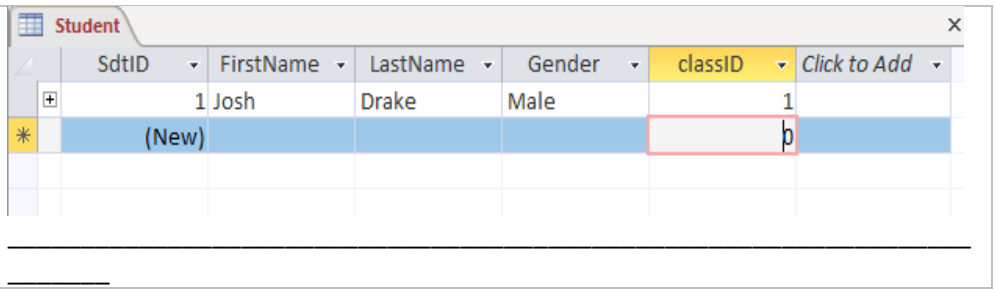
Test No 1.2.1 result: checking if database can store class details …

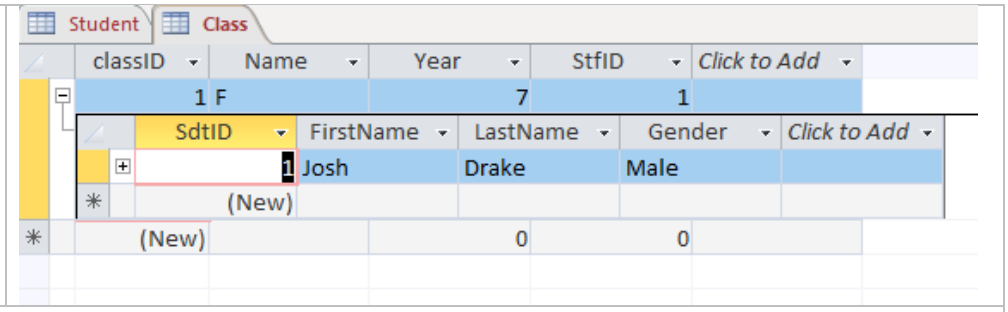
Inputting the test data values for Class table	
Summary: Test Successful – The database is able to store classes and reference a record from staff table	

Test No 1.2.2 result: checking if referential integrity is enforced between Class and Staff…

changing the value to 2	
Summary: Test Successful – referential integrity is enforced.	

Test No 1.3.1 result: checking if database can store student details …

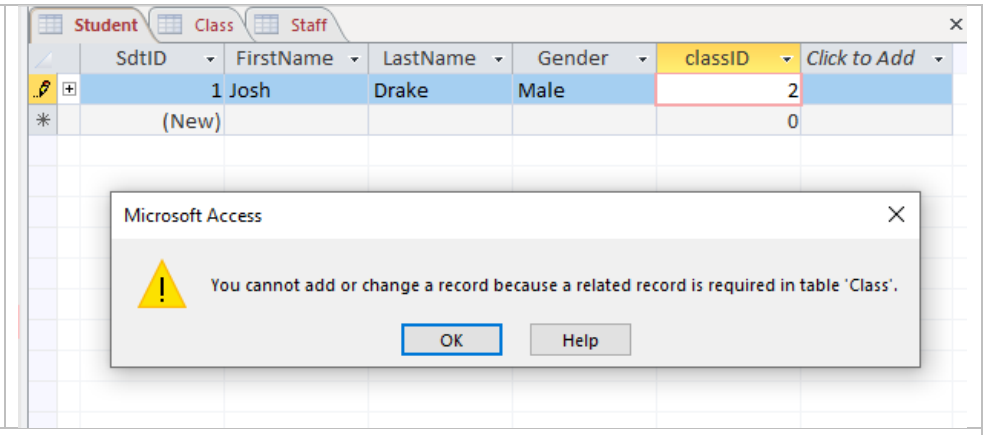
Inputting the test data values for Student table	



Summary: Test Successful – The database is able to store student's and reference a record from Class table

Test No 1.3.2 result: checking if referential integrity is enforced between Class and Student…

changing the value to 2



Summary: Test Successful – referential integrity is enforced.

Prototype 3

Sub-Comments are added into the Comment bank. These will be the phrases that will be used to generate the student's teacher report comment by the system. I have used Solution A to come up with necessary comments needed for the solution.

Screenshot and explanation of Comment Bank table

1	Opening	green	{name} has made very good progress this year, consistently producing work of a high standard and displaying a conscientious attitude throughout each lesson.
2	Opening	green	{name} has worked well this year, demonstrating a strong understanding of the work covered.
3	Opening	green	{name} has participated well in all activities, {pronoun1} always shows a conscientious and committed attitude towards {pronoun2} work.
4	Opening	amber	{name} 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 {pronoun2} own.
5	Opening	amber	{name} has worked well, but has struggled in some of the conent taught this year.
6	Opening	red	{name} has found the lesson content difficult to process and has struggled to remain on task, consequently {pronoun1} has made less than the expected level of progress.
7	Opening	red	{name} has found it difficult to connect with the subject content this year, finding it difficult to maintain {pronoun2} concentration throughout each lesson.
8	Behavior	green	{Pronoun1} has behaved in an appropriate manner, being a polite, helpful and well mannered student; a good role model to other students.
9	Behavior	amber	{Pronoun1} has been attentive for most of the lessons, but {pronoun1} has occasionally lost focus, due to being easily distracted by {pronoun2} surroundings.
10	Behavior	red	The attitude {pronoun1} has shown this year has been disappointing, often {pronoun1} became a distraction to others and disturbed the progress of the lesson. {name} has been given multiple warnings and punishments.
11	Progress	green	{name} has made pleasing progress this year, having consistently worked hard and has ensured all homework tasks have been handed in on time to the best of this ability.
12	Progress	green	{name} has made excellent progress this year, having consistently produced work of a very high standard.
13	Progress	green	{name} has worked well throughout this year, having enjoyed the work covered.
14	Progress	amber	The progress made by {name} would be improved if {pronoun1} made a greater effort to complete homework tasks to the best of {pronoun2} ability and ensure that it was handed in by the set deadline.
15	Progress	amber	{name} has progressed well this year, despite not always finding the work easy.
16	Progress	red	{Pronoun2} progress has been hampered at times by an inconsistent approach and lack of focus upon {pronoun2} own work.
17	Progress	red	Asking for help when needed would help {name} to make a faster rate of progress and would enable {pronoun3} to reach {pronoun2} predicted target grades.
18	Progress	red	{name} has made poor progress this year, having produced work of an inconsistent standard.
19	Strengths/ We	green	{Pronoun1} quickly gets to grips with new concepts and is able to make links with other areas of the curriculum, showing higher order thinking.
20	Strengths/ We	green	Despite not always finding the work easy {name} has always made a concerted effort to overcome any weaknesses and continually strives do to {pronoun2} best.
21	Strengths/ We	amber	{Pronoun1} has a tendency to hand in work after the deadline, better organisation is required.
22	Strengths/ We	amber	{Pronoun1} can be rather quiet during discussion work and rarely contributes {Pronoun2} own ideas, which is a shame as {Pronoun1} has a lot to offer the group.
23	Strengths/ We	red	{Pronoun2} results were lower than I would have expected, indicating a greater level of revision is required in the future.
24	Closing	green	Overall {name} has made a good level of progression this year and has met {pronoun2} target grades. I wish you well for the future {name}.
25	Closing	amber	{name} has shown {pronoun3}self to be an accomplished worker, who needs to show a more consistent approach, through being fully focussed in class.
26	Closing	red	The progress shown by {name} this year has been below expected and {Pronoun1} now needs to make a more concerted effort.

These will be used to generate the comment. The system will replace {name} with the students name, and {pronoun…} with pronouns depending on the students gender found in table student.

{pronoun1} is he/ she

{pronoun2} is his/ her

{pronoun3} is him/ her

When P is capital, the first character of pronoun is capital, e.g. {Pronoun1} is He/ She.

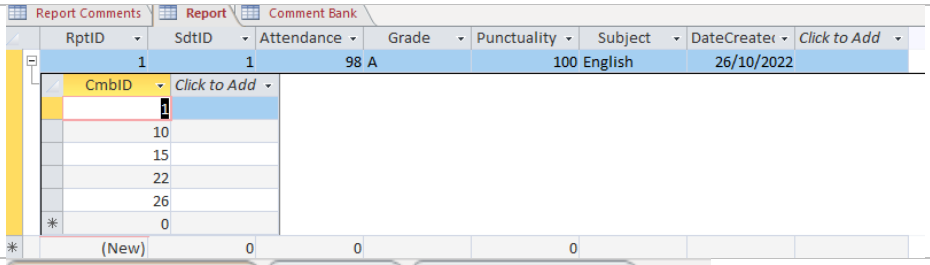
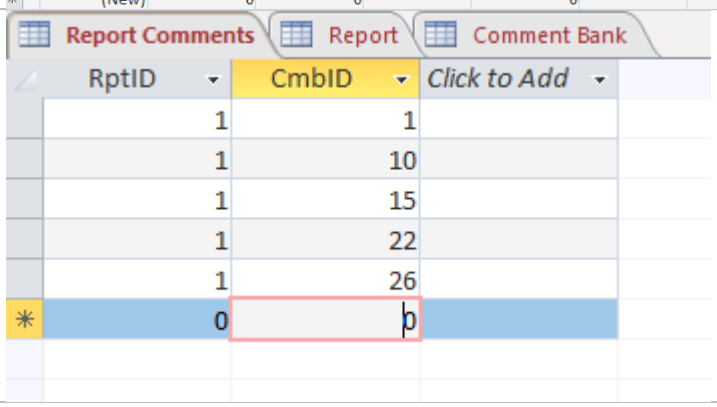
What has been done

The phrases have been written into the database. Ideas for the phrases were taken from existing solution A. This will allow reports to create the comment by linking multiple phrases to a single report. The criteria 9 has been partially met. These questions are used to generate the report comment using multiple choice questions.

Testing of solution

Test carried out are 1.4.

Test No 1.4.1 result: checking database can store reports…

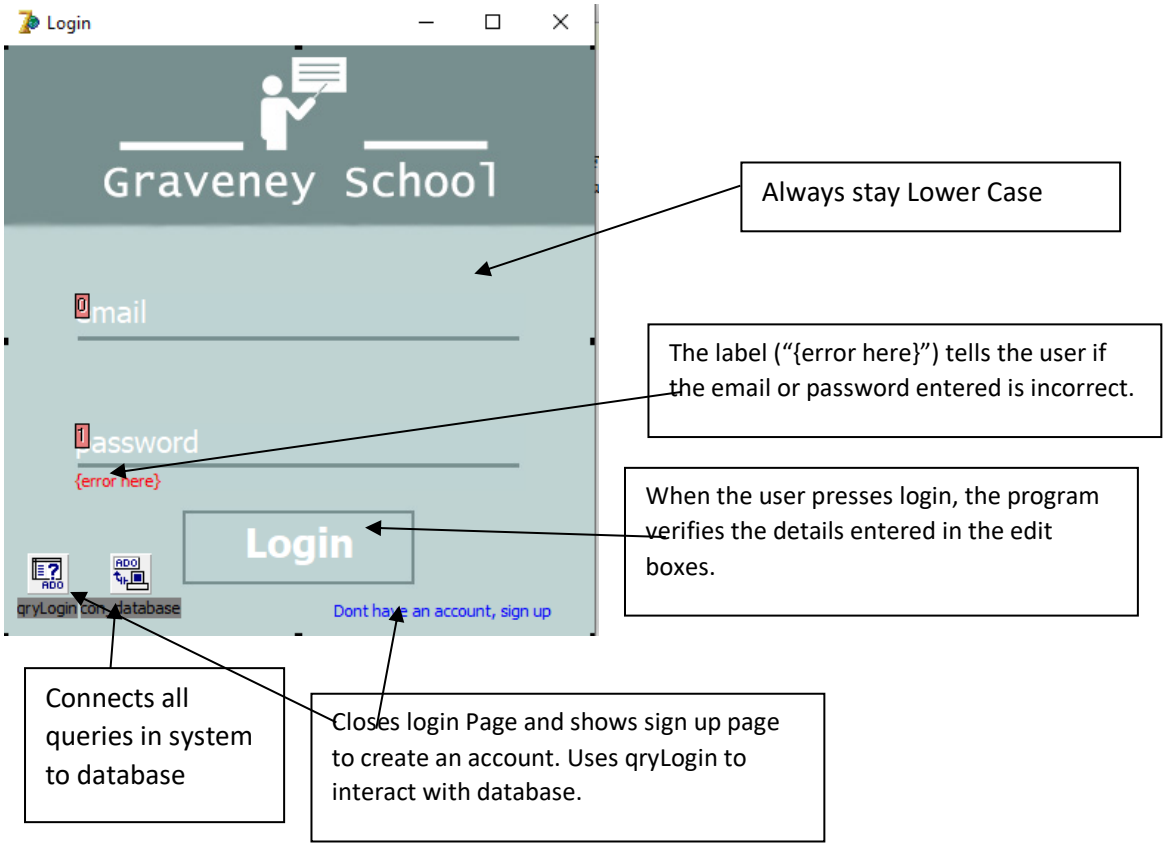
Step 1) Inputting the test data values for Report table	
Step 2) Inputting the test data values for report comments table	
Summary: Test Successful – The database is able to store account data and has a relationship between Report and report comments, and report comments also has a relationship with comment bank	

Stage 2: Implementing the "Login page"

The first form is being created. This is the login page where users will be able to login using their details. The requirements needed to be met by the login page are 4, 8.1.

Prototype 1

Screenshot of "Login Form"



Implemented code for login Form (unit_login)

```
unit unit_login;

interface

uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, jpeg, ExtCtrls, StdCtrls, DB, ADODB, unit_signup, shellAPI,unit_menu;

type
  TfrmLogin = class(TForm)
  published
    logo_header: TImage;
    edtEmail: TEdit;
    edtpassword: TEdit;
    con_database: TADOConnection;
    qryLogin: TADOQuery;
    lblerror: TLabel;
    lblsignUP: TLabel;
    btnLogin: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure btnLoginClick(Sender: TObject);
    procedure btnLoginMouseLeave(Sender: TObject);
```

```
    procedure btnLoginMouseEnter(Sender: TObject);
    procedure lblcloseClick(Sender: TObject);
    procedure lblsignUPClick(Sender: TObject);
    procedure edtpasswordClick(Sender: TObject);
    procedure edtEmailClick(Sender: TObject);
  end;
//type record to store user details
type TUserData = record
    FirstName, LastName, Email, Password: string;
    StaffID: Integer;
end;
var
  frmLogin: TfrmLogin;
  info: TUserData; //variable storing key details from database to running program
to be used

implementation

{$R *.dfm}

//connects program to database
procedure TfrmLogin.FormCreate(Sender: TObject);
begin
  lblerror.caption: ='';
  con_database.Connected := True;
end;


procedure databaseLogin;
begin
//Selects field from login where email is equivelent to the email inputted by user
and selects the row related to the record, from signup
  frmLogin.qryLogin.SQL.Text := 'SELECT staff.StfID, Login.Email, Login.Password,
staff.firstname, staff.lastname FROM Login inner join staff on (login.LgnID =
staff.LgnID) where email = '+quotedstr(frmLogin.edtemail.text)+';';
  frmLogin.qryLogin.Open;
//if 0 records then no account exists under that email address
  if (frmLogin.qryLogin.RecordCount=0)then
    frmLogin.lblerror.caption:='Email not found'
  else //1 record
  begin
    if (frmLogin.qryLogin.fieldbyname('password').AsString =
frmLogin.edtpassword.Text)then
//save user details needed later in the system
      begin
        info.FirstName:=frmLogin.qryLogin.fieldbyname('firstname').AsString;
        info.LastName:=frmLogin.qryLogin.fieldbyname('lastname').AsString;
        info.Email:=frmLogin.qryLogin.fieldbyname('email').AsString;
        info.password:=frmLogin.qryLogin.fieldbyname('password').AsString;
        info.StaffID:=frmLogin.qryLogin.fieldbyname('StfID').AsInteger;
//{label in main menu will display users full name}
//opening main menu in same position, and closing login form
        frmMainMenu.Top := frmLogin.Top;
        frmMainMenu.left := frmLogin.left;
        frmmainmenu.Show;
        frmlogin.Hide;
      end
      else
      begin
        //Informs user password is incorrect (requirement 4.2)
        frmLogin.lblerror.Caption:='incorrect password';
```

```
        //Changes Edit Password back to password when the value was invalid; to
ensure user still is aware of what needs to be inputted there
        frmLogin.edtpassword.Text := 'password';
        frmLogin.edtpassword.PasswordChar := #0;
      end;
  end;
  frmLogin.qryLogin.close;
end;

//Making sure no values are null and user is infromed if there is (requirement
4.2)
procedure TfrmLogin.btnLoginClick(Sender: TObject);
begin
  if not ((edtEmail.Text = '') or (edtpassword.Text = '')) then
    databaseLogin
  else
    ShowMessage('complete the form...');
end;

procedure TfrmLogin.lblsignUPClick(Sender: TObject);
begin
  //opening sign up in same position, and closing login form
  frmSignup.Top := frmLogin.Top;
  frmSignup.left := frmLogin.left;
  frmSignup.Show;
  frmLogin.Hide;
end;

//making password character change to be hidden
procedure TfrmLogin.edtpasswordClick(Sender: TObject);
begin
  if (edtpassword.passwordchar=#0)then
  begin
    edtpassword.PasswordChar := '*';
  end;
end;

// changing color of btnLogin to indicate that the mouse is on top
procedure TfrmLogin.btnLoginMouseLeave(Sender: TObject);
begin
  btnLogin.Font.Color:= clWhite;
end;

procedure TfrmLogin.btnLoginMouseEnter(Sender: TObject);
var max:Integer;
begin
  btnLogin.Font.Color:= RGB(119,143,143);
end;

end.
```

Changed in Prototype 2

Validation carried out

Validation of the user's email	The email entered is searched in the database table "Login". There can only be 0 or 1 record containing the email (email field is unique). If 0 records, then email is either invalid or the account doesn't exist. If query has a record, then the account does exist.
Validation of the user's password	The password is validated if email is found within the database. The password entered compared to the password of the record found. If the password is not the same, the user has entered it incorrectly.

Validate if form values are not empty	If one of the values of the form is empty, it will inform the user.

What has been done

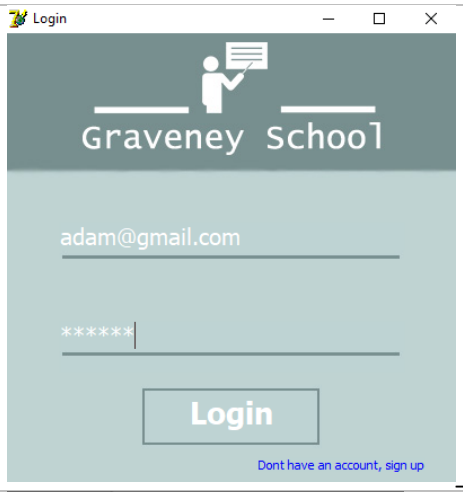
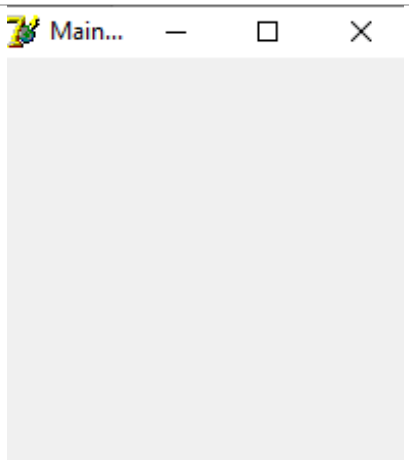
Login form has been created, designed and implemented. Its purpose is to verify a user. Validation algorithms have been used to make sure user is authentic, preventing people from breaking into system. It also allows users to open the sign-up form. It is the first form that shows up when system is run. It has two edit box objects which lets the user input their email and password. The program validates it when the user presses the login button. To open the signup from, the user needs to click on the label at the bottom of the form. When the user logs in, the main menu is shown

Criteria 1 has been completed.

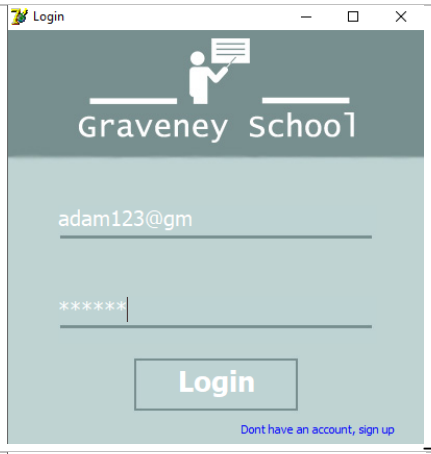
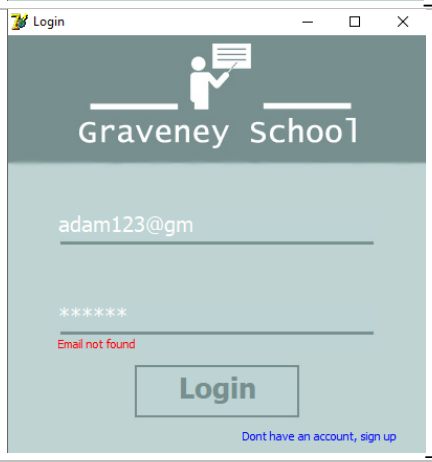
Testing of solution

Test carried out are 2.1, 2.2, 2.3, 2.4.

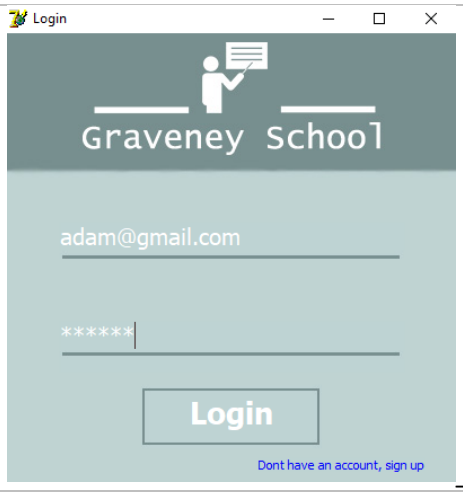
Test No 2.1.1 result: checking if users can login with their accounts…

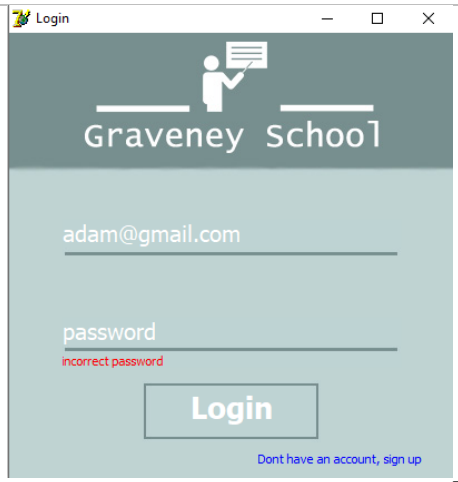
Step 1) Inputting the test data values	
Step 2) Pressing Login Button	 Form isn't complete
Summary: Test Successful – The system is able to login and displayed the menu form, when the correct values of a registered account are entered.	

Test No 2.1.2 result: checking that program informs user when email is incorrect, and doesn't allow them to login…

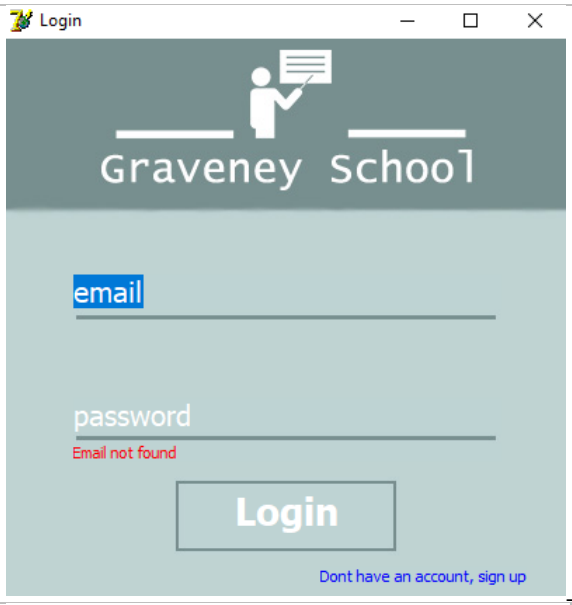
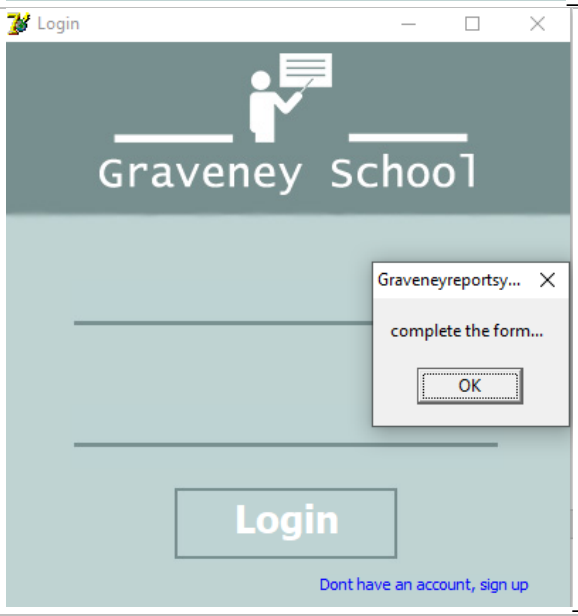
Step 1) Inputting the test data values	
Step 2) Pressing Login Button	
Summary: Test Successful – The system is able to change the caption of the label to 'email not found', and doesn't show menu; doesn't login.	

Test No 2.1.3 result: checking that program informs user when password is incorrect, and doesn't allow them to login…

Step 1) Inputting the test data values	

Step 2) Pressing Login Button	
Summary: Test Successful – The system is able to change the caption of the label to 'incorrect password', change the value of the password edit box, back to password, and doesn't show menu; doesn't login.	

Test No 2.2.1 result: checking system informs user when values are null…

When pressing login without typing anything	
Clearing edit box to value null, then pressing login button	

Summary: Test Failed – Logically, the system should output ‘complete the form’ when the values of email and password edit boxes are ‘email’ and ‘password’ as the user didn’t input that, and they are just there indicating the user what to type in and shouldn’t be considered as a value. I will have to write a set of validation in the next prototype making sure when the email edit box is “email” and the password edit box is “password” the system portrays that as an empty field and displays the complete the form pop up.

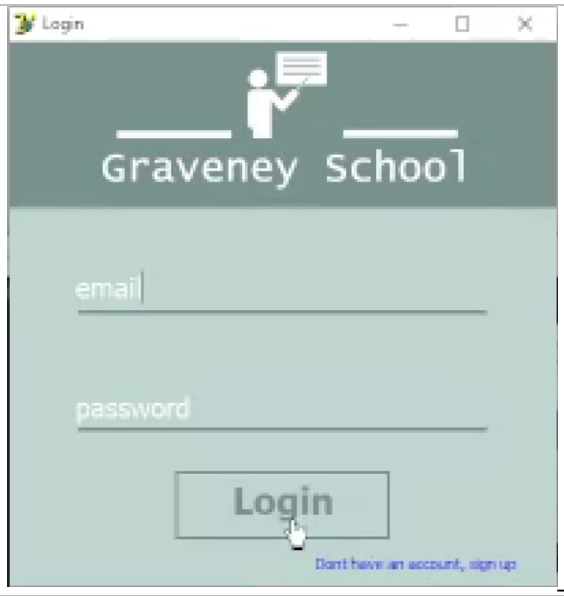
Test No 2.2.2 result: checking if system informs user when email value is null…

Automatic Test Failure, same issue as Test No 1.2.1.

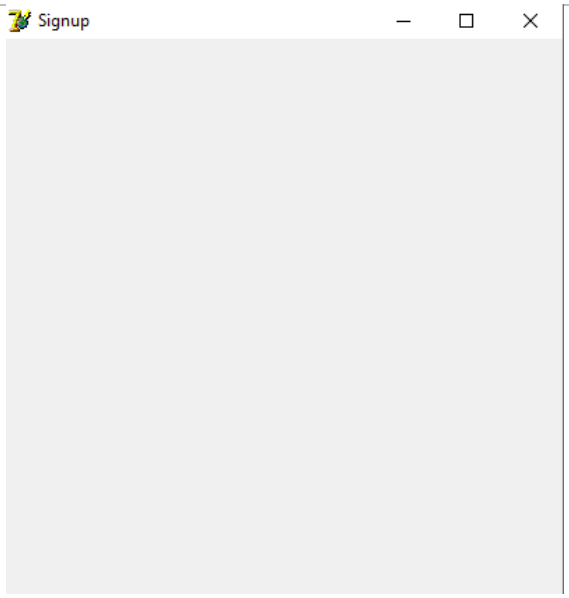
Test No 2.2.3 result: checking if system informs user when password value is null…

Automatic Test Failure, same issue as Test No 1.2.1.

Test No 2.3 result: checking if system changes button colour when cursor hovering on top…

Hovering Cursor on button	
Summary: Test Successful – The system is able to change the colour only when the mouse is on top, otherwise it is white.	

Test No 2.4 result: checking if system can open sign up from login form…

Clicking on ‘don’t have and account, sign up’ on login form	 Form isn’t complete

Summary: Test Successful – The system is able to open signup form from login form by clicking the label 'don't have and account, sign up'. It is also able to close login form

Prototype 2

Prototype 2 is to fix the errors that occurred within the test in prototype1. It also makes some changes by improving the user interaction with the systems Graphical Interface.

Changes in Implemented code

Code	Explanation

`procedure TfrmLogin.btnLoginClick(Sender: TObject);`
`begin`
`  if not((edtEmail.Text = '')or (edtpassword.Text = '')or(edtEmail.Text = 'email')or (edtpassword.Text = 'password')) then`
`    databaseLogin`
`  else`
`    ShowMessage('complete the form...');`
`end;`
	Attempting to Fix error that occurred in test 1.2.1, 1.2.2, 1.2.3, by changing the condition of the selection in procedure 'TfrmLogin.btnLoginClick'.

`procedure TfrmLogin.edtpasswordClick(Sender: TObject);`
`begin`
`  if ((edtpassword.Text = 'password')and(edtpassword.passwordchar=#0))then`
`  begin`
`    edtpassword.Text := '';`
`    edtpassword.PasswordChar := '*';`
`  end`
`  else`
`    edtpassword.PasswordChar := '*';`
`  if((edtemail.Text =''))then`
`    edtemail.Text := 'email';`
`end;`
 	
`procedure TfrmLogin.edtEmailClick(Sender: TObject);`
`begin`
`  if (edtemail.Text = 'email')then`
`    edtemail.Text := '';`
`  if((edtpassword.Text =''))then`
`  begin`
`    edtpassword.Text := 'password';`
`    edtpassword.PasswordChar := #0;`
`  end;`
`end;`
	Disappears 'email' or 'password'.
	
	When user clicks on the password edit box…
	…System checks if password edit box is 'password' and if the characters are showing. If the condition is true, the password edit box becomes null, so user can start typing without deleting the default text 'password', and the system changes characters to '*' when user types. This is to keep the user's password hidden, to prevent anyone from knowing what it is.
	…System checks if email edit box value is null. If condition is true, the system changes it to 'email', so that the user is aware of what needs to be entered within it.
	
	When the user clicks on the email edit box…
	…System checks if email edit box is 'email'. If the condition is true, the email edit box becomes null, so user can start typing without deleting the default text ('email'). …System checks if password edit box is null. If condition is true, it changes it to 'password', with characters showing, so that the user is aware of what needs to be entered within it.

Changes in Validation carried out

Validate if form values are not empty	If one of the values of the form is empty, or if the value is 'email' or 'password' respectively, it will inform the user.

What has been done

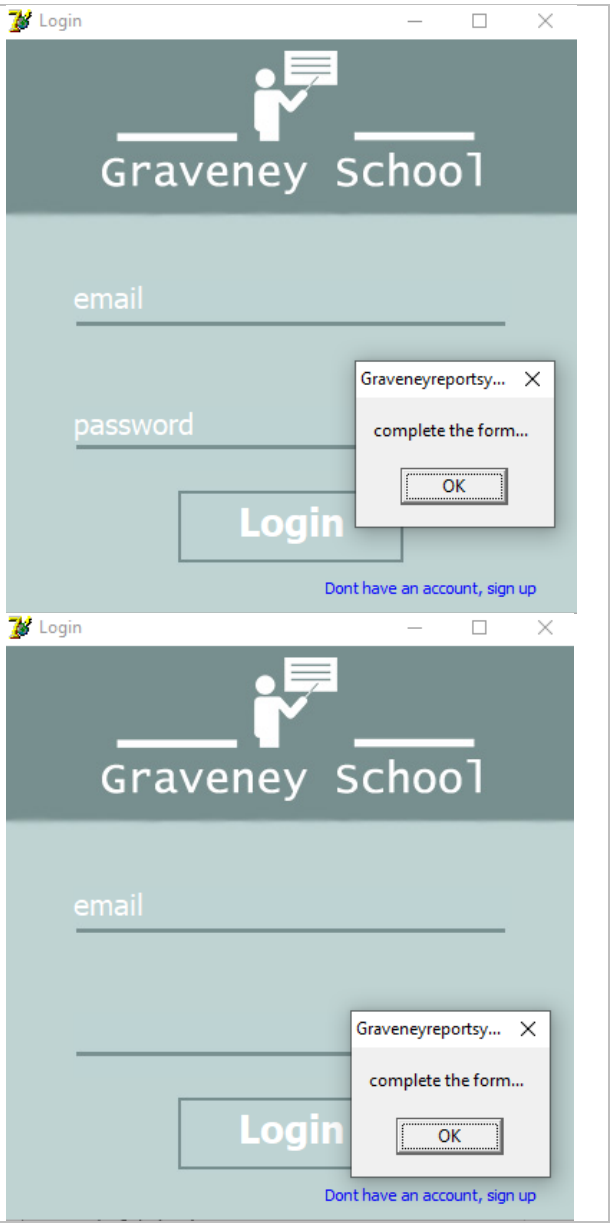
Changes have been made to prototype 1 to fix errors. Prototype 2 fixes the issue of the system not being able to inform user when values are null correctly. Prototype 2 also makes the system more user friendly, by disappearing hints for the user automatically.

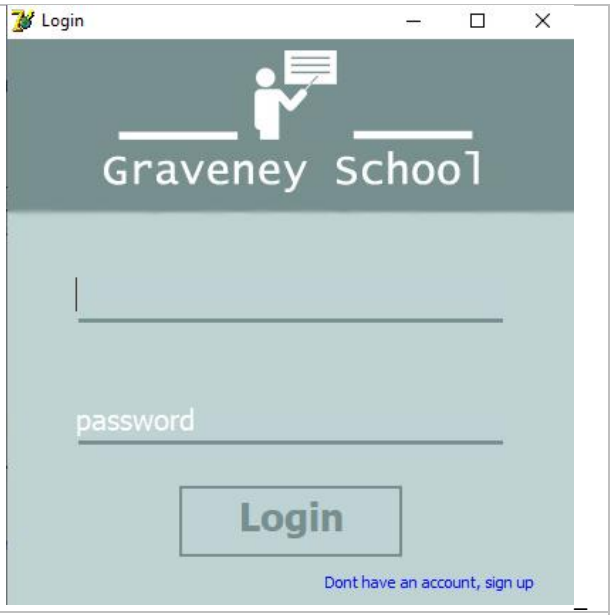
Requirement 4.2 has been met.

Testing of solution

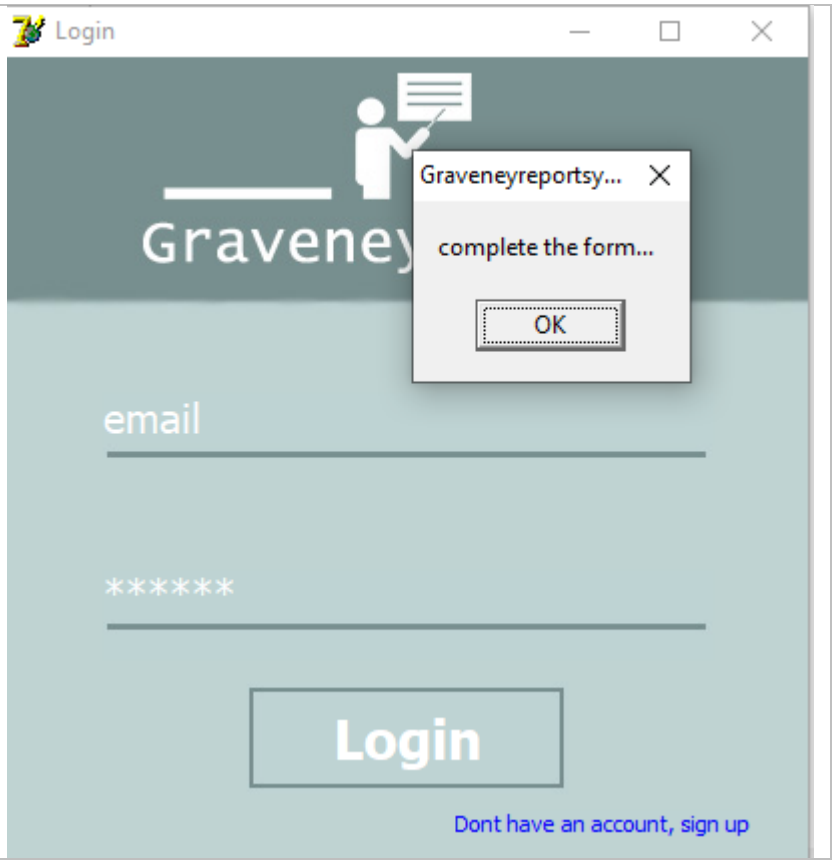
Test carried out are 2.2.

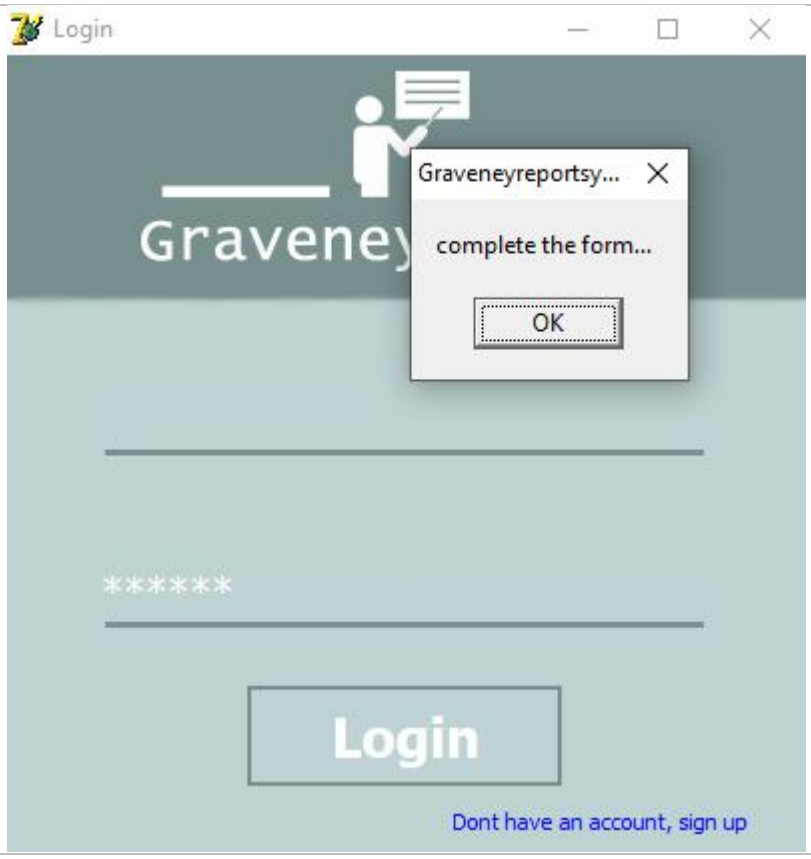
Test No 2.2.1 result: checking system informs user when values are null…

When pressing login without typing anything	

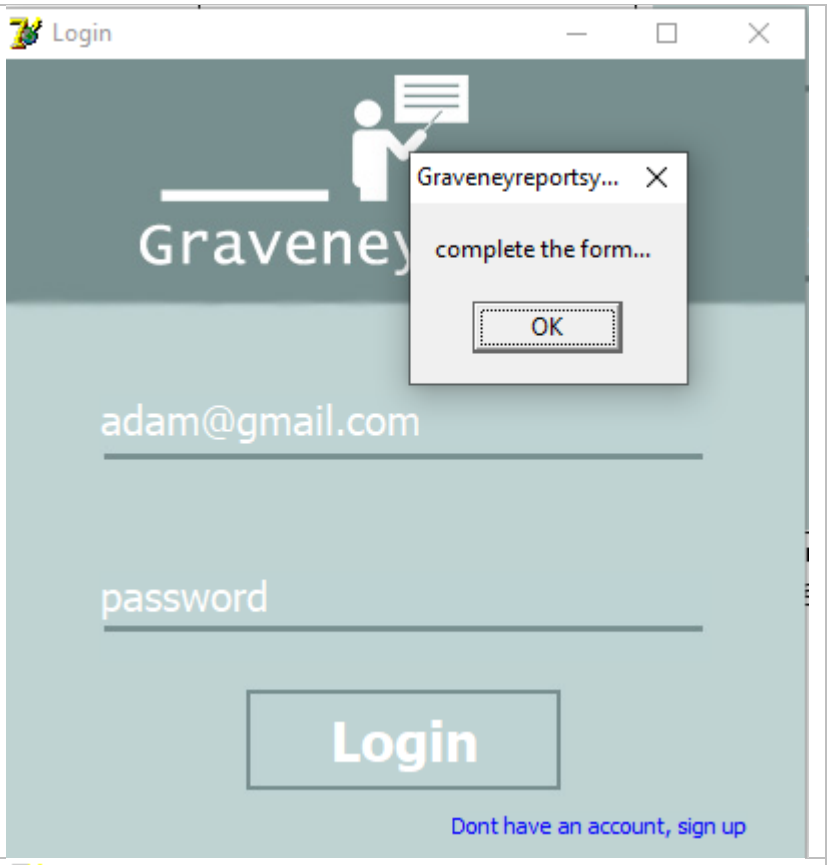
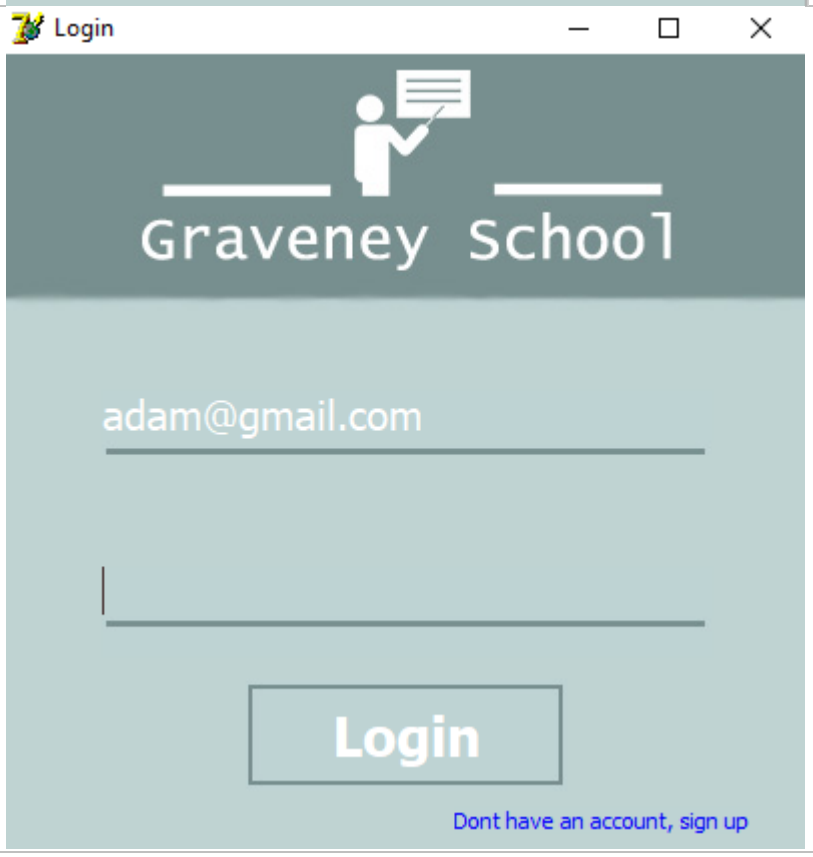
	
Summary: Test Successful – 'Email' and 'password' are just hints and considered as null. The system is able to output the message in all possible ways where the values of the edit box are either null or 'email'/ 'password'.	

Test No 2.2.2 result: checking if system informs user when email value is null…

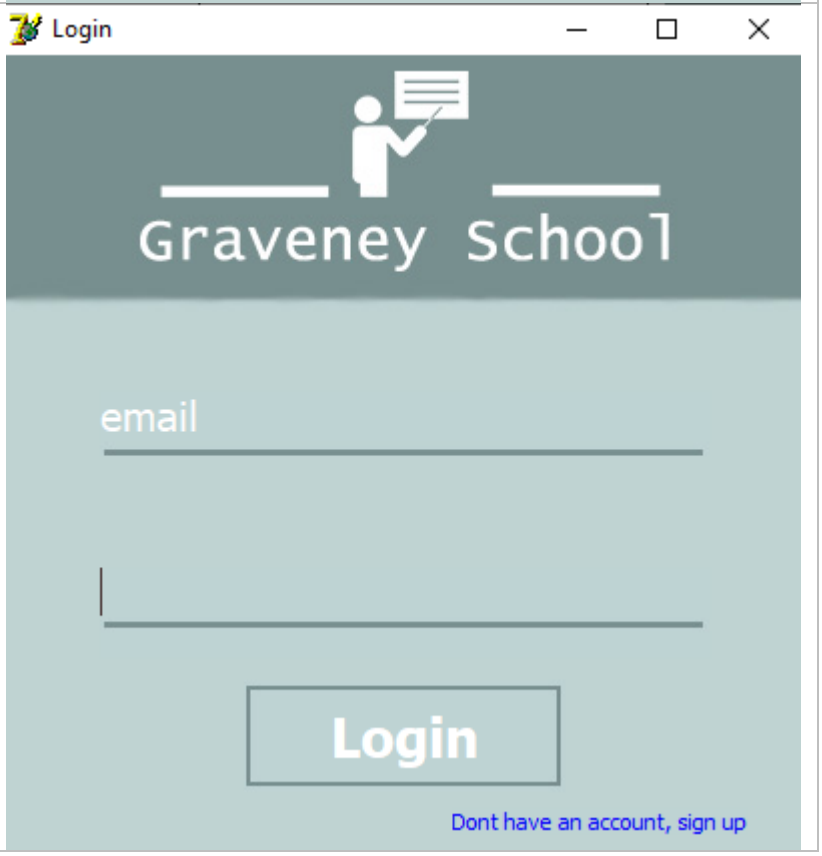
Entering 'Adam10' into Password edit box, then pressing Login	

Entering ‘Adam10’ into Password edit box and deleting ‘email’ from email edit box, then pressing Login	
Summary: Test Successful – ‘Email’ is just a hint and considered as null. The system is able to output the message when the value of email edit box is null and password edit box is not null.	

Test No 2.2.3 result: checking if system informs user when password value is null…

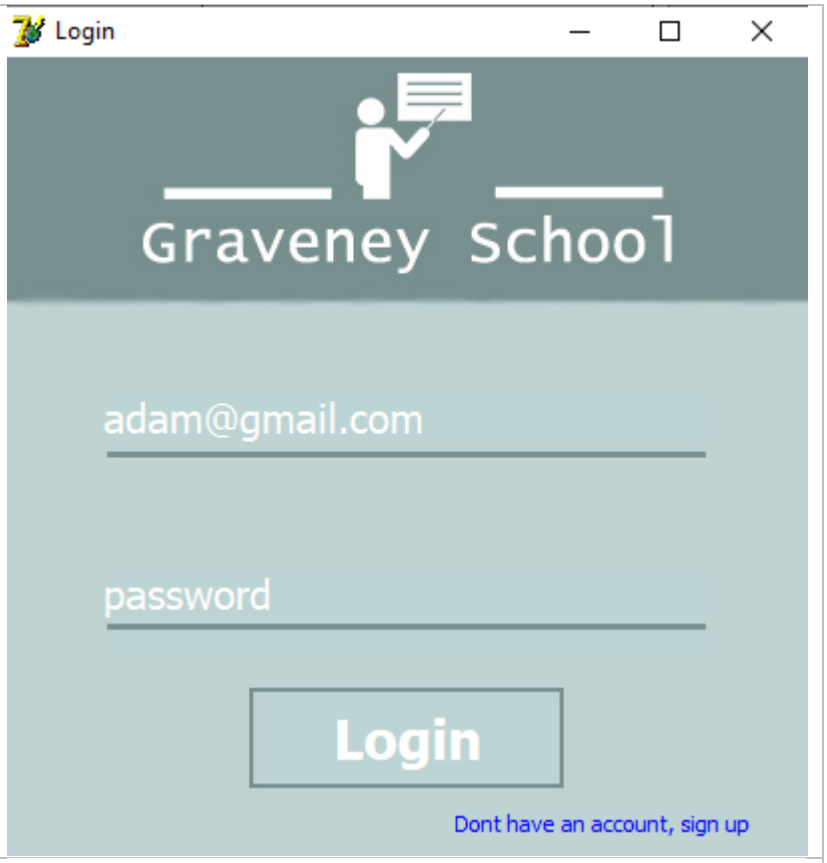
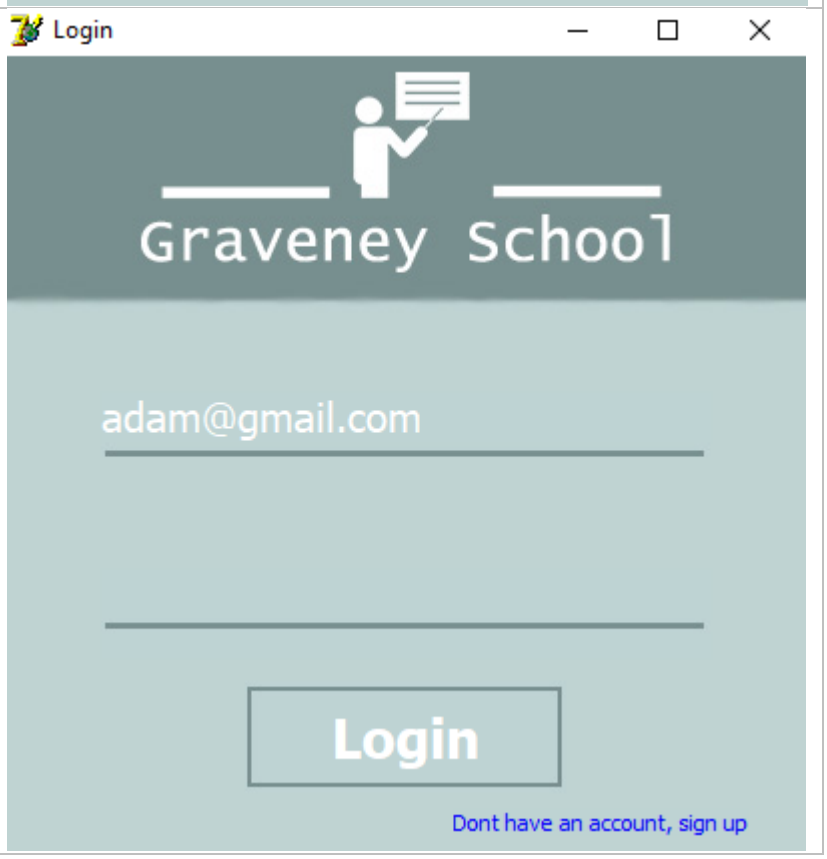
Entering ‘adam@gmail.com’ into email edit box, then pressing Login	
Entering ‘adam@gmail.com’ into email edit box and deleting ‘password’ from password edit box, then pressing Login	
Summary: Test Successful – ‘Password’ is just a hint and considered as null. The system is able to output the message when the value of password edit box is null and email edit box is not null.	

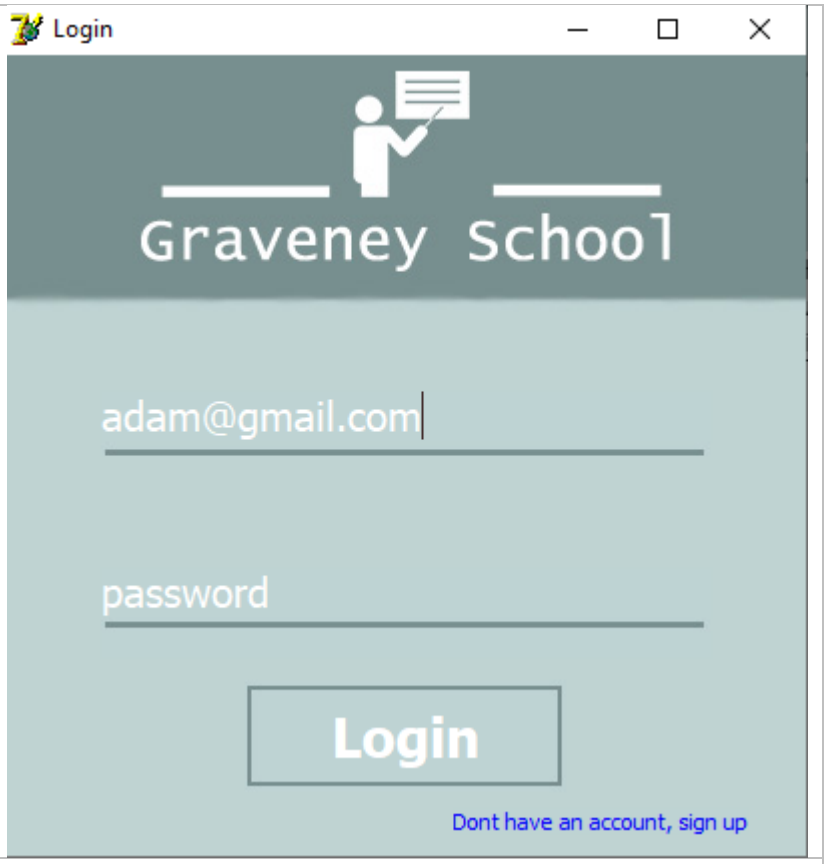
Test No 2.5.1 result: checking if the system can hide and show hints automatically…

Step1) clicking on email edit box	
Step2) clicking on password edit box	

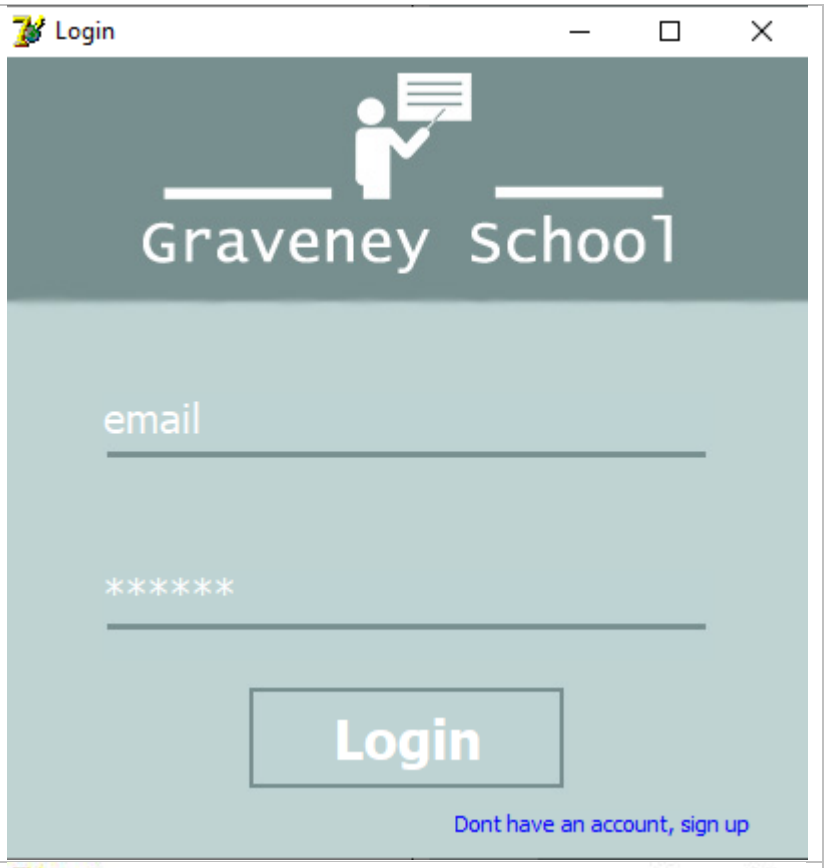
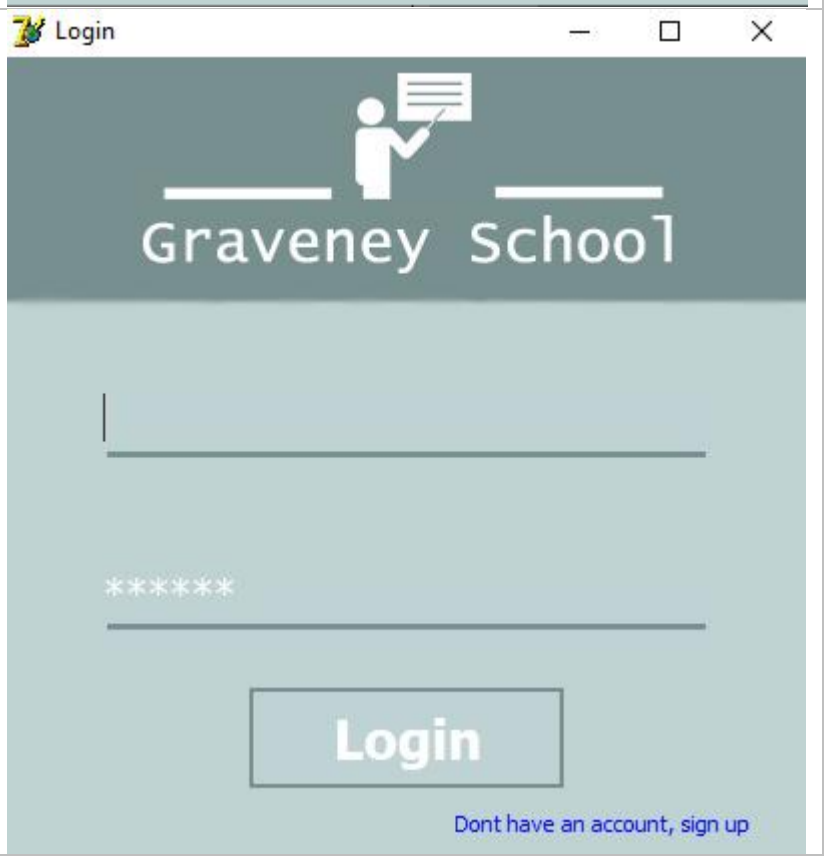
Step3) clicking on email edit box	
Summary: Test Successful – The system is able to show and hide the hints for the edit boxes. This means the user does not have to delete the hints when they want to enter a value. It also makes them appear back, making sure the user is fully aware of what needs to go inside the edit boxes.	

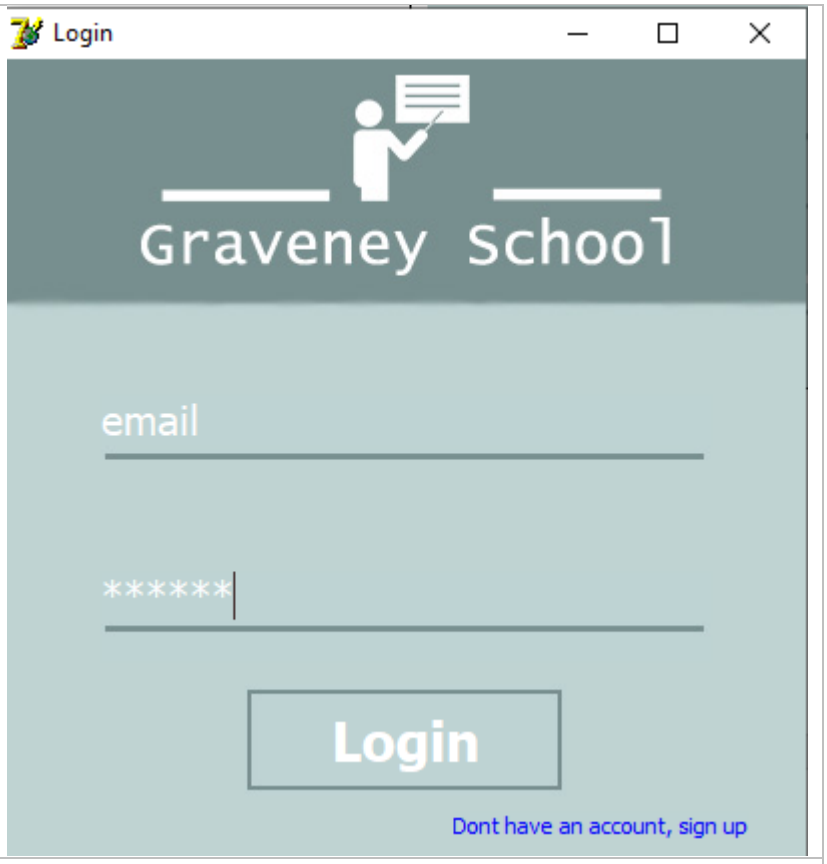
Test No 2.5.2 result: checking if the system can prevent users email entered from being altered…

Step1) entering value into email edit box	
Step2) clicking on password edit box	

Step3) clicking on email edit box	
Summary: Test Successful – The system is able to show and hide the hint for email edit box when the value is only null or 'email' respectively. The system will not alter or erase the values entered by the user in email edit box.	

Test No 2.5.3 result: checking if the system can prevent user's password entered from being altered…

Step1) entering value into password edit box	
Step2) clicking on email edit box	

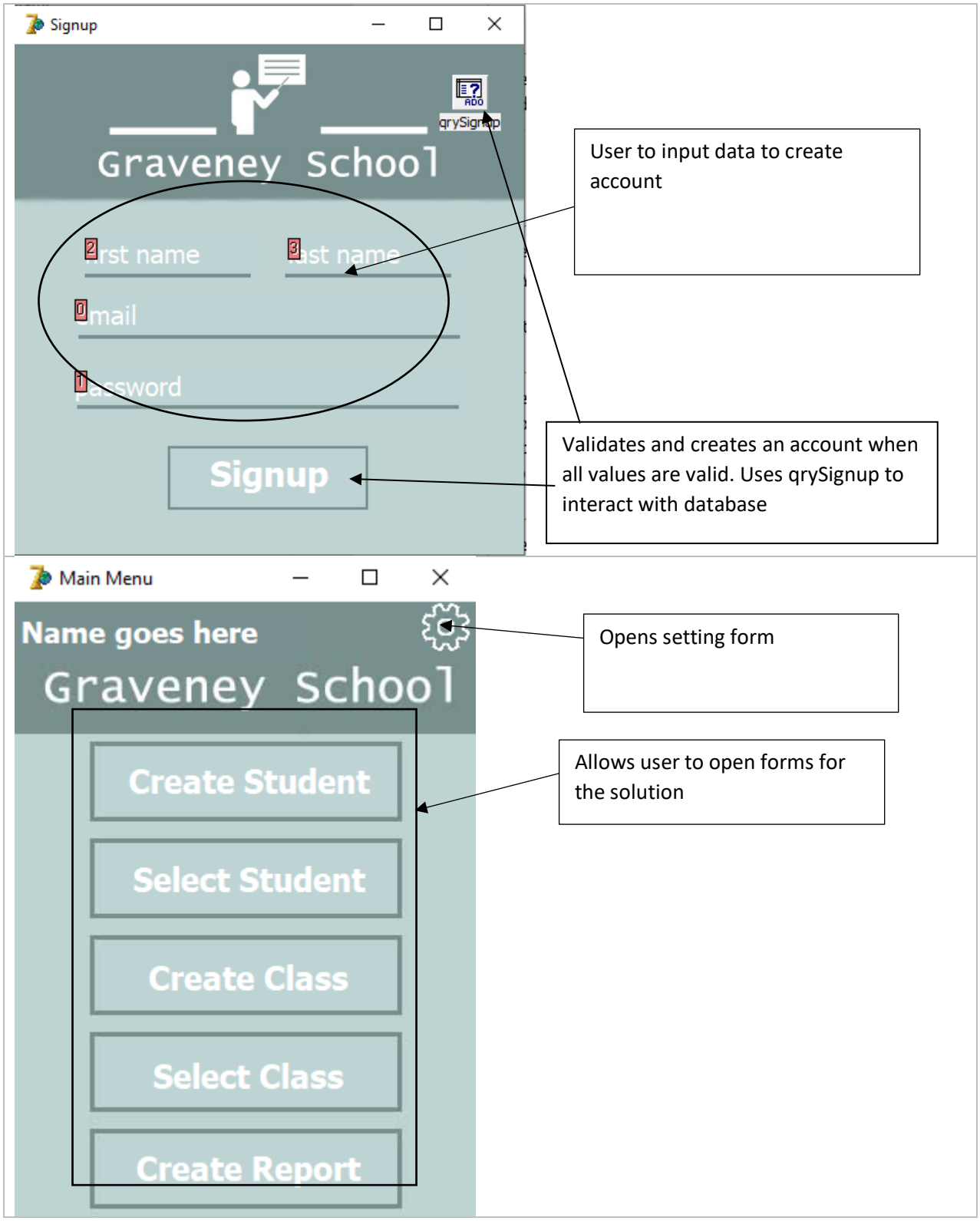
Step3) clicking on password edit box	
Summary: Test Successful – The system is able to show and hide the hint for password edit box when the value is only null or 'password' respectively. The system will not alter or erase the values entered by the user in password edit box.	

Stage 3: Implementing the "Signup page" and "Main Menu"

The Signup form and Main Menu are being implemented. The signup form allows users to create an account and login. Main Menu will link all forms needed to solve the problem.

Prototype 1

Screenshot of "Signup Form" and "Main Menu Form"



Implemented Code for Signup Form (unit_signup)

```
unit unit_signup;

interface

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

type
  TfrmSignup = class(TForm)
  published
    logo_header: TImage;
    edtEmail: TEdit;
    edtpassword: TEdit;
    edtfName: TEdit;
    btnSignup: TLabel;
    edtlName: TEdit;
    qrySignup: TADOQuery;
    procedure btnSignupClick(Sender: TObject);
    procedure btnSignupMouseEnter(Sender: TObject);
    procedure btnSignupMouseLeave(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure edtEmailClick(Sender: TObject);
    procedure edtfNameClick(Sender: TObject);
    procedure edtlNameClick(Sender: TObject);
    procedure edtpasswordClick(Sender: TObject);
  end;

var
  frmSignup: TfrmSignup;

implementation

uses unit_login;

{$R *.dfm}

//Checks if there is an integer in a string
function isintinstr(search:string):Boolean;
var i,j:Integer;
  found:Boolean;
  const numbers = ['0'..'9'];
begin
  for i:=1 to Length(search) do
  begin
    if (Search[i] in numbers)then
      found := True;
  end;
  isintinstr := found;
end;

function FirstLetterUppercase(temp:string):string;
begin
  temp := lowercase(temp);
  temp[1] := uppercase(temp[1]);
  FirstLetterUppercase := temp;
end;

//Checks if string only consists of alphabetical characters
function stringisAlpha(name:string):Boolean;
```

```
const charslist = ['A'..'Z', 'a'..'z'];
var i:Integer;
    problem:Boolean;
begin
  problem:=false;//initial value is false
  for i:=1 to Length(name) do
  begin
    if not((name[i] in charsList))then
      problem:=true; // Contains a character, which is not an alphebet.
  end;
  stringisAlpha:=not(problem);
end;


function verifyName_and_Password(fname,lname,password:string):Boolean;
var outcome:Boolean;
begin
  outcome:=false;
  //Name must only consist of alphabetical characters
  if not((stringisAlpha(fname))and (stringisAlpha(lname)))then
    ShowMessage('invalid name')
  //password length must be greater than/ equal to 8
  else if (Length(password) >=8) then
  begin
    if not(isintinstr(password))then
    begin
      ShowMessage('password must be greater than 8'+#13#10+'characters, and
include a digit');
    end
    else
      outcome:=true;
  end
  else
  begin
      ShowMessage('password must be greater than 8'+#13#10+'characters, and
include a digit');
  end;
  verifyName_and_Password := outcome;
end;

//verifys email address
function IsValidEmail(email: string): boolean;
const
  charslist = ['_', '-', '.', '0'..'9', 'a'..'z'];
var
  FoundAt, lastpoint,valid : boolean;
  i, emailLen : integer;
  c : char;
begin
  emailLen := Length(email);
  FoundAt := false;
  lastpoint := false;
  valid := true;
  for i:=1 to emailLen do
  begin
    c := email[i];
    if c = '@' then
    begin
      if(FoundAt)then // Can only be one @ symbol in email
        valid := false
      else
```

```
        FoundAt := true;
    end
    else if(c = '.')and(foundAt)then  //at least 1 '.' after arobasc
      lastpoint := true
    else if not(c in charslist)then  //valid chars
      valid := false;
  end;
  if not(lastpoint) or (email[emailLen] = '.')then  //if no '.'in email, or
finishes at '.' Then…
    valid := false;
  IsValidEmail:=valid;
end;

//writes account to database tables login and staff
function writeAccount(fname,lname,email,password:string):boolean;
var LgnID:Integer;
begin
  frmSignup.qrySignup.SQL.Text := 'Select LgnID from login where
email='+quotedStr(email)+';';
  frmSignup.qrySignup.open;
  //checks if email exits…
  if (frmSignup.qrySignup.RecordCount=0)then
  begin
    //if it doesn't exist, it inserts it to the tables
    frmSignup.qrySignup.close;
    frmSignup.qrySignup.SQL.Text := 'insert into login(email,password) values
('+quotedStr(email)+','+ quotedStr(password)+');';
    frmSignup.qrySignup.ExecSQL;
    //finds the LgnID so that the staff table can reference it
    frmSignup.qrySignup.SQL.Text := 'Select LgnID from login where
email='+quotedStr(email)+';';
    frmSignup.qrySignup.open;
    LgnID := frmSignup.qrySignup.fieldbyname('LgnID').AsInteger;
    frmSignup.qrySignup.close;
    frmSignup.qrySignup.SQL.Text := 'insert into
staff(LgnID,Firstname,Lastname) values
('+inttostr(LgnID)+','+quotedStr(fname)+','+ quotedStr(lname)+');';
    frmSignup.qrySignup.ExecSQL;
    ShowMessage('account created');
    writeAccount:=True;
  end
  else
  begin
    frmSignup.qrySignup.close;
    ShowMessage('account with email address already exists');
    writeAccount:=False;
  end;
end;

procedure TfrmSignup.btnSignupClick(Sender: TObject);
var i:integer;
  Temp:string;
begin
  if (verifyName_and_Password(edtfName.Text, edtlName.Text,
edtpassword.Text))then
  begin
    if (IsValidEmail(edtEmail.Text))then
    begin
      if (writeaccount(edtfName.Text, edtlName.Text, edtEmail.Text,
edtpassword.Text))then
      begin
```

```
        //keep same pos
        frmLogin.Top := frmSignup.Top;
        frmLogin.left := frmSignup.left;
        //closes signup form when account is created
        frmLogin.show;
        frmSignup.close;
      end;
    end
    else
      ShowMessage('invalid Email address');
  end;
end;

procedure TfrmSignup.btnSignupMouseEnter(Sender: TObject);
begin
  btnsignup.Font.Color:= RGB(119,143,143);
end;

procedure TfrmSignup.btnSignupMouseLeave(Sender: TObject);
begin
  btnSignup.Font.Color:= clWhite;
end;

//Opens login when signup is closed
procedure TfrmSignup.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  //keep same pos
  frmLogin.Top := frmSignup.Top;
  frmLogin.left := frmSignup.left;
  frmLogin.show;
end;

//Make sure Edit Boxes always show what needs to be entered in it.
procedure TfrmSignup.edtEmailClick(Sender: TObject);
begin
  if (edtemail.Text = 'email')then
    edtemail.Text := '';
  if((edtpassword.Text =''))then
    edtpassword.Text := 'password';
  if((edtfname.Text =''))then
    edtfname.Text := 'first name';
  if((edtlname.Text =''))then
    edtlname.Text := 'last name';

end;

procedure TfrmSignup.edtfNameClick(Sender: TObject);
begin
  if (edtemail.Text = '')then
    edtemail.Text := 'email';
  if((edtpassword.Text =''))then
    edtpassword.Text := 'password';
  if((edtfname.Text ='first name'))then
    edtfname.Text := '';
  if((edtlname.Text =''))then
    edtlname.Text := 'last name';
end;

procedure TfrmSignup.edtlNameClick(Sender: TObject);
begin
  if (edtemail.Text = '')then
```

```
    edtemail.Text := 'email';
  if((edtpassword.Text =''))then
    edtpassword.Text := 'password';
  if((edtfname.Text =''))then
    edtfname.Text := 'first name';
  if((edtlname.Text ='last name'))then
    edtlname.Text := '';
end;

procedure TfrmSignup.edtpasswordClick(Sender: TObject);
begin
  if (edtemail.Text = '')then
    edtemail.Text := 'email';
  if((edtpassword.Text ='password'))then
    edtpassword.Text := '';
  if((edtfname.Text =''))then
    edtfname.Text := 'first name';
  if((edtlname.Text =''))then
    edtlname.Text := 'last name';
end;

end.
```

Validation carried out for "signup" (unit_signup)

Validation of email	An email has to consist of only one '@' symbol; separates username with mail server name. An email must have at least one full stop. The full stop is valid to be used within the username. It also is used to separate the mail server with domain. The domain is the end of the email address, therefore if there is a full stop at the end of an email address the user enters, it is not a valid email address. An email address only consists of'_', '-', '.', any digit(s), any Alphabetical character and a '@'.
Validation of password	A password must be greater than or equal to 8 characters. It must have at least one digit.
Validation of name	Name must only be Alphabetical characters.
Validation of Account	If an account with same email exists, the account is not created.

Implemented Code for "Main Menu" (unit_menu)

```
unit unit_menu;

interface

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

type
  TfrmMainMenu = class(TForm)
  published
    lblProfileName: TLabel;
    ImgMainMenu: TImage;
    btnCreateStudent: TLabel;
    btnSelectStudent: TLabel;
    btnCreateClass: TLabel;
    btnSelectClass: TLabel;
```

```
    btnCreateReport: TLabel;
    imgSettings: TImage;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure btnCreateClassClick(Sender: TObject);
    procedure btnSelectClassClick(Sender: TObject);
    procedure btnCreateStudentClick(Sender: TObject);
    procedure btnCreateStudentMouseEnter(Sender: TObject);
    procedure btnCreateStudentMouseLeave(Sender: TObject);
    procedure btnSelectStudentClick(Sender: TObject);
    procedure btnSelectStudentMouseEnter(Sender: TObject);
    procedure btnSelectStudentMouseLeave(Sender: TObject);
    procedure btnCreateClassMouseEnter(Sender: TObject);
    procedure btnCreateClassMouseLeave(Sender: TObject);
    procedure btnSelectClassMouseEnter(Sender: TObject);
    procedure btnSelectClassMouseLeave(Sender: TObject);
    procedure btnCreateReportClick(Sender: TObject);
    procedure btnCreateReportMouseEnter(Sender: TObject);
    procedure btnCreateReportMouseLeave(Sender: TObject);
    procedure imgSettingsClick(Sender: TObject);
  end;

var
  frmMainMenu: TfrmMainMenu;

implementation

uses unit_login;

{$R *.dfm}

procedure TfrmMainMenu.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  frmlogin.close; //once main menu is closed, the whole program closes
end;

//Create student button - label component used as a button
procedure TfrmMainMenu.btnCreateStudentClick(Sender: TObject);
begin
  //open form
end;

//changes the color when cursor is on top; lets user know it's a button
procedure TfrmMainMenu.btnCreateStudentMouseEnter(Sender: TObject);
begin
  btnCreateStudent.Font.Color:= RGB(119,143,143);
end;

procedure TfrmMainMenu.btnCreateStudentMouseLeave(Sender: TObject);
begin
  btnCreateStudent.Font.Color:= clWhite;
end;

//Select student button - label component used as a button
procedure TfrmMainMenu.btnSelectStudentClick(Sender: TObject);
begin
  //open form
end;

procedure TfrmMainMenu.btnSelectStudentMouseEnter(Sender: TObject);
begin
```

```
  btnSelectStudent.Font.Color:= RGB(119,143,143);
end;

procedure TfrmMainMenu.btnSelectStudentMouseLeave(Sender: TObject);
begin
  btnSelectStudent.Font.color:=clWhite;
end;

//Create Class button - label component used as a button
procedure TfrmMainMenu.btnCreateClassClick(Sender: TObject);
begin
  //open form
end;

//changes the color when cursor is on top; lets user know it's a button
procedure TfrmMainMenu.btnCreateClassMouseEnter(Sender: TObject);
begin
  btnCreateClass.Font.Color:= RGB(119,143,143);
end;

procedure TfrmMainMenu.btnCreateClassMouseLeave(Sender: TObject);
begin
  btnCreateClass.Font.Color:= clWhite;
end;

//Select Class button - label component used as a button
procedure TfrmMainMenu.btnSelectClassClick(Sender: TObject);
begin
  //open form
end;

//changes the color when cursor is on top; lets user know it's a button
procedure TfrmMainMenu.btnSelectClassMouseEnter(Sender: TObject);
begin
  btnSelectClass.Font.Color:= RGB(119,143,143);
end;

procedure TfrmMainMenu.btnSelectClassMouseLeave(Sender: TObject);
begin
  btnSelectClass.Font.Color:= clWhite;
end;

//Create Report button - label component used as a button
procedure TfrmMainMenu.btnCreateReportClick(Sender: TObject);
begin
  //open form
end;

//changes the color when cursor is on top; lets user know it's a button
procedure TfrmMainMenu.btnCreateReportMouseEnter(Sender: TObject);
begin
  btnCreateReport.Font.Color:= RGB(119,143,143);
end;

procedure TfrmMainMenu.btnCreateReportMouseLeave(Sender: TObject);
begin
  btnCreateReport.Font.Color:= clWhite;
end;

//update user details button - image component used as a button
procedure TfrmMainMenu.imgSettingsClick(Sender: TObject);
```

```
begin
  //open form
end;

end.
```

What has been done

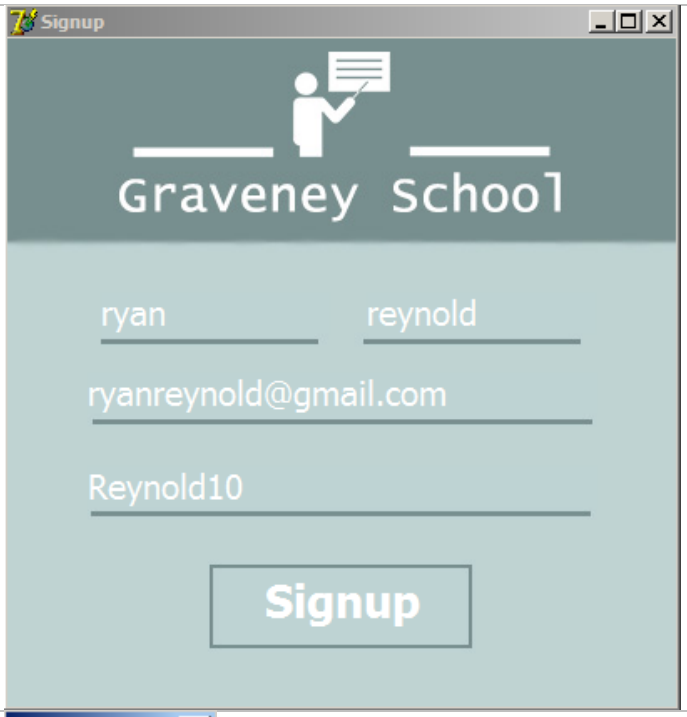
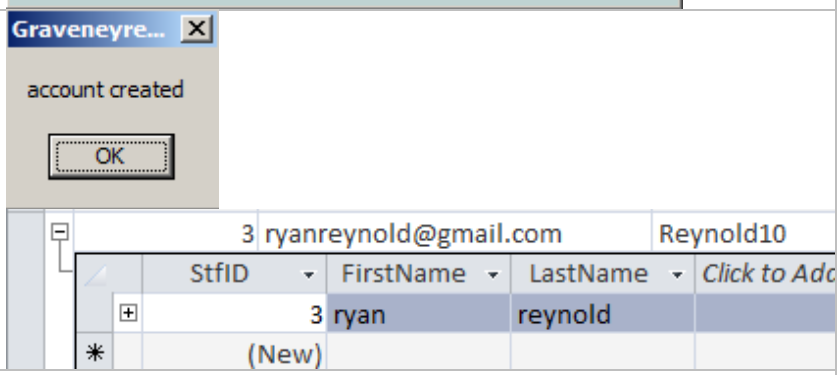
The signup and main menu form have been implemented. Criteria 2 has been met. It allows the user to create an account to be able to use the system, hence requirement 5 has been met. The Main Menu Form connects all the forms needed to solve the computational problem. It allows users to open the forms, hence requirement 8.3 is partially met. The prototype was tested in runtime of the code. Screenshots of results of the tests are shown below.

Summary: The user is able to create and sign in using an account. The main menu form has also been created, which will allow users to open different forms for the solution.

Testing of solution

Test carried out are 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7.

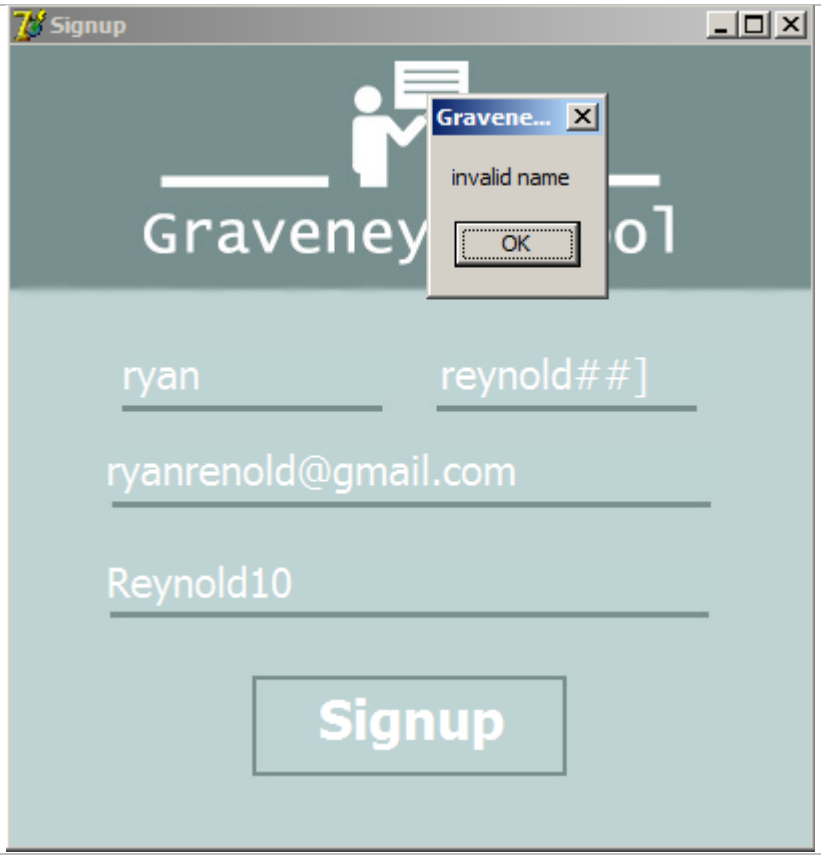
Test No 3.1 result: checking if users can create an account…

Step 1) Entering the values	
Step 2) pressing signup button	
Summary: Test Successful – The system is able to create an account and have it stored within the database. This allows the user to login. The system also prompts the user a message when the account has been created. Requirement 5.3 and 5.1 has been met.	

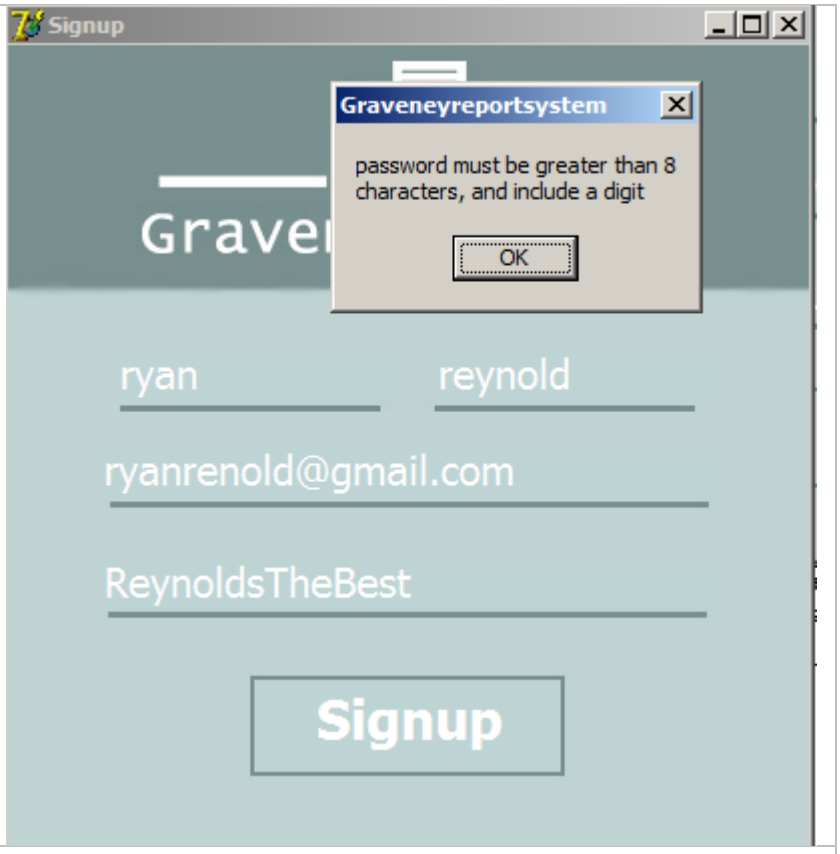
Test No 3.2.1 result: making sure system does not create an account when name is invalid…

Entering the values and pressing signup button	
Summary: Test Successful – The system is able to validate the name. In this case, the system is able to recognise first name is invalid, and not create an account. It is able to inform user why the account as not been created. Requirement 5.4 and 5.6 has been met.	

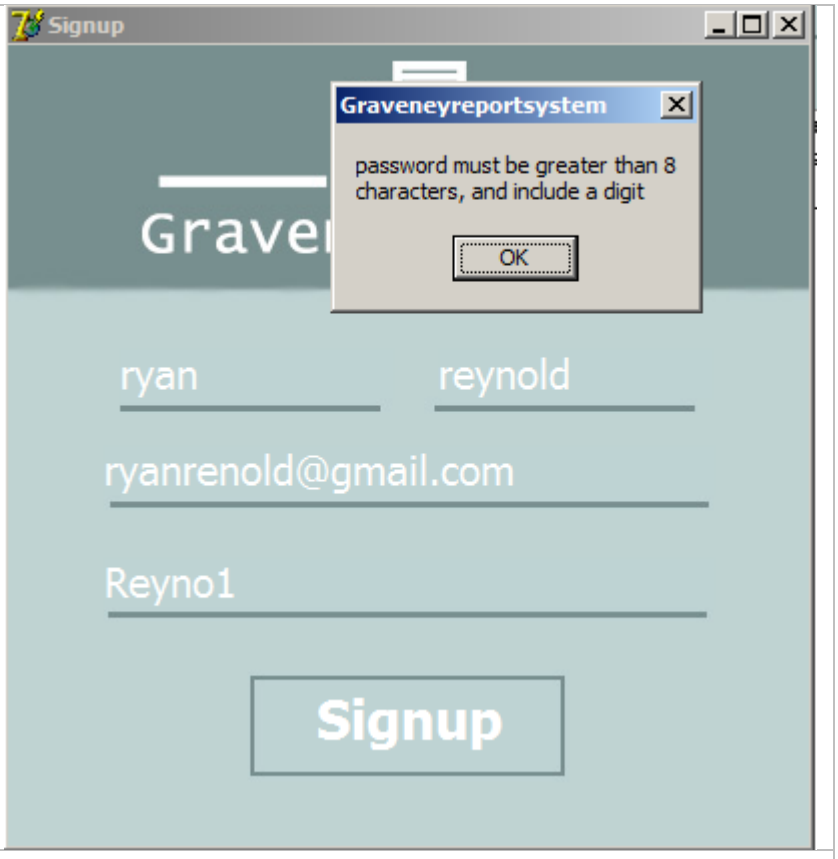
Test No 3.2.2 result: making sure system does not create an account when name is invalid…

Entering the values and pressing signup button	
Summary: Test Successful – The system is able to validate the name. In this case, the system is able to recognise last name is invalid, and not create an account. It is able to inform user why the account as not been created. Requirement 5.4 and 5.6 has been met.	

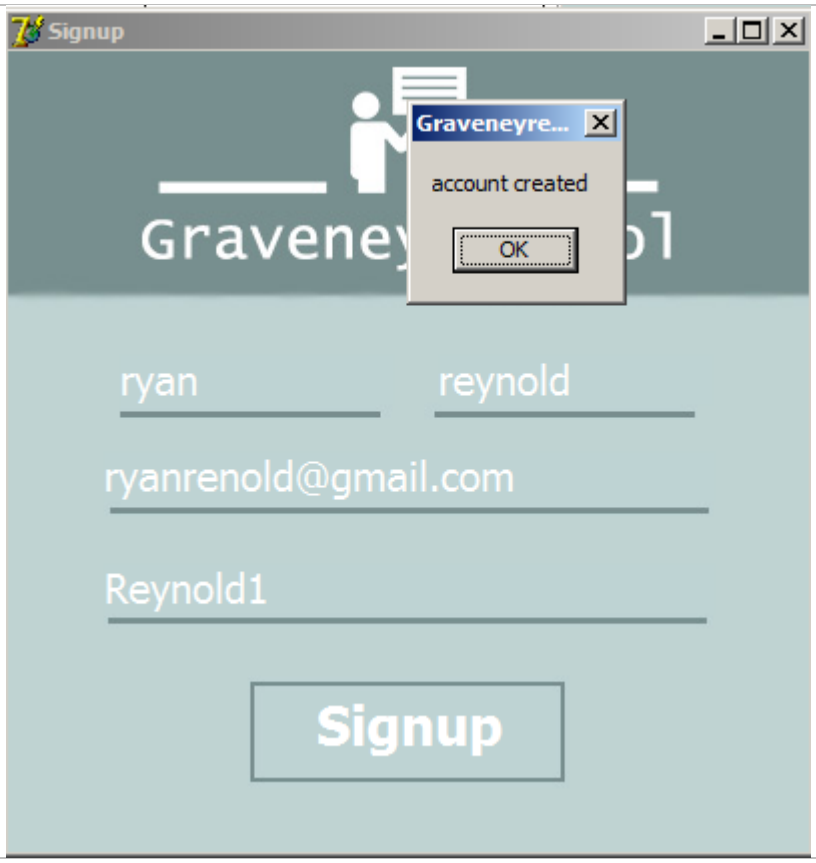
Test No 3.3.1 result: making sure system does not create an account when password is invalid…

Entering the values and pressing signup button	
Summary: Test Successful – The system is able to validate the password. In this case, the system is able to recognise password does not contain a digit, and does not create an account. It is able to inform user why the account as not been created. Requirement 5.4 and 5.5 has been met.	

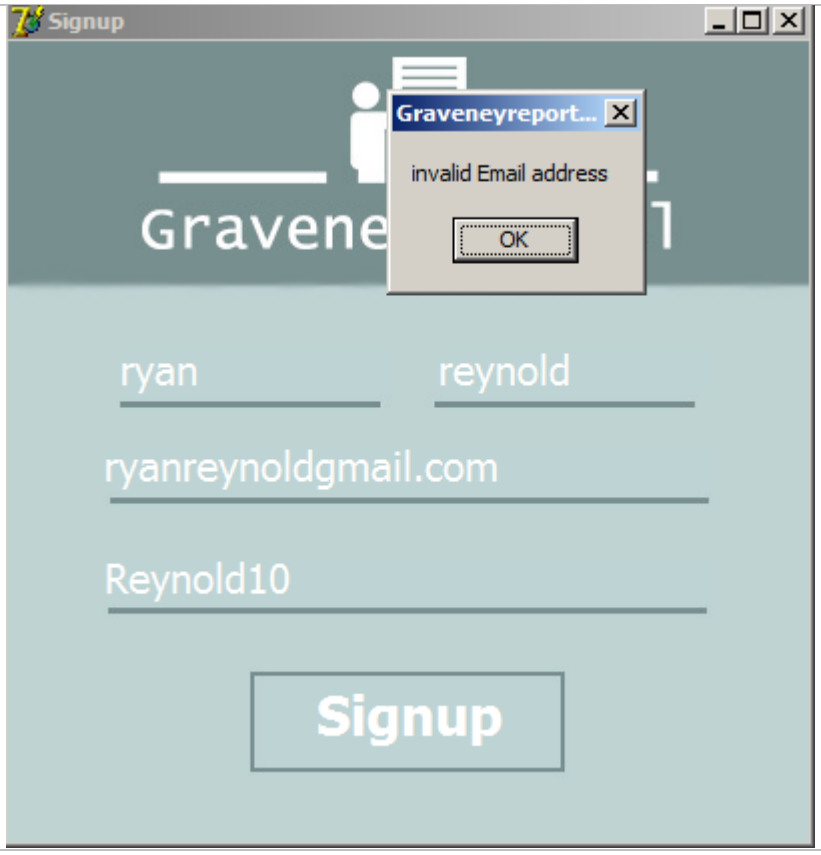
Test No 3.3.2 result: making sure system does not create an account when password is invalid…

Entering the values and pressing signup button	
Summary: Test Successful – The system is able to validate the password. In this case, the system is able to recognise password is less than 8 characters, and does not create an account. It is able to inform user why the account as not been created. Requirement 5.4 and 5.5 has been met.	

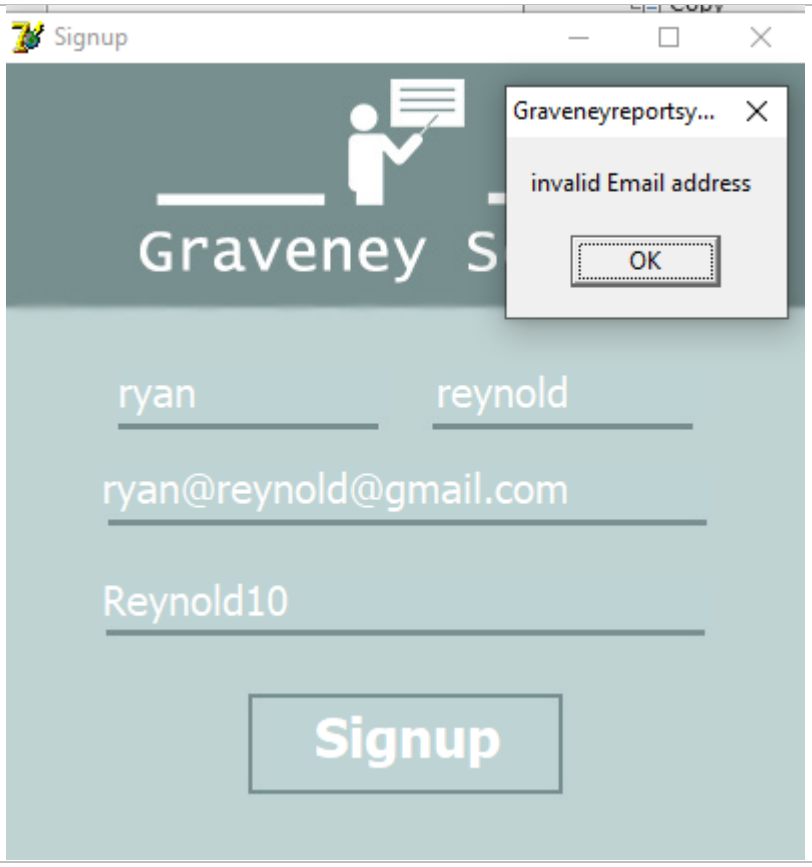
Test No 3.3.3 result: making sure system does not create an account when password is invalid…

Entering the values and pressing signup button	
Summary: Test Successful – The system is able to create an account, when password length is 8 (on boundary) and have it stored within the database. This allows the user to login. The system also prompts the user a message when the account has been created. Requirement 5.3, 5.5, and 5.1 has been met.	

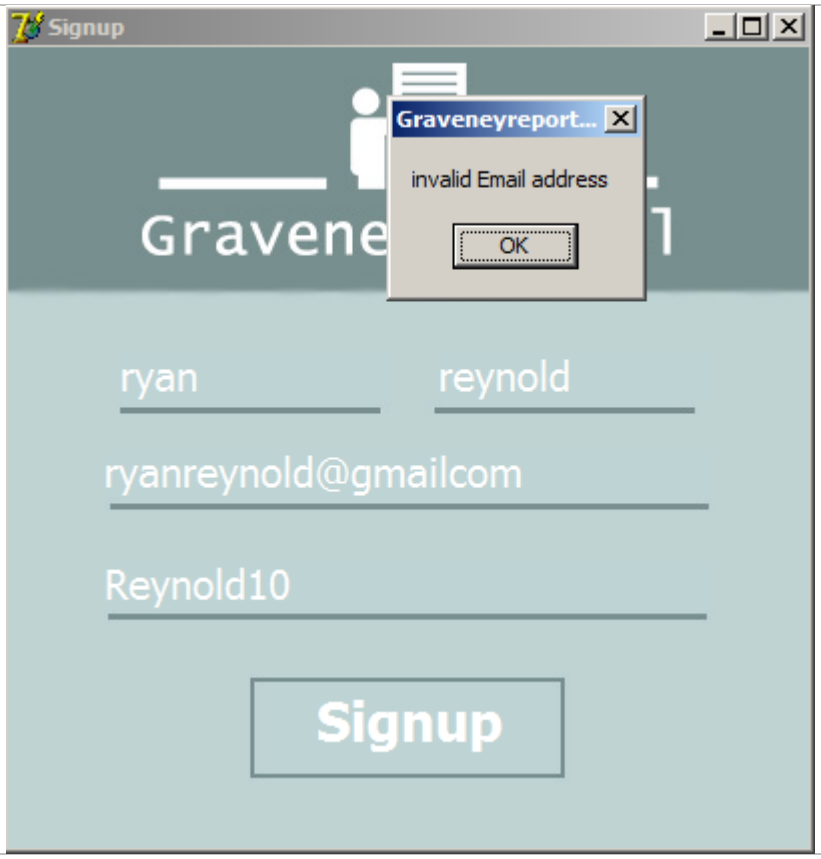
Test No 3.4.1 result: making sure system does not create an account when email is invalid…

Entering the values and pressing signup button	
Summary: Test Successful – The system is able to detect when the email is invalid. In this case, the system cannot detect any mail server in the email address. An email address is required a '@' to indicate what the mail server is. The system was able to tell the user that the email address is invalid. The system does not create the account. Requirement 7 met.	

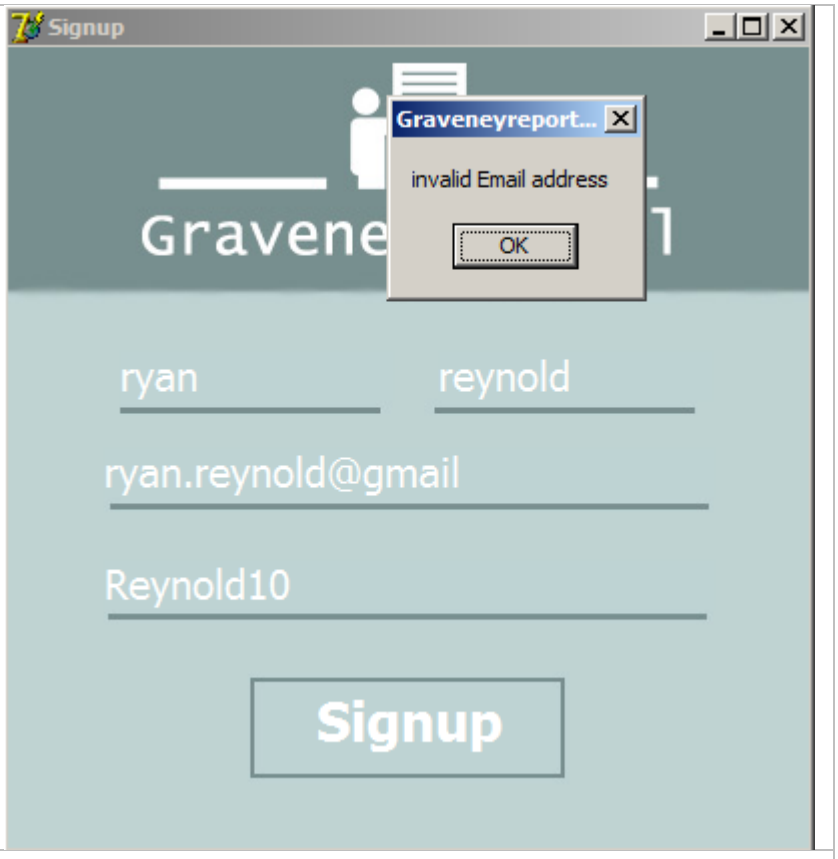
Test No 3.4.2 result: making sure system does not create an account when email is invalid…

Entering the values and pressing signup button	
Summary: Test Successful – The system is able to detect when the email is invalid. In this case, the system detects more than one '@' symbol. The system was able to tell the user that the email address is invalid. The system does not create the account. Requirement 7 met.	

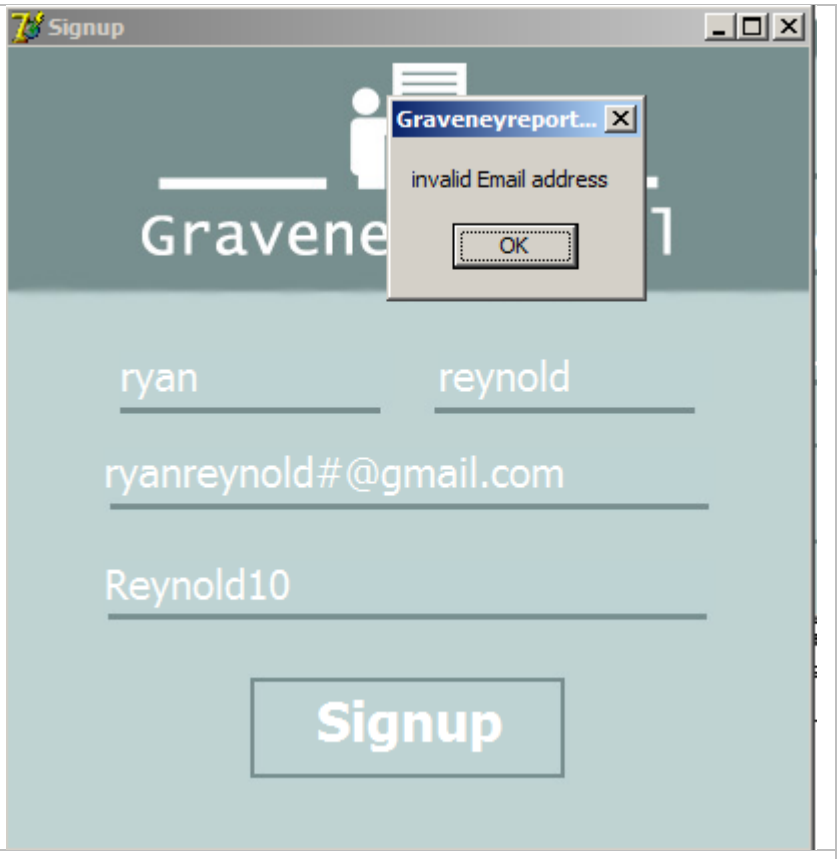
Test No 3.4.3 result: making sure system does not create an account when email is invalid…

Entering the values and pressing signup button	
Summary: Test Successful – The system is able to detect when the email is invalid. In this case, the system cannot detect any domain in the email address. An email address is required a domain. The system was able to tell the user that the email address is invalid. The system does not create the account. Requirement 7 met.	

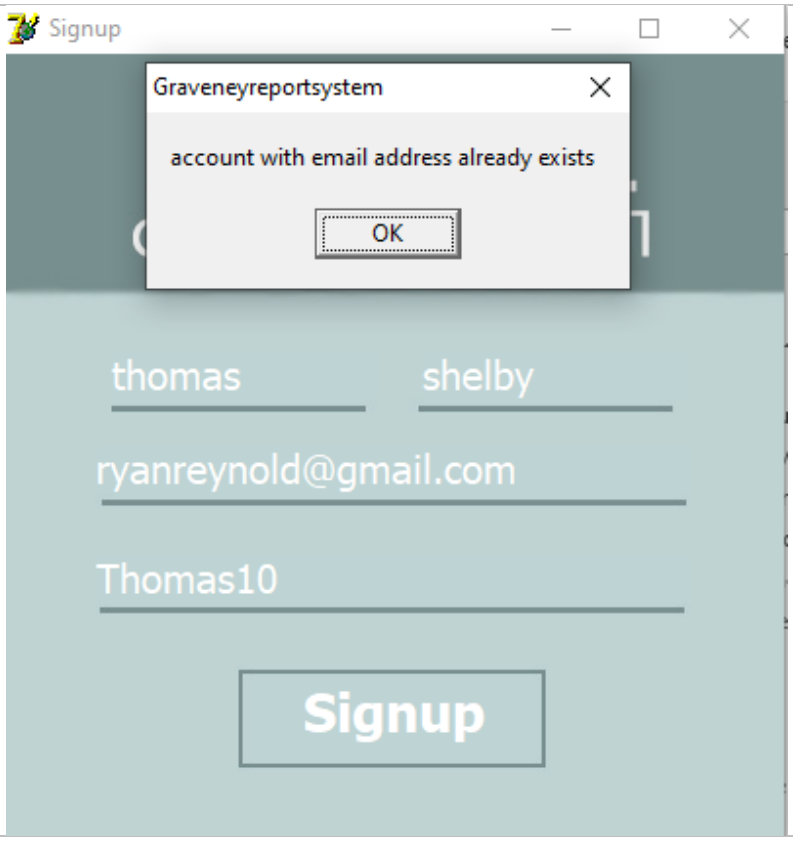
Test No 3.4.4 result: making sure system does not create an account when email is invalid…

Entering the values and pressing signup button	
Summary: Test Successful – The system is able to detect when the email is invalid. In this case, the system cannot detect any domain in the email address. It does not mistake the domain as 'reynold@gmail'. An email address is required a domain. The system was able to tell the user that the email address is invalid. The system does not create the account. Requirement 7 met.	

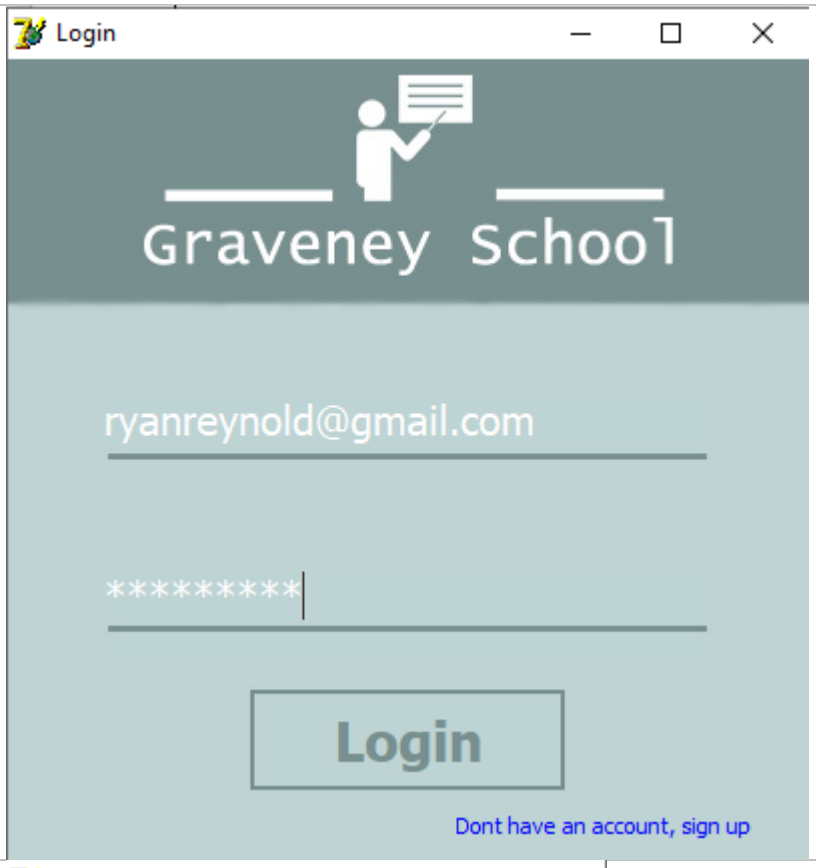
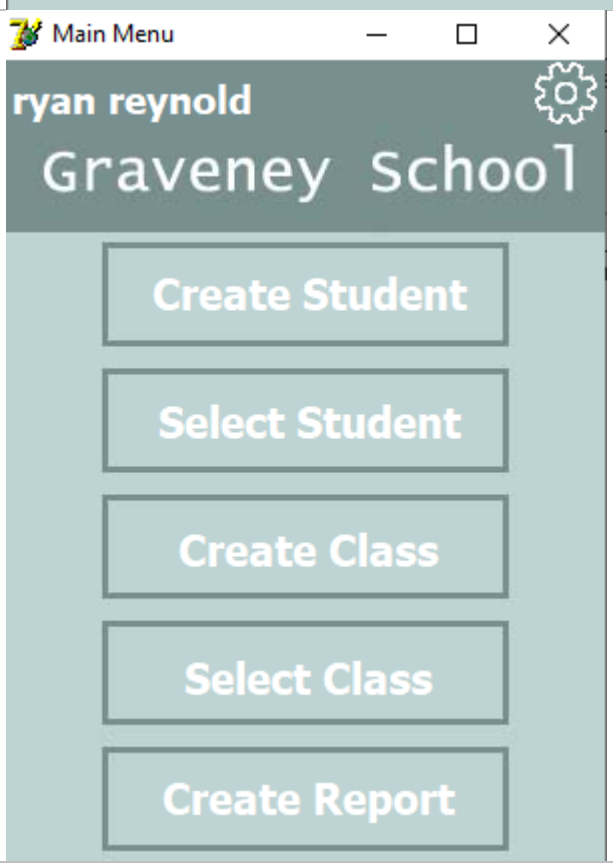
Test No 3.4.5 result: making sure system does not create an account when email is invalid…

Entering the values and pressing signup button	
Summary: Test Successful – The system is able to detect when the email is invalid. In this case, it was able to spot the character '#', which in not a valid character in email addresses. The system was able to tell the user that the email address is invalid. The system does not create the account. Requirement 7 met.	

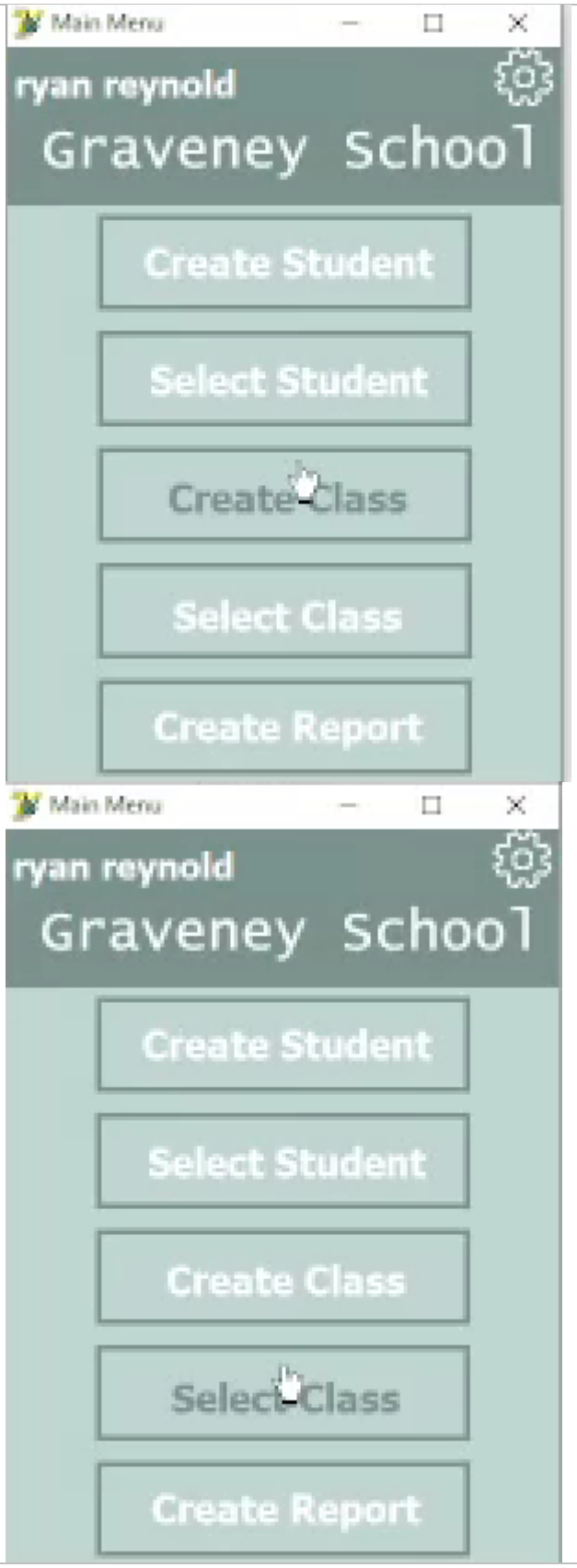
Test No 3.5 result: making sure system does not create an account when account already exists…

Entering the values and pressing signup button	
Summary: Test Successful – The system is able to detect if the account already exists and is able to inform the user. The system does not create the account.	

Test No 3.6 result: Checking if users name displays on main menu…

Step 1) Entering the values…	
Step 2) Pressing login button	
Summary: Test Successful – The system is able to display the users name on main menu in the correct position.	

Test No 3.7 result: checking if system changes button colour when cursor hovering on top…

Hovering Cursor on button	
Summary: Test Successful – The system is able to change the colour only when the mouse is on top, otherwise it is white.	

Prototype 2

Prototype 2 is a more efficient way of changing the colour of the buttons for the main menu form.

Changes in Implemented Code for "Main Menu Form" (unit_menu)

```
…
type
  TfrmMainMenu = class(TForm)
  published
    lblProfileName: TLabel;
    ImgMainMenu: TImage;
    btnCreateStudent: TLabel;
    btnSelectStudent: TLabel;
    btnCreateClass: TLabel;
    btnSelectClass: TLabel;
    btnCreateReport: TLabel;
    imgSettings: TImage;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure btnCreateClassClick(Sender: TObject);
    procedure btnSelectClassClick(Sender: TObject);
    procedure btnCreateStudentClick(Sender: TObject);
    procedure btnMouseEnter(Sender: TObject);
    procedure btnMouseLeave(Sender: TObject);
    procedure btnSelectStudentClick(Sender: TObject);
    procedure btnCreateReportClick(Sender: TObject);
    procedure imgSettingsClick(Sender: TObject);
end;

var
  frmMainMenu: TfrmMainMenu;

implementation

uses unit_login;

{$R *.dfm}

procedure TfrmMainMenu.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  frmlogin.close;
end;

//Changes Color, lets user know that its a button
procedure TfrmMainMenu.btnMouseEnter(Sender: TObject);
begin
  TLabel(Sender).Font.Color := RGB(119,143,143);
end;

procedure TfrmMainMenu.btnMouseLeave(Sender: TObject);
begin
  TLabel(Sender).Font.Color := clWhite;
end;

//Create student button - label component used as a button
procedure TfrmMainMenu.btnCreateStudentClick(Sender: TObject);
begin
  //open form
end;
```

```
//Select student button - label component used as a button
procedure TfrmMainMenu.btnSelectStudentClick(Sender: TObject);
begin
  //open form
end;

//Create Class button - label component used as a button
procedure TfrmMainMenu.btnCreateClassClick(Sender: TObject);
begin
  //open form
end;

//Select Class button - label component used as a button
procedure TfrmMainMenu.btnSelectClassClick(Sender: TObject);
begin
  //open form
end;

//Create Report button - label component used as a button
procedure TfrmMainMenu.btnCreateReportClick(Sender: TObject);
begin
  //keep open
end;

//update user details button – image component used as a button
procedure TfrmMainMenu.imgSettingsClick(Sender: TObject);
begin
  //open form
end;

end.
```

What has been done

Instead of using separate procedures for mouse enter and mouse leave events, for each object, prototype 2 allows a single procedure for mouse enter and one for mouse leave to be used for all objects, reducing it from ten procedures to only two coded.

Testing of solution

Retest No 3.7 result: checking if system changes button colour when cursor hovering on top…

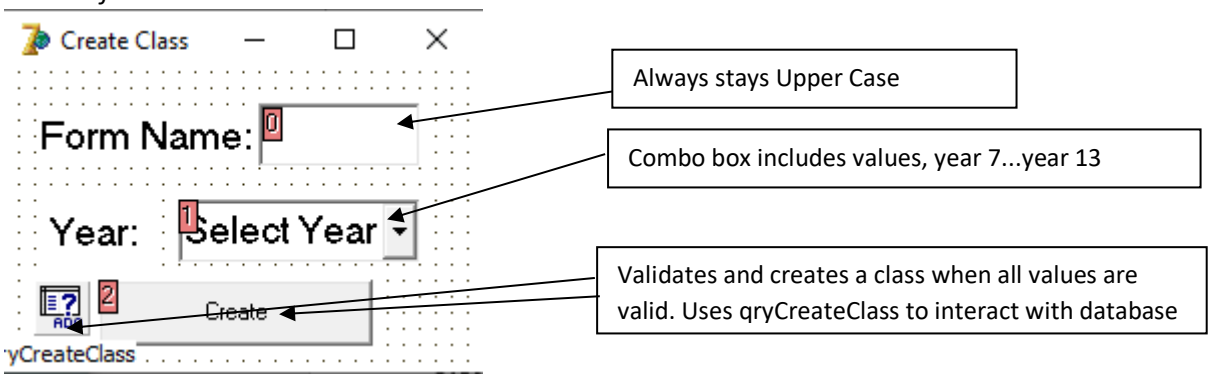
Summary: Test Successful – The system is still able to change the colour only when the mouse is on top, otherwise it is white.

Stage 4: Implementing the "Create Class Form"

The Create Class form allows users to enter the details of a class they want to create. Classes play a key part in the system. It organises students, making it easier for the user to find a student to create a report.

Prototype 1

Screenshot of "Create Class Form"



Implemented Code for Create Class Form (unit_createClass)

```
unit unit_createClass;

interface

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

type
  TfrmCreateClass = class(TForm)
  published
    lbledtFormclassName: TLabeledEdit;
    cboSelectYear: TComboBox;
    btnCreateClass: TButton;
    qryCreateClass: TADOQuery;
    lblYear: TLabel;
    procedure btnCreateClassClick(Sender: TObject);
  end;

var
  frmCreateClass: TfrmCreateClass;

implementation

{$R *.dfm}

//Checks if class already exists under the user
function ClassExists(formclassName:string; year:Integer):boolean;
var res:Boolean;
begin
  frmcreateclass.qrycreateclass.SQL.Text := 'Select classID from class where
name='+quotedStr(formclassName)+' and stfID='+inttostr(info.staffID)+' and
year='+inttostr(year)+';';
  frmCreateClass.qryCreateClass.Open;
  //qry will either have 0 or 1 records. If it is not 0 then class already
exists.
```

```
  if (frmCreateClass.qryCreateClass.RecordCount>0)then
    res:=True
  else
    res:=false;
  frmCreateClass.qryCreateClass.close;
  ClassExists := res;
end;

//verifys the class name
function verifyClassName(formclassName:string; year:Integer):boolean;
var res:Boolean;
begin
  if (((Length(formclassName)=1) and (year>=7) and
(year<=11))or((Length(formclassName)=3) and ((year=12) or (year=13))))then
    res:=True
  else
    res:= false;
  verifyClassName := res;
end;

//write class to database and infroms user
procedure writeclass(name:string;year,staffID:Integer);
begin
  frmcreateclass.qrycreateclass.SQL.Text := 'Insert into class(name,
year,StfID)
values('+quotedStr(name)+','+inttostr(year)+','+inttostr(staffID)+');';
  frmCreateClass.qryCreateClass.ExecSQL;
  ShowMessage('class created');
end;

//when create class button is clicked, verifys all data and creates class.
procedure TfrmCreateClass.btnCreateClassClick(Sender: TObject);
var alreadyExists, classNameVerified:Boolean;
  className:string;
  year:Integer;
begin
  className:= lbledtFormclassName.Text;
  year:= cboSelectYear.ItemIndex + 7;//year 7 is on index 0, and so on.
Therefore +7.
  alreadyExists:= ClassExists(className, year);
  classNameVerified:= verifyClassName(className,year);
  if (cboSelectYear.ItemIndex = -1)or(className='')then
    ShowMessage('Fill in the values to create class...')
  else if(classNameVerified)then
  begin
    if (alreadyExists)then
      ShowMessage('class exists...')//infroms user when class exists
    Else
      //creates class when all data is valid
      writeClass(className,year,info.staffID);
  end
  else
    ShowMessage('Invalid class name');//Infroms user, when class name is
invalid
end;

end.
```

Validation carried out for "Create Class Form" (unit_createClass)

Validation of Class Name	The Class name must be valid to create a class. Classes of year 7 to 11 must be one character in length, whilst year 12 and 13's must be 3 characters in length. This is the format of Graveney School's class name.
Validation of Class Existence	The Class will only be created if it does not already exist.
Validates if form is empty	If the value of the name is null or if the user did not select a year for the class, the system displays a message to the user informing them to fill in the missing values

Changes in Implemented Code for "Main Menu Form" (unit_menu)

Coded Section Before	Coded Section After

`procedure TfrmMainMenu.btnCreateClassClick(Sender: TObject);`
`begin`
`  //open form`
`end;`
	`procedure TfrmMainMenu.btnCreateClassClick(Sender: TObject);`
	`begin`
	`  //open form`
	`  frmCreateClass.ShowModal;`
	`end;`

`uses unit_login;`	`uses unit_login, unit_createClass;`
Justification: Allow user to open the Create Class form from main menu.	

What has been done

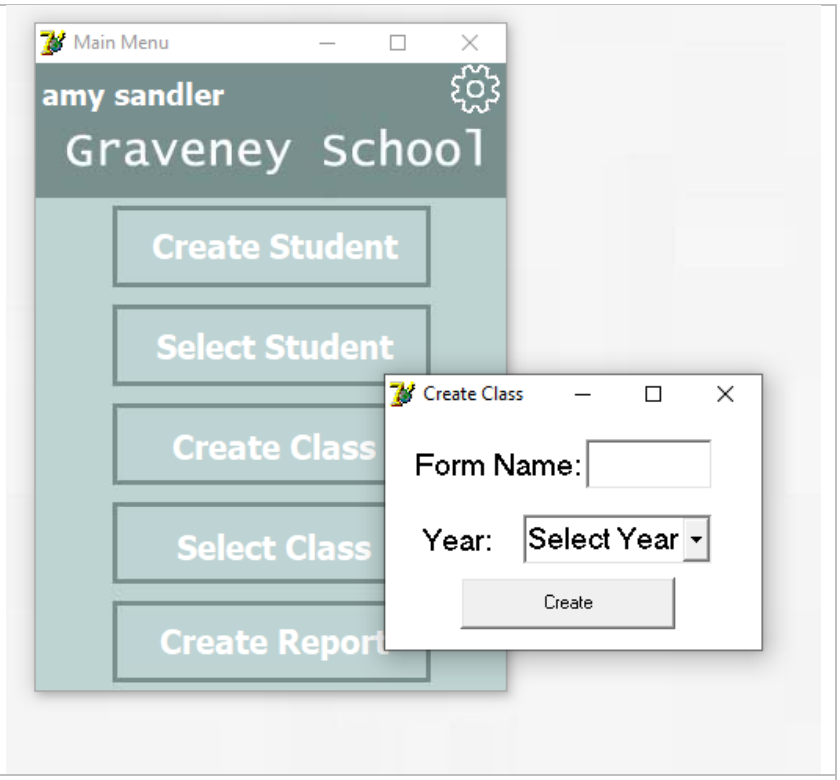
The Create Class Form has been implemented. Criteria 4 has been met. It allows the user to create a class, hence requirement 12 has been met. The Create Class Form can also be open from the main menu, hence requirement 8.3 has been partially met. The prototype was tested in runtime of the code. Screenshots of results of the tests are shown below.

Summary: The user is able to create and sign in using an account. The main menu form has also been created, which will allow users to open different forms for the solution. They are also able to create classes once logged in. The create class form can be opened from main menu.

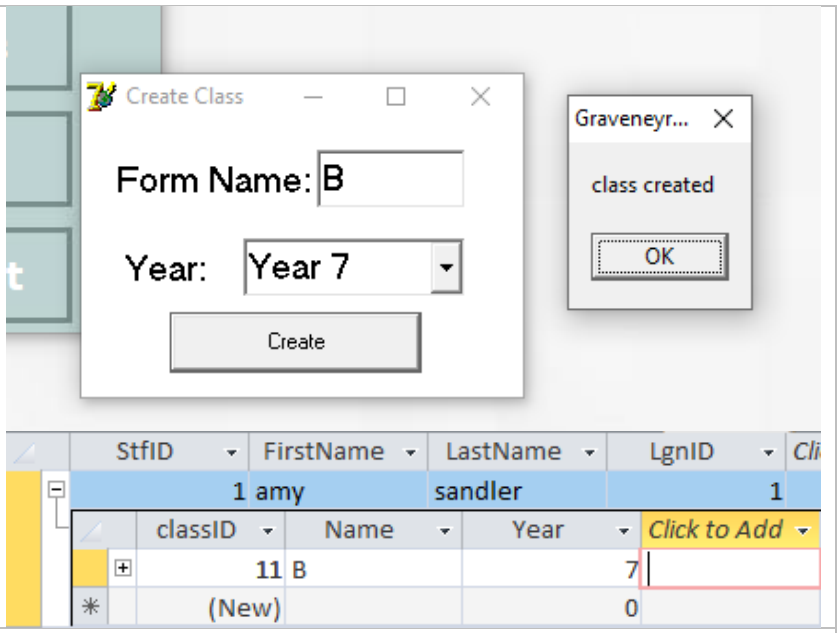
Testing of solution

Test carried out are 4.1, 4.2, 4.3, 4.4, 4.5.

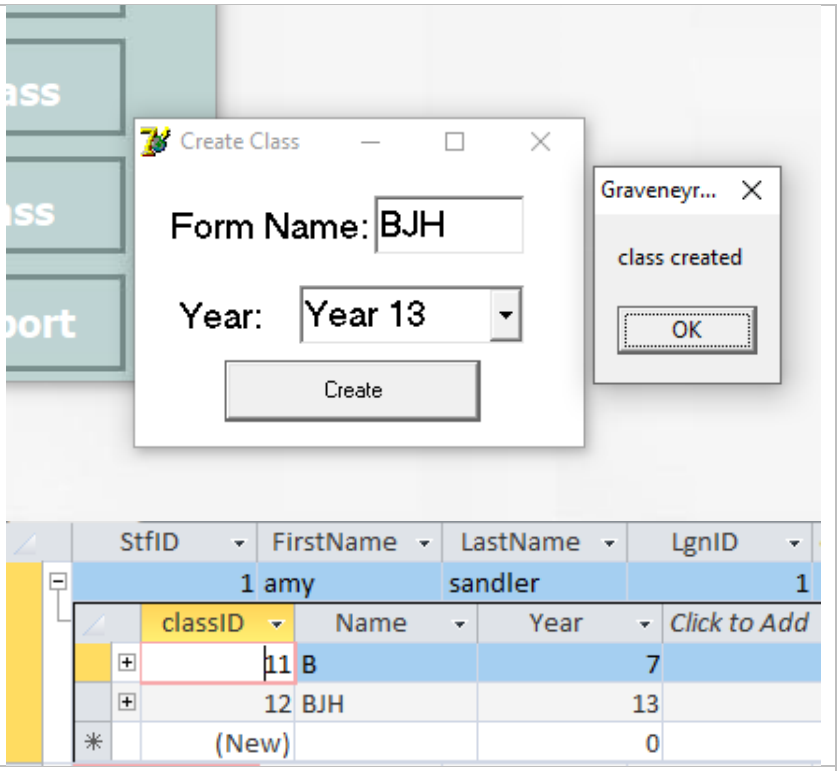
Test No 4.1 result: checking if users can open create class form…

Pressing Create Class button on Main Menu	
Summary: Test Successful – The Main Menu is able to open the create class form, when the user presses the button.	

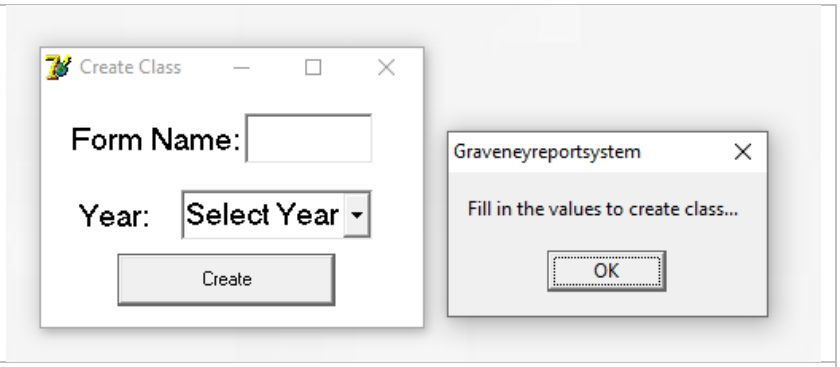
Test No 4.2.1 result: checking if users can create a class…

Typing the Values and clicking create button	
Summary: Test Successful – The Create class form is able to create a class, with valid values, and inform the user that the class has been created.	

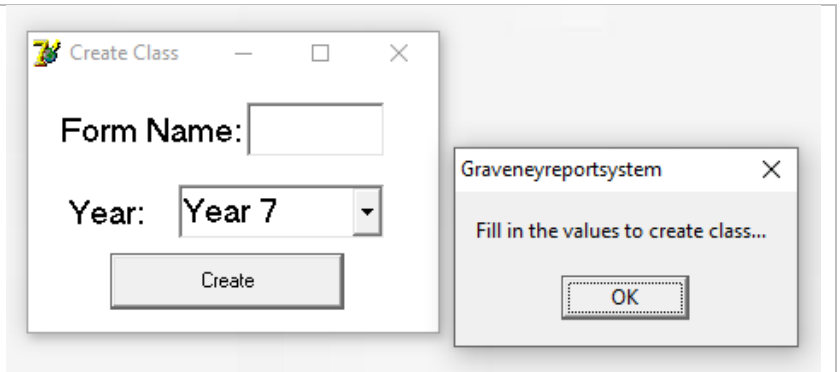
Test No 4.2.2 result: checking if users can create a class…

Typing the Values and clicking create button	
Summary: Test Successful – The Create class form is able to create a class, with valid values, and inform the user that the class has been created.	

Test No 4.3.1 result: checking result of creating a class, when form is not filled in…

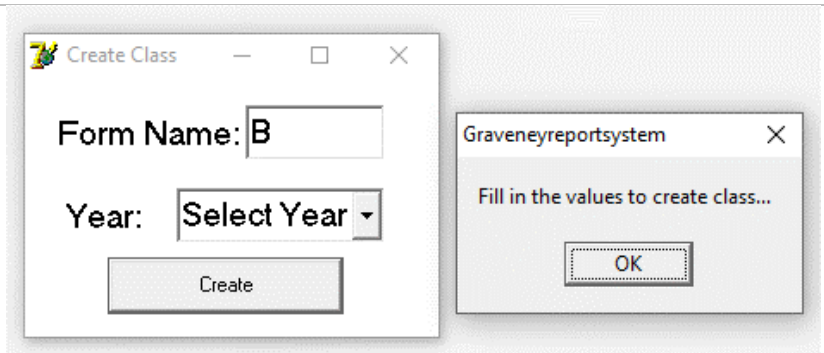
clicking create button	
Summary: Test Successful – The Create class form is able to inform user when the form is not complete, and does not create the class.	

Test No 4.3.2 result: checking result of creating a class, when value of name is null…

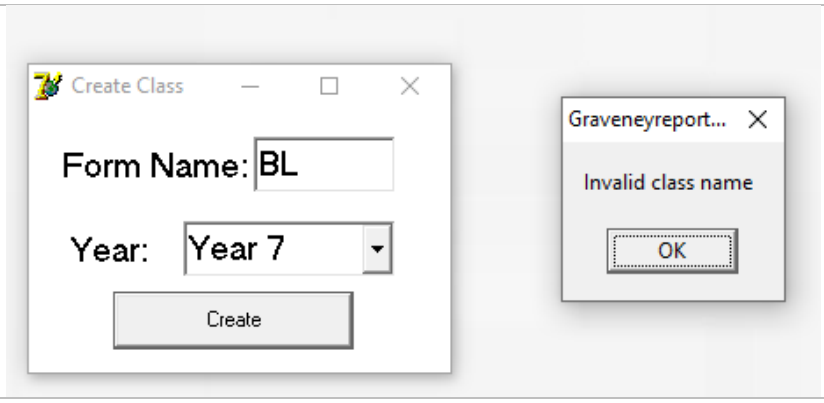
Selecting year and clicking create button	

Summary: Test Successful – The Create class form is able to inform user when the form is not complete, and does not create the class.

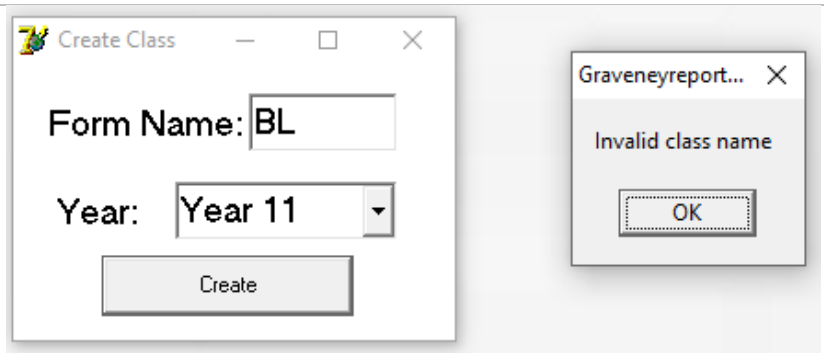
Test No 4.3.3 result: checking result of creating a class, when year is not selected…

Entering value and clicking create button	
Summary: Test Successful – The Create class form is able to inform user when the form is not complete, and does not create the class.	

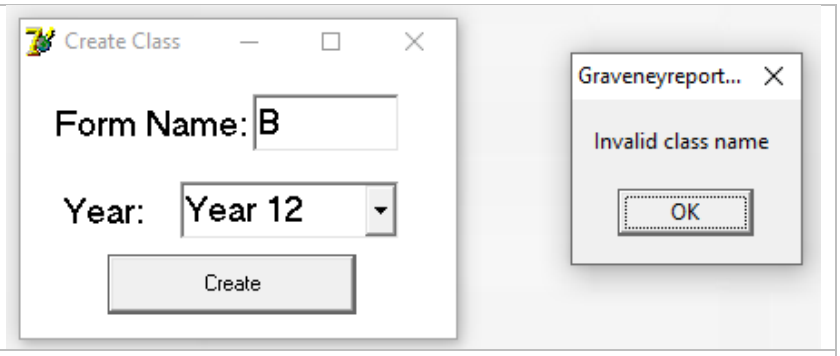
Test No 4.4.1 result: checking if system can recognise invalid name…

Typing the values and clicking create button	
Summary: Test Successful – The Create Class form is able to recognise class name length is not one, therefore invalid, and not create the class. It is then able to tell the user why the class has not been created.	

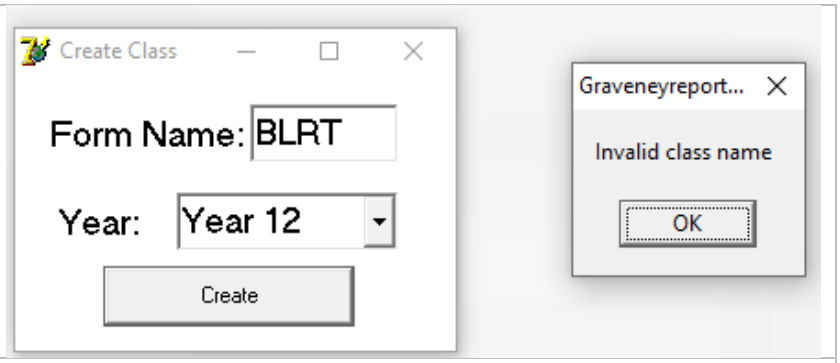
Test No 4.4.2 result: checking if system can recognise invalid name…

Typing the values and clicking create button	
Summary: Test Successful – The Create Class form is able to recognise class name length is not one, therefore invalid, and not create the class. It is then able to tell the user why the class has not been created.	

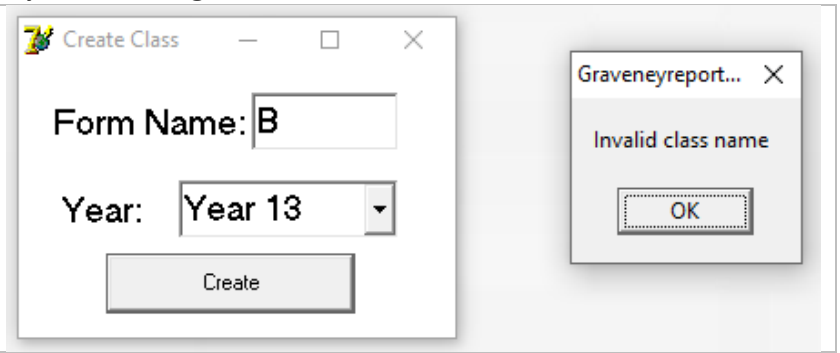
Test No 4.4.3 result: checking if system can recognise invalid name…

Typing the values and clicking create button	
Summary: Test Successful – The Create Class form is able to recognise class name length is not three, therefore invalid, and not create the class. It is then able to tell the user why the class has not been created.	

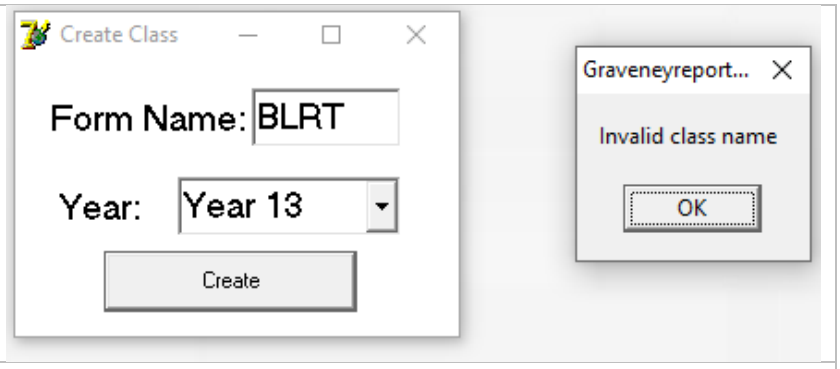
Test No 4.4.4 result: checking if system can recognise invalid name…

Typing the values and clicking create button	
Summary: Test Successful – The Create Class form is able to recognise class name length is not three, therefore invalid, and not create the class. It is then able to tell the user why the class has not been created.	

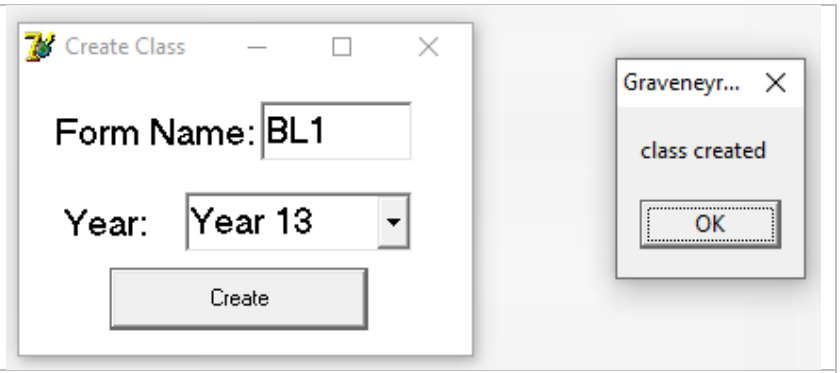
Test No 4.4.5 result: checking if system can recognise invalid name…

Typing the values and clicking create button	
Summary: Test Successful – The Create Class form is able to recognise class name length is not three, therefore invalid, and not create the class. It is then able to tell the user why the class has not been created.	

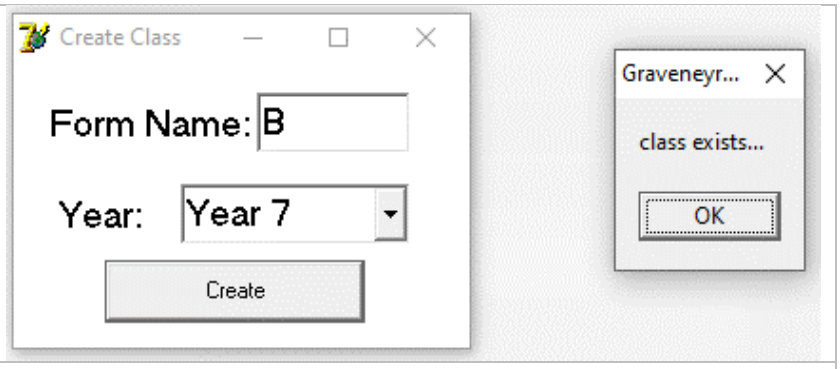
Test No 4.4.6 result: checking if system can recognise invalid name…

Typing the values and clicking create button	
Summary: Test Successful – The Create Class form is able to recognise class name length is not three, therefore invalid, and not create the class. It is then able to tell the user why the class has not been created.	

Test No 4.4.7 result: checking if system can recognise invalid name…

Typing the values and clicking create button	
Summary: Test failed – Code has not been implemented to check if form name consists of only alphabetical characters, therefore Create Class form takes value BL1 as a valid value, and creates the class. I will have to write a set of validation in order to avoid erroneous data like this being inputted. This will be done in the next prototype.	

Test No 4.5 result: checking if system can check if class already exists…

Typing the values and clicking create button	
Summary: Test Successful – The system is able to recognise if class already exist and not create another class. The system is able to inform user that the class already exists.	

Prototype 2

Prototype 2 fixes the logical error in prototype 1, where the user was able to create a class, with the name including an integer. I have implemented the required validation needed to deny any erroneous data as displayed in the failed test.

Changes in Implemented Code for "Signup Form" (unit_signup) : Attempt to resolve issue seen in test 4.4.7 carried out in prototype 1

```
…
  Public
    function stringisAlpha(name:string):Boolean; //used in create class
  end;
…
function Tfrmsignup.stringisAlpha(name:string):Boolean;
const charslist = ['A'..'Z', 'a'..'z'];
var i:Integer;
    problem:Boolean;
begin
  problem:=false;//initial value is false
  for i:=1 to Length(name) do
  begin
    if not((name[i] in charsList))then
      problem:=true; // Contains a character, which is not an alphebet.
  end;
  stringisAlpha:=not(problem);
end;


function verifyName_and_Password(fname,lname,password:string):Boolean;
var outcome:Boolean;
begin
  outcome:=false;
  //Name must only consist of alphabetical characters
  if not((frmSignup.stringisAlpha(fname))and
(frmSignup.stringisAlpha(lname)))then
…
```

The Grey highlighted text is what has been modified from stage 3.
Justification: The function, stringisAlpha, has been made public. The function will be used within unit_createClass to check if the form name is a string or not.

Changes in Implemented Code for "Create Class Form" (unit_createClass)

Coded Section Before	Coded Section After

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

Function verifyClassName(formclassName:string; year:Integer):boolean;
var res:Boolean;
begin
  if (((Length(formclassName)=1) and (year>=7) and (year<=11))or((Length(formclassName)=3) and ((year=12) or (year=13))))then
    res:=True
  else
    res:= false;
  verifyClassName := res;
end;
	function verifyClassName(formclassName:string; year:Integer):boolean;
	var res:Boolean;
	begin
	  res:= false;
	  if (((Length(formclassName)=1) and (year>=7) and (year<=11))or((Length(formclassName)=3) and ((year=12) or (year=13))))then
	  begin
	    if (frmSignup.stringisAlpha(formclassName)) then
	      res:=True;
	  end;
	  verifyClassName := res;

	`end;`
Justification: unit_signup is referenced in unit_createClass so that its able to use its public resources. Function verifyClassName has been modified. It uses the function stringisAlpha, from signup, to make sure name only consists of alphabetical characters.	

Validation carried out for "Create Class Form" (unit_createClass)

Validation of Class Name being alphabetical	Class name must only consist of alphabetical charaters to be valid.

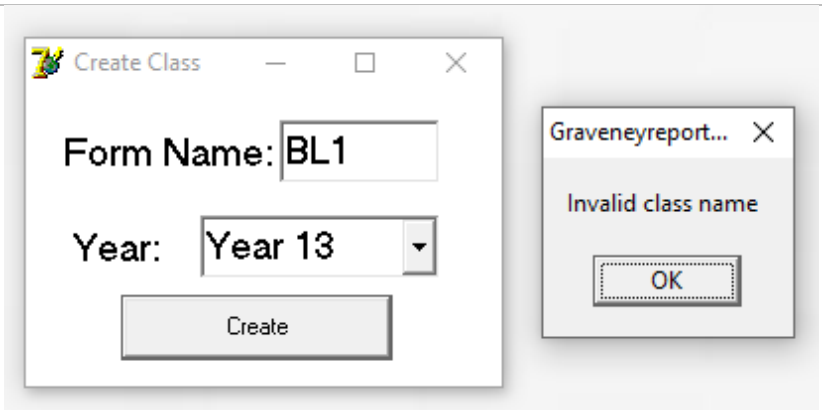
What has been done

Signup form and create Class form have been modified to have a successful test 4.4.7. Create Class Form is able to check if the class name only consists of alphabetical characters, hence successfully meeting requirements 12.3.

Summary: The user is able to create and sign in using an account. The main menu form has also been created, which will allow users to open different forms for the solution. They are also able to create classes once logged in. The create class form can be opened from main menu.

Testing of solution

Test No 4.4.7 result: checking if system can recognise invalid name…

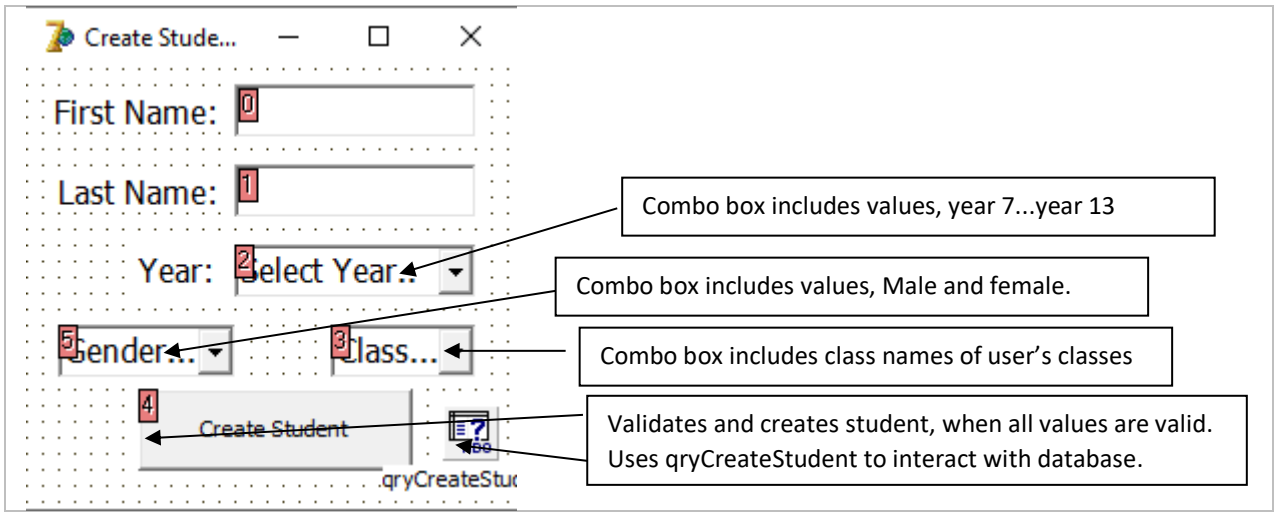
Typing the values and clicking create button	
Summary: Test Successful – The system is able to check that the form name only contains alphabetical characters. The system does not create the class when name is invalid and informs user.	

Stage 5: Implementing the "Create Student Form"

Create student form allows users to store students, allowing them to create reports of the student.

Prototype 1

Screenshot of "Create Student Form"



Changes in Implemented Code for "Main Menu Form" (unit_menu)

Coded Section Before	Coded Section After

`procedure TfrmMainMenu.btnCreateStudentClick(Sender: TObject);`
`begin`
`  //open form`
`end;`
	`procedure TfrmMainMenu.btnCreateStudentClick(Sender: TObject);`
	`begin`
	`  //open form`
	`  frmCreateStudent.ShowModal;`
	`end;`

`uses unit_login, unit_createClass;`	`uses unit_login, unit_createClass, unit_createStudent;`
Justification: Allow user to open the Create Student form from main menu.	

Implemented Code for Create Student Form (unit_createStudent)

```
unit unit_createStudent;

interface

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

type
  TfrmCreateStudent = class(TForm)
  published
    lbledtStudent_FirstName: TLabeledEdit;
    lbledtStudent_LastName: TLabeledEdit;
    cboSelectYear: TComboBox;
    cboSelectClass: TComboBox;
    qryCreateStudent: TADOQuery;
    btnCreateStudent: TButton;
    cboSelectGender: TComboBox;
```

```
    lblYear: TLabel;
    procedure cboSelectYearChange(Sender: TObject);
    procedure btnCreateStudentClick(Sender: TObject);
  end;

var
  frmCreateStudent: TfrmCreateStudent;

implementation

{$R *.dfm}

//The form adds the correspoding items in class name combo box to the year
selected
procedure TfrmCreateStudent.cboSelectYearChange(Sender: TObject);
var i:Integer;
begin
  qrycreatestudent.SQL.Text := 'Select name from class where
year='+inttostr(cboSelectYear.ItemIndex+7)+' and
stfID='+inttostr(info.staffID)+';';
  cboSelectClass.Clear; //Get rid of any values added corresponding to another
year
  qryCreateStudent.Open;
  for i:=1 to qrycreatestudent.recordcount do
  begin
    cboSelectClass.Items.Add(qryCreateStudent.fieldbyname('name').AsString);
    qryCreateStudent.Next; //goes on to the next record
  end;
  qryCreateStudent.Close;
end;

//Checks if student exists, else Creates class
procedure writeStudent;
var classID:String;
begin
  //Getting ClassID of the selected class
  frmCreateStudent.qrycreatestudent.SQL.Text := 'Select classID from class
where year='+inttostr(frmCreateStudent.cboSelectYear.ItemIndex+7)+' and
stfID='+inttostr(info.staffID)+' and
name='+quotedStr(frmCreateStudent.cboselectclass.Text)+';';
  frmCreateStudent.qryCreateStudent.Open;
  ClassID :=
frmCreateStudent.qryCreateStudent.fieldbyname('classID').AsString;
  frmCreateStudent.qryCreateStudent.Close;

  //checking if student already exists in class
  frmCreateStudent.qrycreatestudent.SQL.Text := 'Select FirstName, LastName
from student where classID='+classID+' and
FirstName='+quotedStr(frmCreateStudent.lbledtStudent_FirstName.Text)+' and
LastName='+quotedStr(frmCreateStudent.lbledtStudent_LastName.Text)+' and
Gender='+quotedStr(frmCreateStudent.cboSelectGender.text)+';';
  frmCreateStudent.qryCreateStudent.Open;
  if (frmCreateStudent.qryCreateStudent.RecordCount>0)then
  begin
    showmessage('Already Exists...'); //informing user student already exists;
does not create student
    frmCreateStudent.qryCreateStudent.Close;
    end
  else
  begin
    //adding values to student_Table, Creating the student
```

```
    frmCreateStudent.qryCreateStudent.SQL.Text := 'insert into
student(FirstName, LastName,Gender, ClassID) values
('+quotedStr(frmCreateStudent.lbledtStudent_FirstName.Text)+','+quotedStr(frmC
reateStudent.lbledtStudent_LastName.Text)+','+quotedStr(frmCreateStudent.cboSe
lectGender.text)+','+classID+');';
    frmCreateStudent.qryCreateStudent.ExecSQL;
    showmessage('Student Created'); //informing user student has been created
  end;
end;

//When user clicks on create student button…
procedure TfrmCreateStudent.btnCreateStudentClick(Sender: TObject);
begin
  //checks if form is filled out
  if not((lbledtStudent_FirstName.Text = '') or (lbledtStudent_LastName.Text =
'') or (cboSelectGender.ItemIndex = -1) or (cboSelectYear.ItemIndex = -1) or
(cboSelectClass.ItemIndex = -1) or (cboSelectGender.ItemIndex = -1))then
  begin
    //Validates student name; Only consists Alphabetical Characters
    if( (frmSignup.stringisAlpha(lbledtStudent_FirstName.Text)) and
(frmSignup.stringisAlpha(lbledtStudent_LastName.Text)))then
      writeStudent
    else
      ShowMessage('invalid name'); //Informs user if name is invalid
  end
  else
    ShowMessage('fill in all values'); //Informs user the form is not filled
out
end;

end.
```

Highlighted Code is changed in Prototype 2

Validation carried out for "Create Class Form" (unit_createClass)

Validation of Student Name	Name must only be Alphabetical characters, otherwise invalid.
Validation of Student Existence.	The student will only be created if it does not already exist.
Validates if form is empty	If the value of the name is null or if the user did not select a year, class and gender for the student, the system displays a message to the user informing them to fill in the missing values, and does not create the student.

What has been done

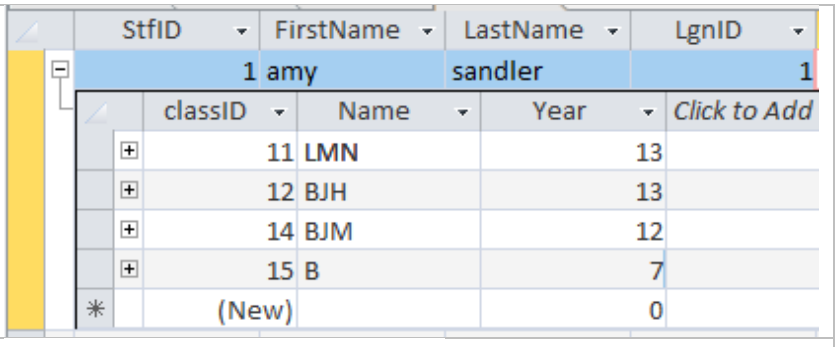
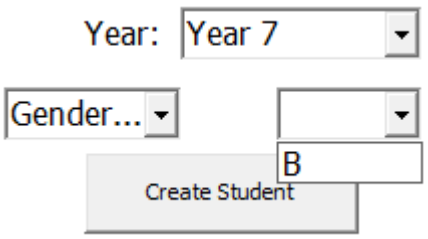
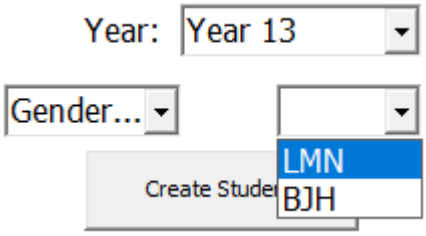
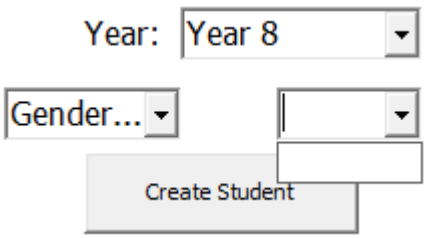
The Create Student form has been implemented. Criteria 7 has been met. It allows the user to create a student and store the student in a specific class, hence requirement 10 has been met. The Create Student Form can also be open from the main menu, hence requirement 8.3 has been partially met. The prototype was tested in runtime of the code. Screenshots of results of the tests are shown below.

Summary: The user is able to create and sign in using an account. The main menu form has also been created, which will allow users to open different forms for the solution. They are also able to create classes. They can now also create a class and place the student inside an existing class.

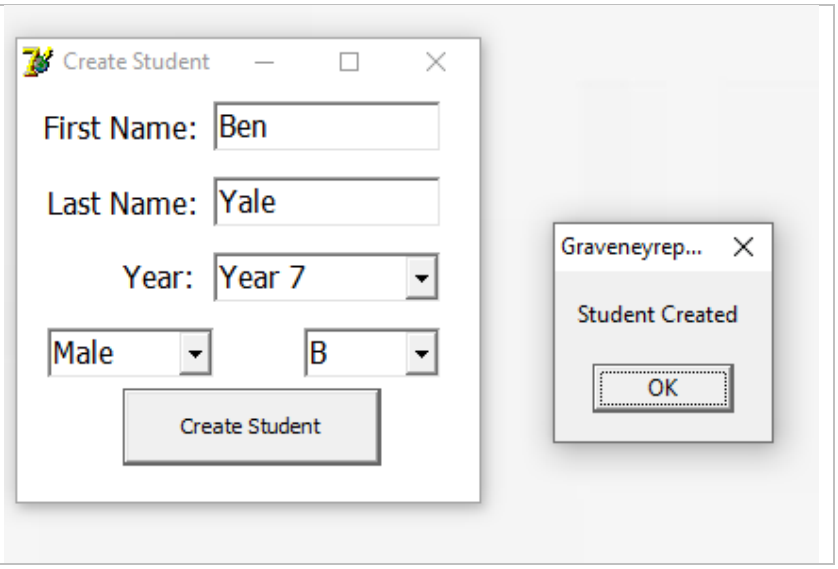
Testing of solution

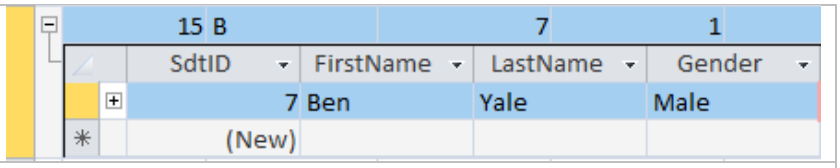
Test carried out are 5.1, 5.2, 5.3, 5.4, 5.5.

Test No 5.1 result: checking if users can select class, when creating a student…

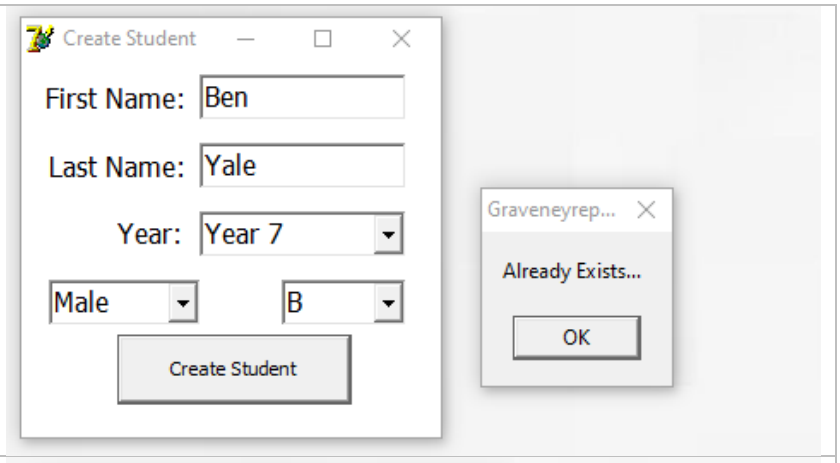
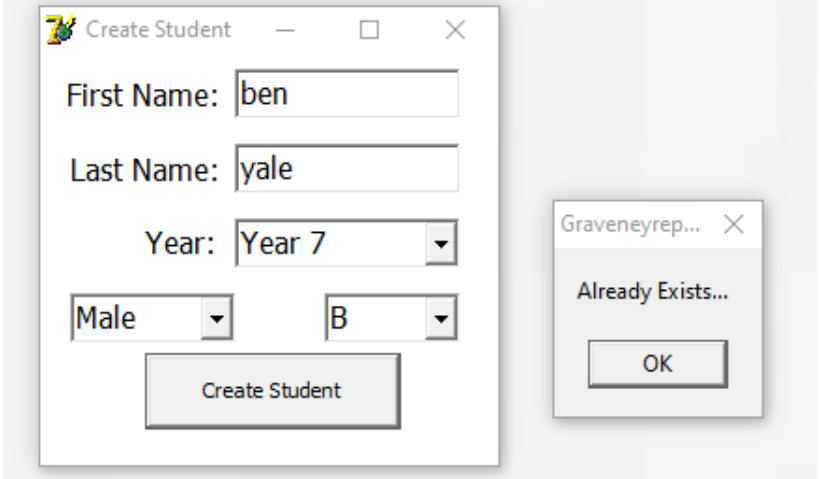
Existing Classes under user at the moment	
Step 1) Select year 7	
Step 2) Select Year 13	
Step 3) Select Year 8	
Summary: Test Successful – The Create Student form displays all classes corresponding to the year selected.	

Test No 5.2 result: checking if users can create a student…

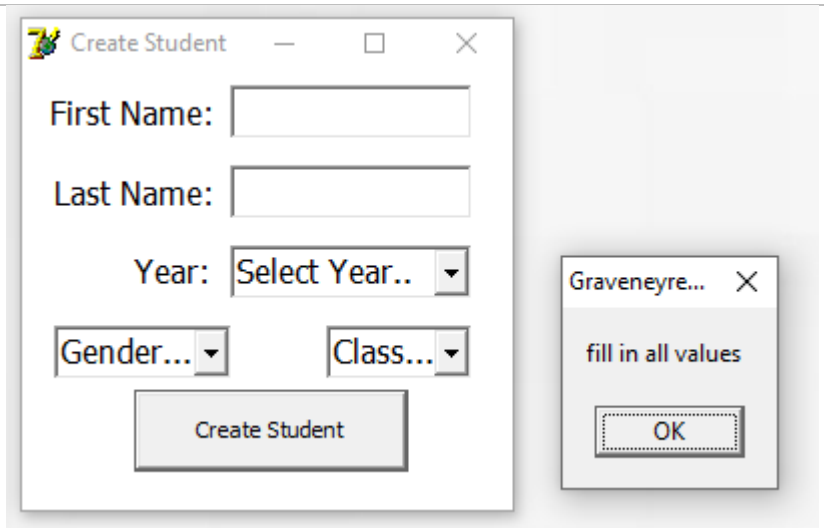
Entering the Values and Clicking the create student button	

	
Summary: Test Successful – The Create Student form is able to create a student, and inform the user.	

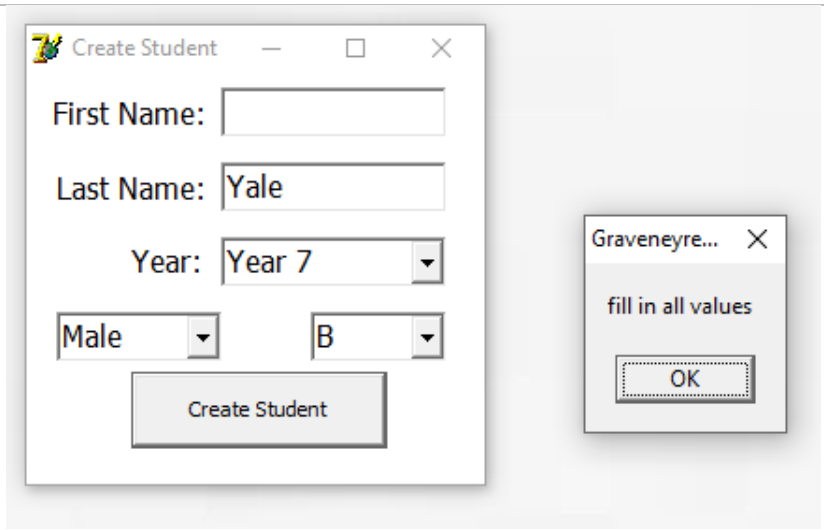
Test No 5.3 result: checking if user can create student, when student exists…

Step 1) Entering the values and pressing button	
Step 2) Entering the values and pressing button	
Summary: Test Successful – The Create Class form is able to check if student already exists. It is able to inform user the student exists. The Create Class form is not case sensitive. It is able to verify that the class exists in all possible ways of case used.	

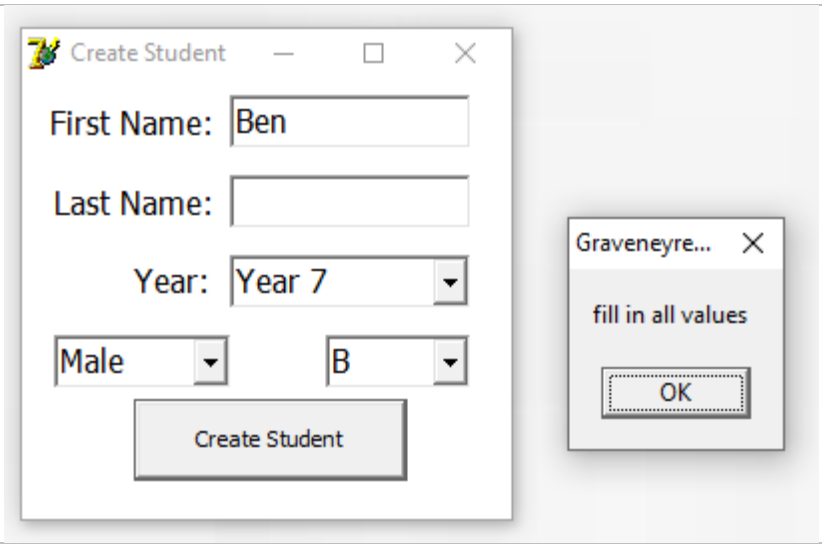
Test No 5.4.1 result: checking if form is completed…

Click on Create Student Button	
Summary: Test Successful – The Create Student form does not create an account with null values and informs user to fill in the values.	

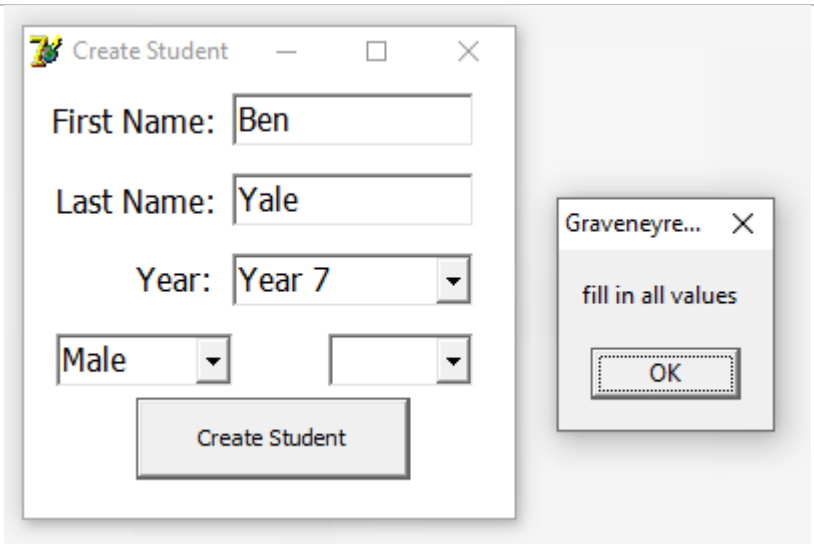
Test No 5.4.2 result: checking if form is completed, when first name is null…

Enter the values and Click on Create Student Button	
Summary: Test Successful – The Create Student form does not create an account with first name as null and informs user to fill in the values.	

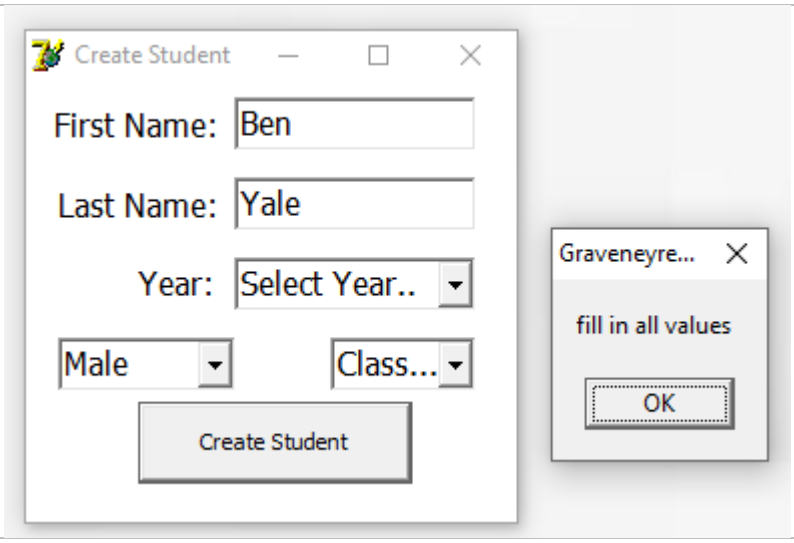
Test No 5.4.3 result: checking if form is completed, when last name is null…

Enter the values and Click on Create Student Button	
Summary: Test Successful – The Create Student form does not create an account with last name as null and informs user to fill in the values.	

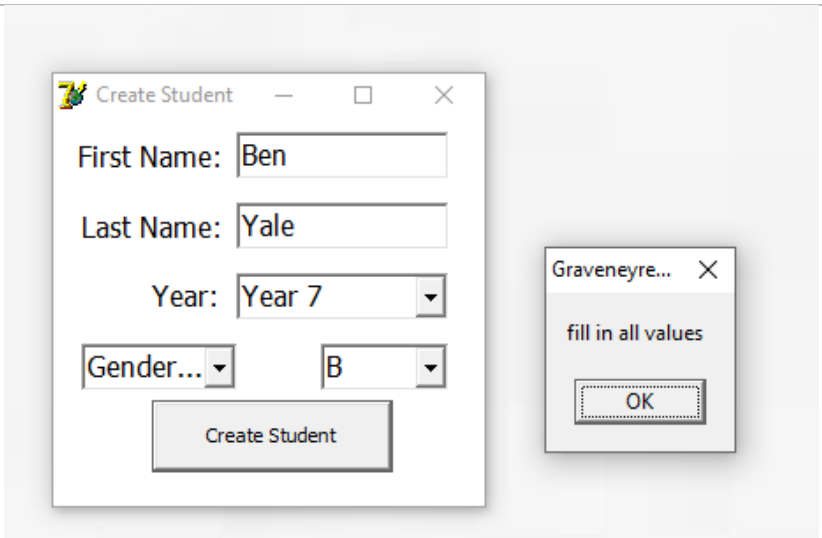
Test No 5.4.4 result: checking if form is completed, when class is not selected…

Enter the values and Click on Create Student Button	
Summary: Test Successful – The Create Student form does not create an account when no class is selected and informs user to fill in the values.	

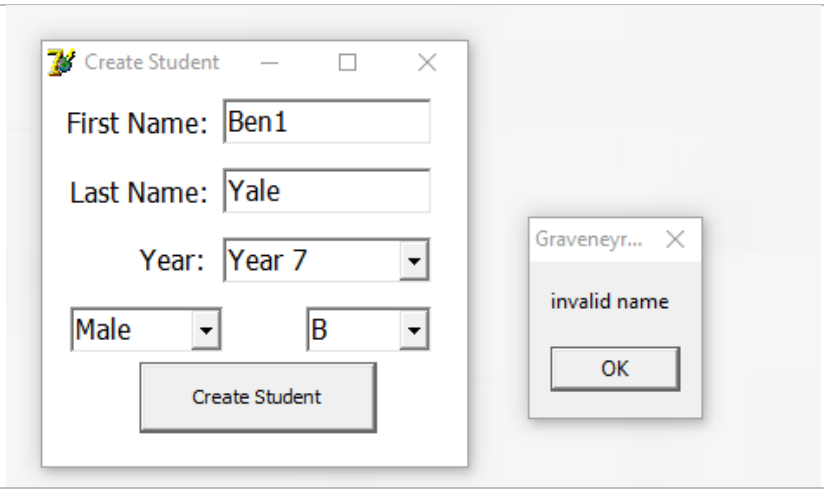
Test No 5.4.5 result: checking if form is completed, when year is not selected…

Enter the values and Click on Create Student Button	
Summary: Test Successful – The Create Student form does not create an account when no year is selected and informs user to fill in the values.	

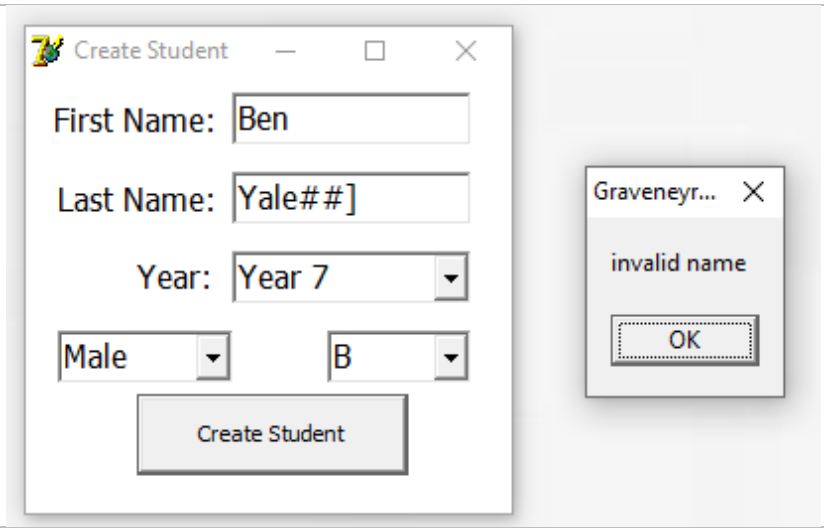
Test No 5.4.6 result: checking if form is completed, when gender is not selected…

Enter the values and Click on Create Student Button	
Summary: Test Successful – The Create Student form does not create an account when no gender is selected and informs user to fill in the values.	

Test No 5.5.1 result: making sure system does not create student when student name is invalid…

Enter the values and Click on Create Student Button	
Summary: Test Successful – The Create Student form does not create an account, when name is invalid, and informs the user.	

Test No 5.5.2 result: making sure system does not create student when student name is invalid…

Enter the values and Click on Create Student Button	
Summary: Test Successful – The Create Student form does not create an account, when name is invalid, and informs the user.	

Prototype 2

Stakeholder, Mr Patel, has tested out the Form and wishes to make a change. The change that he wishes to make is only showing the years that the user has existed classes for, instead of all years 7 to 13. This is to make it easier for user to filter out.

Changes in Implemented Code for "Main Menu Form" (unit_menu)

Coded Section Before	Coded Section after

`procedure TfrmMainMenu.btnCreateStudentClick(Sender: TObject);`
`begin`
`  //open form`
`  frmCreateStudent.ShowModal;`
`end;`
	`procedure TfrmMainMenu.btnCreateStudentClick(Sender: TObject);`
	`var i:Integer;`
	`begin`
	`  with frmCreateStudent do`
	`  begin`
	`  cboSelectYear.Clear;`

```
    qryCreateStudent.SQL.Text:='select
year from class where StfID =
'+inttostr(info.staffID)+' group by year
order by year;';
    qryCreateStudent.Open;
  end;
  for i:=1 to
frmcreatestudent.qrycreatestudent.recordco
unt do
  begin

frmCreateStudent.cboSelectYear.items.Add('
Year'
+frmCreateStudent.qryCreateStudent.fieldby
name
('year').AsString);

frmCreateStudent.qryCreateStudent.Next;
  end;
  frmCreateStudent.qryCreateStudent.Close;
  //open form
  frmCreateStudent.ShowModal;
end;
```

Justification: Everytime Create Student form is open, it must check for what years to add to the combo box.
This cannot be done when the form is created, since the Form is only created once and a new class can be created when system is running and the combo box will not be able to see the year of it. The Create Class form cannot be created everytime the user opens it as it will require more main memory.

Changes in Implemented Code for "Create Student Form" (unit_createStudent)

```
…
//Works out the year selected by user
function getClassYear(year:string):string;
begin
  year := year[Length(year)];
  if (StrToInt(year)<7)then
    year:=IntToStr(StrToInt(year)+10);
  getClassYear:=year;
end;

procedure TfrmCreateStudent.cboSelectYearChange(Sender: TObject);
var i:Integer;
begin
  qrycreatestudent.SQL.Text := 'Select name from class where
year='+getClassYear(cboSelectYear.Text)+' and stfID='+inttostr(info.staffID)+';';
  cboSelectClass.Clear;
  qryCreateStudent.Open;
  for i:=1 to qrycreatestudent.recordcount do
  begin
    cboSelectClass.Items.Add(qryCreateStudent.fieldbyname('name').AsString);
    qryCreateStudent.Next;
  end;
  qryCreateStudent.Close;
end;

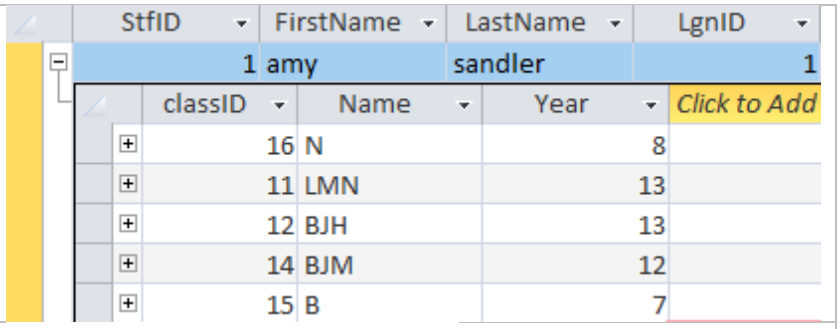
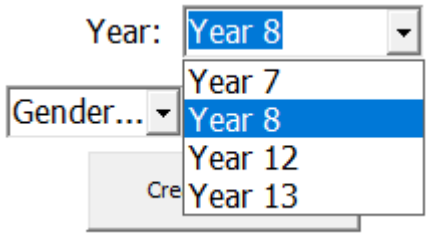
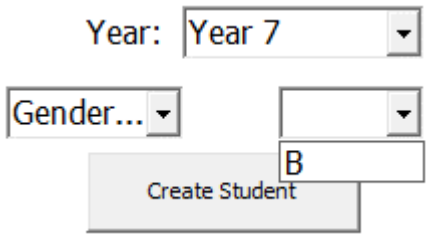
procedure writeStudent;
var classID:String;
begin
  //Getting ClassID of the selected class
```

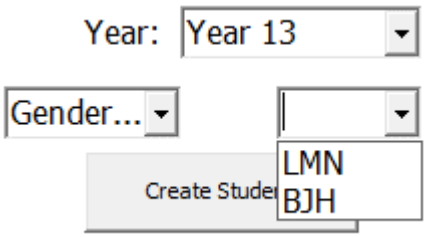
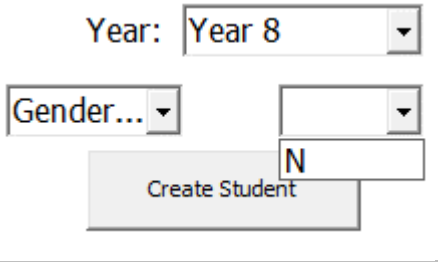
```
  frmCreateStudent.qrycreatestudent.SQL.Text := 'Select classID from class where
year='+getClassYear(frmCreateStudent.cboSelectYear.Text)+' and
stfID='+inttostr(info.staffID)+' and
name='+quotedStr(frmCreateStudent.cboselectclass.Text)+';';
  …
```

Justification: Function getClassYear is coded to work out the year of the class, since the old method will not work anymore. Old code relied of years to be written from year 7 to 13, so it can increment the index by 7 to find the year. The function is executed in both, writeStudent and cboSelectYearChange procedures.

Testing of solution

Retest No 5.1 result: checking if users can select class, when creating a student…

Existing Classes under user at the moment	
Only Shows the years that the user has classes for	
Step 1) Select year 7	

Step 2) Select Year 13	
Step 3) Select Year 8	
Summary: Test Successful – The Create Student form displays all classes corresponding to the year selected.	

Prototype 3

Whilst using the system, an error was found with the event of cboyear being changed. The system prompts an error whenever an alphabetical letter was entered within the cbo. This was not user friendly. The error was caused since, the year is searched within the database, and the field year is an integer.

Code Changes

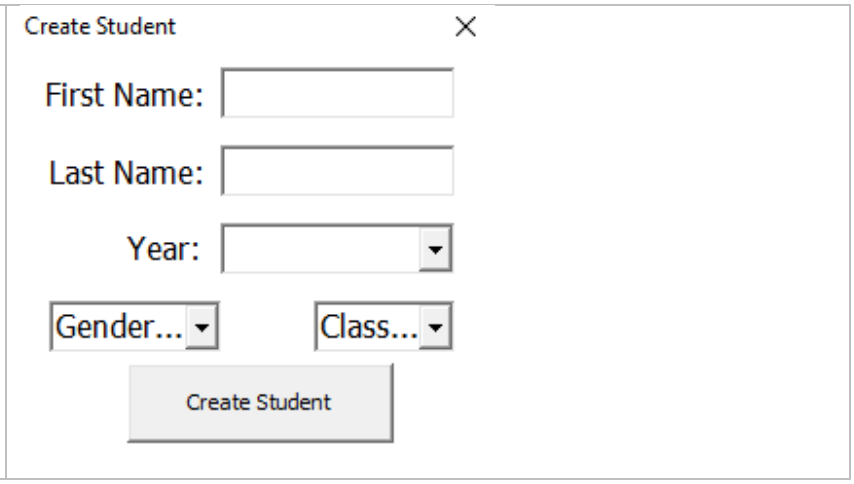
```
procedure TfrmCreateStudent.selectClasses(cboClass, cboYear: TObject);
var i:Integer;
begin
  qryCreateStudent.SQL.Text := 'Select name from class where
year='+getClassYear(TComboBox(cboYear).text)+' and
stfID='+inttostr(info.staffID)+';';
  TComboBox(cboclass).Clear;
  qryCreateStudent.Open;
  for i:=1 to qryCreateStudent.recordcount do
  begin
    TComboBox(cboclass).Items.Add(qryCreateStudent.fieldbyname('name').AsString);
    qryCreateStudent.Next;
  end;
  qryCreateStudent.Close;
end;

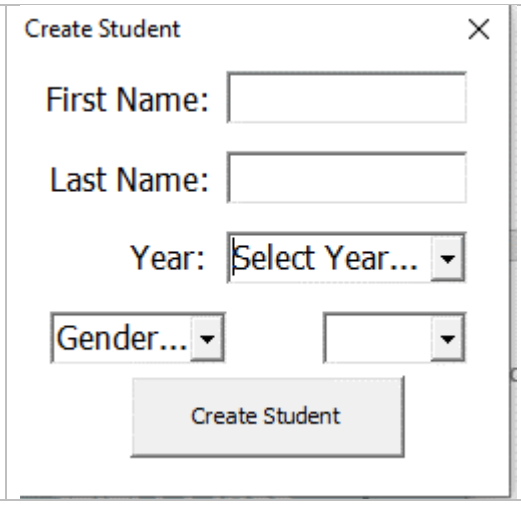
procedure TfrmCreateStudent.cboSelectYearChange(Sender: TObject);
var i:Integer;
begin
  try
    selectClasses(cboSelectClass,cboselectyear);
  except
    cboselectyear.Text:='Select Year...';
    cboselectclass.Clear;
  end;
end;
```

Testing of solution

Test No 5.6 result: checking result of adding an alphabetical character to cboselectyear …

Before



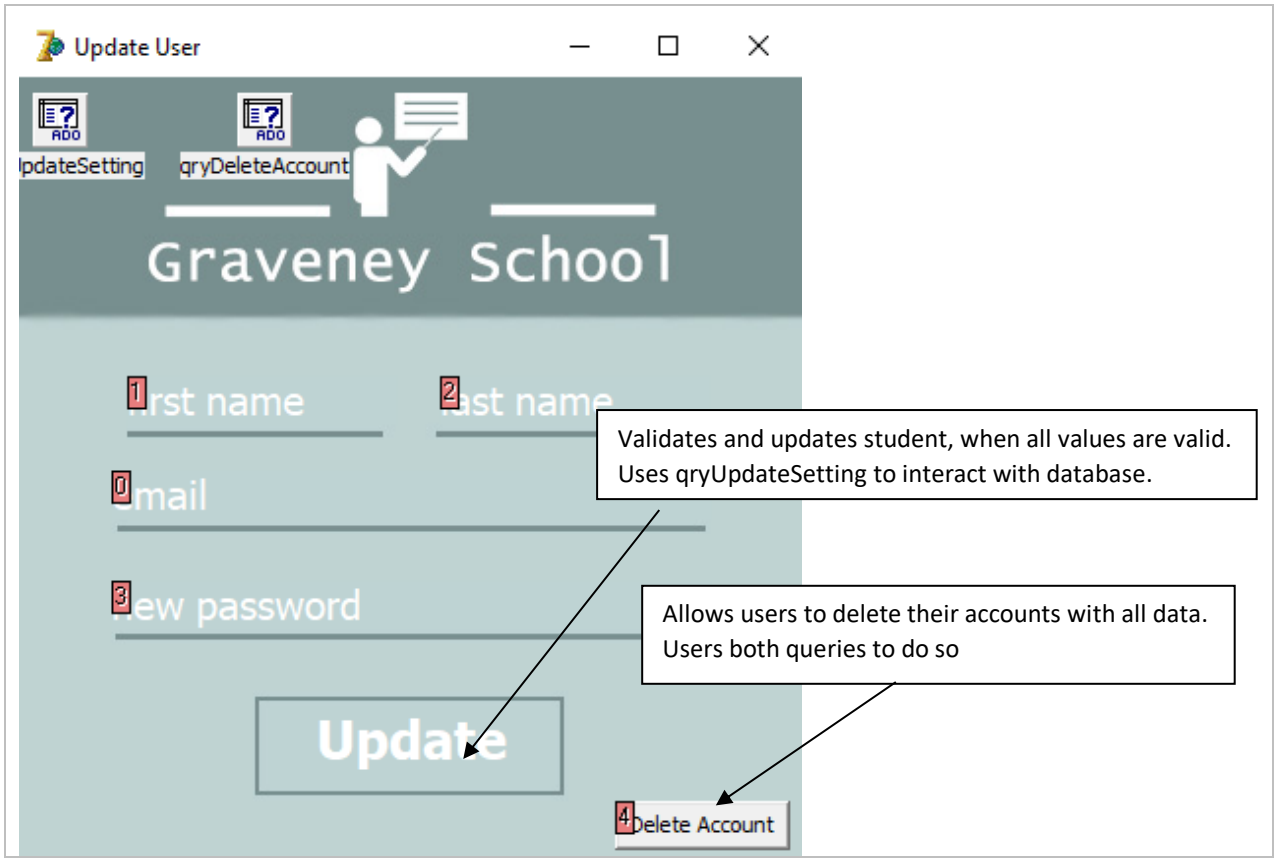
Adding the value		
Summary: Test Successful – The system will not report any errors to the user and will just change the value to select year.		

Stage 6: Implementing the "Update User Form"

The update setting form allows users to update their personal details, such as email, name and password. It also allows them to delete their accounts along with all data.

Prototype 1

Screenshot of "Update User Form"



Changes in Implemented Code for "Main Menu Form" (unit_menu)

Coded Section Before	Coded Section After

`procedure TfrmMainMenu.imgSettingsClick(Sender: TObject);`
`begin`
`  //open form`
`end;`
	`procedure TfrmMainMenu.imgSettingsClick(Sender: TObject);`
	`begin`
	`  //open form`
	`  with frmUpdateSetting do`
	`  begin`
	`  edtfName.Text:= info.FirstName;`
	`  edtlName.Text:= info.LastName;`
	`  edtEmail.Text:= info.Email;`
	`  ShowModal;`
	`  end;`
	`end;`

`uses unit_login, unit_createClass, unit_createStudent, ;`	`uses unit_login, unit_createClass, unit_createStudent, Unit_UpdateSetting;`
Justification: Allow user to open the Update User form. It also changes the values of the name and email to the users current details, to show them what their details are at the moment.	

Implemented Code for Update User Form (unit_UpdateSetting)

```
unit Unit_UpdateSetting;

interface

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

type
  TfrmUpdateSetting = class(TForm)
  published
    logo_header: TImage;
    edtEmail: TEdit;
    edtfName: TEdit;
    edtlName: TEdit;
    edtpassword: TEdit;
    btnUpdate: TLabel;
    qryUpdateSetting: TADOQuery;
    btnDeleteAccount: TButton;
    qryDeleteAccount: TADOQuery;
    procedure btnUpdateMouseEnter(Sender: TObject);
    procedure btnUpdateMouseLeave(Sender: TObject);
    procedure edtpasswordClick(Sender: TObject);
    procedure edtEmailorNameClick(Sender: TObject);
    procedure btnUpdateClick(Sender: TObject);
    procedure btnDeleteAccountClick(Sender: TObject);
  end;

var
  frmUpdateSetting: TfrmUpdateSetting;

implementation

{$R *.dfm}

//changes the colour of the button; indicating to the user that is clickable
procedure TfrmUpdateSetting.btnUpdateMouseEnter(Sender: TObject);
begin
  btnUpdate.Font.Color:= RGB(119,143,143);
end;

procedure TfrmUpdateSetting.btnUpdateMouseLeave(Sender: TObject);
begin
  btnUpdate.Font.Color:= clWhite;
end;

//removes and adds the default value of edit password when user clicks on a
component, makes it easier for the user to use the component
procedure TfrmUpdateSetting.edtpasswordClick(Sender: TObject);
begin
  if (edtpassword.Text='new password')then
    edtpassword.Text := '';
end;

procedure TfrmUpdateSetting.edtEmailorNameClick(Sender: TObject);
begin
  if (edtpassword.Text='')then
    edtpassword.Text := 'new password';
end;
```

```
//gets the users login ID corresponding to its Staff ID from the database
function getLgnIDusingStfID(staffID:string):string;
var loginID:string;
begin
  with frmUpdateSetting.qryUpdateSetting do
  begin
    SQL.Text:='select lgnID from staff where
stfID='+inttostr(info.staffID)+';';
    Open;
    LoginID:=FieldByName('LgnID').AsString;
    Close;
  end;
  //One to one relationship, therefore only one Login ID per Staff ID
  getLgnIDusingStfID:=loginID;
end;

//Updates Staff Details when the button is pressed
procedure TfrmUpdateSetting.btnUpdateClick(Sender: TObject);
var loginID:string;
begin
  //Checks if the form is filled out
  if not((edtfName.Text = '') or (edtlName.Text = '') or (edtpassword.Text
='') or (edtEmail.Text = '') or (edtpassword.Text ='new password'))then
  begin
    //verifies password and name
    if (frmSignup.verifyName_and_Password(edtfName.Text, edtlName.Text,
edtpassword.Text))then
    begin
      //Verifies Email
      if (frmSignup.IsValidEmail(edtEmail.Text))then
      begin
        LoginID:=getLgnIDusingStfID(IntToStr(info.staffID));
        qryUpdateSetting.SQL.Text:='select lgnID from login where
email='+quotedStr(edtemail.Text)+';';
        qryUpdateSetting.Open;
        //Checks if there is another account with same email
        if not((qryUpdateSetting.RecordCount<>0) and
(qryUpdateSetting.FieldByName('lgnID').AsString<>loginID)) then
        begin
          qryupdatesetting.SQL.Text:='update Staff set
FirstName='+quotedStr(edtfname.Text)+', LastName='+quotedStr(edtlname.Text)+'
where StfID='+inttostr(info.staffID)+';';
          qryUpdateSetting.ExecSQL;
          qryupdatesetting.SQL.Text:='update Login set
Email='+quotedStr(edtemail.Text)+', password='+quotedStr(edtpassword.Text)+'
where LgnID='+loginID+';';
          qryUpdateSetting.ExecSQL;
          showmessage('Account Updated');
        end
        else
          ShowMessage('Account with email address already exists...');
      end
      else
        ShowMessage('invalid email address');
    end;
  end
  else
    ShowMessage('fill in the form...');
end;

//deletes all reports created by staff
```

```
procedure deleteAllReports(staffID:string);
var reportCount, i:integer;
begin
  with frmUpdateSetting.qryUpdateSetting do
  begin
    SQL.Text := 'SELECT Report.RptID FROM Staff INNER JOIN (Class INNER JOIN
(Student INNER JOIN Report ON Student.SdtID = Report.SdtID) ON Class.classID =
Student.classID) ON Staff.StfID = Class.StfID WHERE Staff.StfID =
'+staffID+';';
    Open;
    reportCount := recordcount;
  end;
  for i:= 1 to reportCount do
  begin
    with frmUpdateSetting do
    begin
      qryDeleteAccount.SQL.text:='DELETE * FROM [report comments] where
rptID='+qryupdatesetting.fieldbyname('RptID').asstring+';';
      qryDeleteAccount.ExecSQL;
      qryUpdateSetting.Next;
    end;
  end;
  frmUpdateSetting.qryUpdateSetting.close;

  with frmUpdateSetting.qryUpdateSetting do
  begin
    SQL.Text := 'SELECT Student.SdtID FROM (Staff INNER JOIN Class ON
Staff.StfID = Class.StfID) INNER JOIN Student ON Class.classID =
Student.classID  WHERE Staff.StfID = '+staffID+';';
    Open;
    reportCount := recordcount;
  end;
  for i:= 1 to reportCount do
  begin
    with frmUpdateSetting do
    begin
      qryDeleteAccount.SQL.text:='DELETE * FROM report where
SdtID='+qryupdatesetting.fieldbyname('SdtID').asstring+';';
      qryDeleteAccount.ExecSQL;
      qryUpdateSetting.Next;
    end;
  end;
  frmUpdateSetting.qryUpdateSetting.close;
end;

//Deletes all students created by staff
procedure deleteAllStudents(staffID:string);
var classCount,i:Integer;
begin
  with frmUpdateSetting.qryUpdateSetting do
  begin
    SQL.Text := 'select classID FROM class where stfID='+staffID+';';
    Open;
    classCount := recordcount;
  end;
  for i:= 1 to classCount do
  begin
    with frmUpdateSetting do
    begin
      qryDeleteAccount.SQL.text:='DELETE * FROM student where
classID='+qryupdatesetting.fieldbyname('classID').asstring+';';
```

```
      qryDeleteAccount.ExecSQL;
      qryUpdateSetting.Next;
    end;
  end;
  frmUpdateSetting.qryUpdateSetting.close;
end;

//Deletes all classes created by staff
procedure deleteAllClasses(staffID:string);
begin
  with frmUpdateSetting.qryDeleteAccount do
  begin
    SQL.text:='DELETE * FROM class where stfID='+inttostr(info.StaffID)+';';
    ExecSQL;
  end;
end;

//Deletes Staffs Login and Staff Record
procedure deleteAccountDetails(staffID:string);
var loginID:string;
begin
  loginID:=getLgnIDusingStfID(staffID);
  with frmUpdateSetting.qryDeleteAccount do
  begin
    SQL.Text:='DELETE * FROM staff where stfID='+inttostr(info.StaffID)+';';
    ExecSQL;
    SQL.Text:='DELETE * FROM login where lgnID='+loginID+';';
    ExecSQL;
  end;
  //Closes the system as user is logged onto a deleted account
  Showmessage('deleted, restart system...');
  frmLogin.Close;
end;

//runs all procedures to delete the account fully when the user clicks delete
account
procedure TfrmUpdateSetting.btnDeleteAccountClick(Sender: TObject);
var i,count:Integer;
  loginID:string;
begin
  //Confirms with user if they want to delete their account, as data cannot be
recovered
  case MessageDlg('Confirm Deleting Account with all data...', mtConfirmation,
[mbYes, mbNo], 0) of
    mrYes:
    begin
      deleteAllReports(IntToStr(info.StaffID));
      deleteAllStudents(IntToStr(info.StaffID));
      deleteAllClasses(IntToStr(info.StaffID));
      deleteAccountDetails(IntToStr(info.StaffID));
    end;
    mrNo:
    begin
      ShowMessage('cancelled');
    end;
  end;
end;
end.
```

Validation carried out for "Update User Form" (unit_UpdateSetting)

Validation of email	An email has to consist of only one '@' symbol; separates username with mail server name. An email must have at least one full stop. The full stop is valid to be used within the username. It also is used to separate the mail server with domain. The domain is the end of the email address, therefore if there is a full stop at the end of an email address the user enters, it is not a valid email address. An email address only consists of'_', '-', '.', any digit(s), any Alphabetical character and a '@'.
Validation of password	A password must be greater than or equal to 8 characters. It must have at least one digit.
Validation of name	Name must only be Alphabetical characters.
Validation of Account	If an account with same email exists, the account is not update.
Validates if form is empty	If the values are null or the value of password is new password means the form isn't filled out.

What has been done

The update setting form has been implemented. Criteria 3 has been met. It allows the user to update and delete their account, hence requirement 9 has been met. The update setting form can also be open from the main menu, hence requirement 8.3 has been partially met. The prototype was tested in runtime of the code. Screenshots of results of the tests are shown below.

Summary: The user is able to create and sign in using an account. The main menu form has also been created, which will allow users to open different forms for the solution. They are also able to create classes and students within it. They can now able to update their account details and delete their account along with all their data.

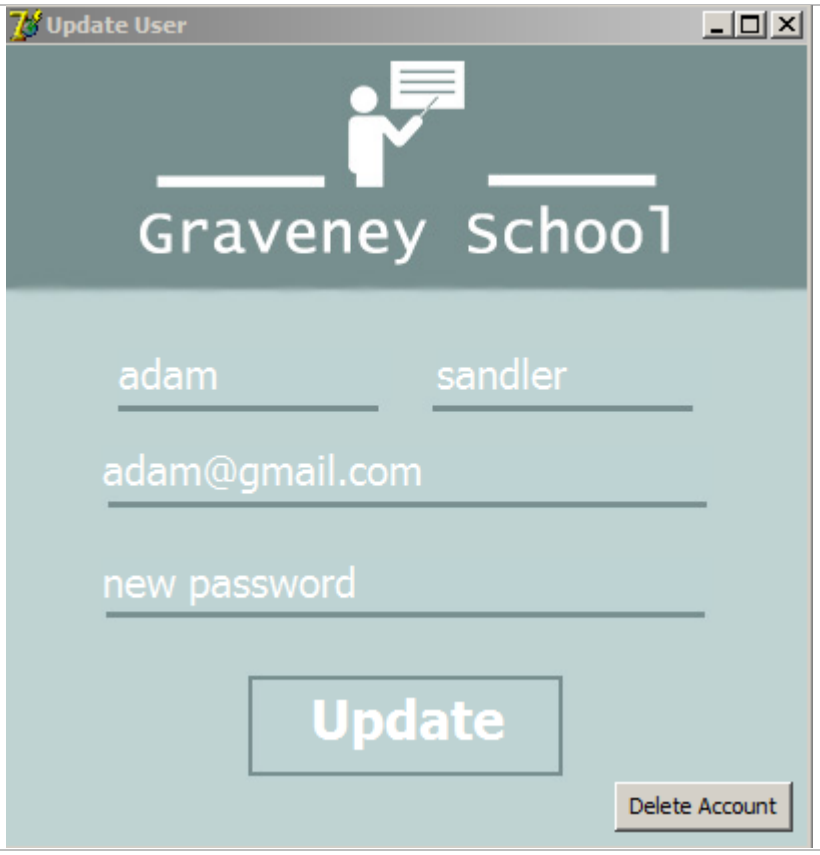
The update setting form is able to let users edit their details. It also allows the user to delete its account.

Criteria 3 has been completed.

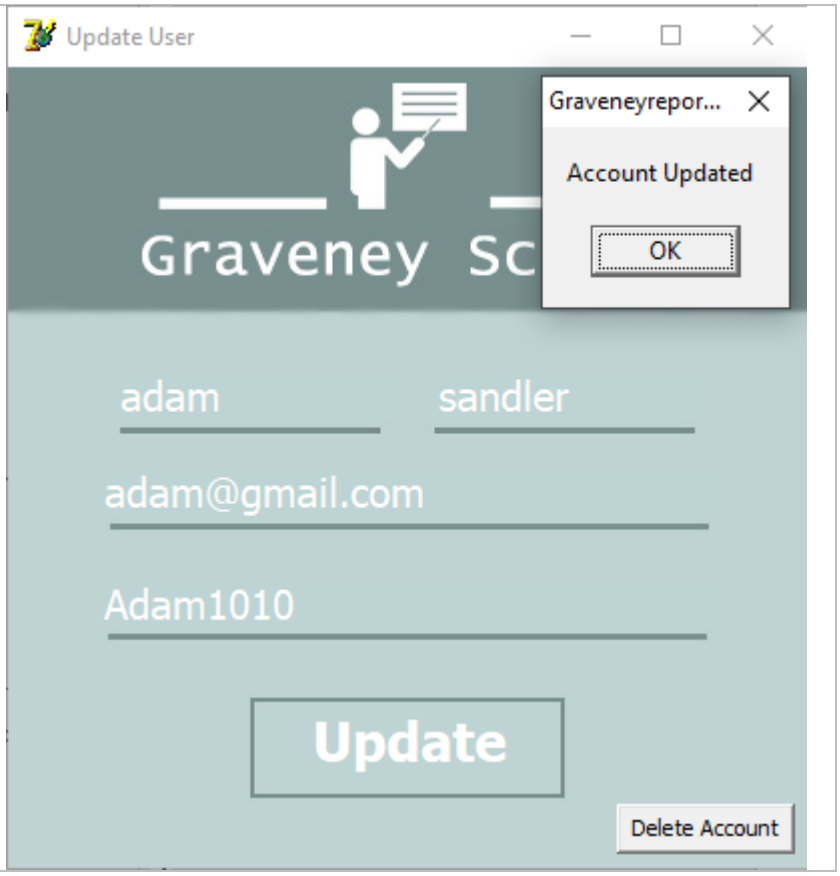
Testing of solution

Test carried out are 6.1, 6.2, 6.3, 6.4, 6.5, 6.6

Test No 6.1 result: checking if main menu can open the form…

Clicking on setting button	
Summary: Test Successful – Main menu form is able to display the Update Setting Form. It is able to write the users details in its corresponding edit boxes.	

Test No 6.2 result: checking if system can update account details, when it is the same…

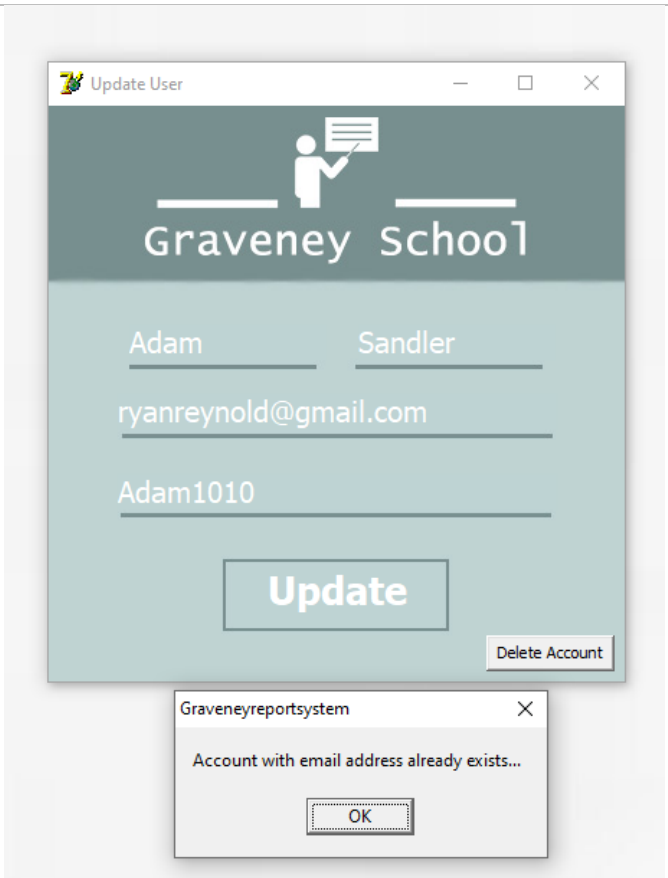
Typing password and clicking on setting button	

Summary: Test Successful – The System is will allows users to update their account with the same values. This is needed to as the solution, the system should be able to only change one value and have the others the same. Since the system allows to update it to the same values, it will allow users to update their account in all possible ways with some values being the same.

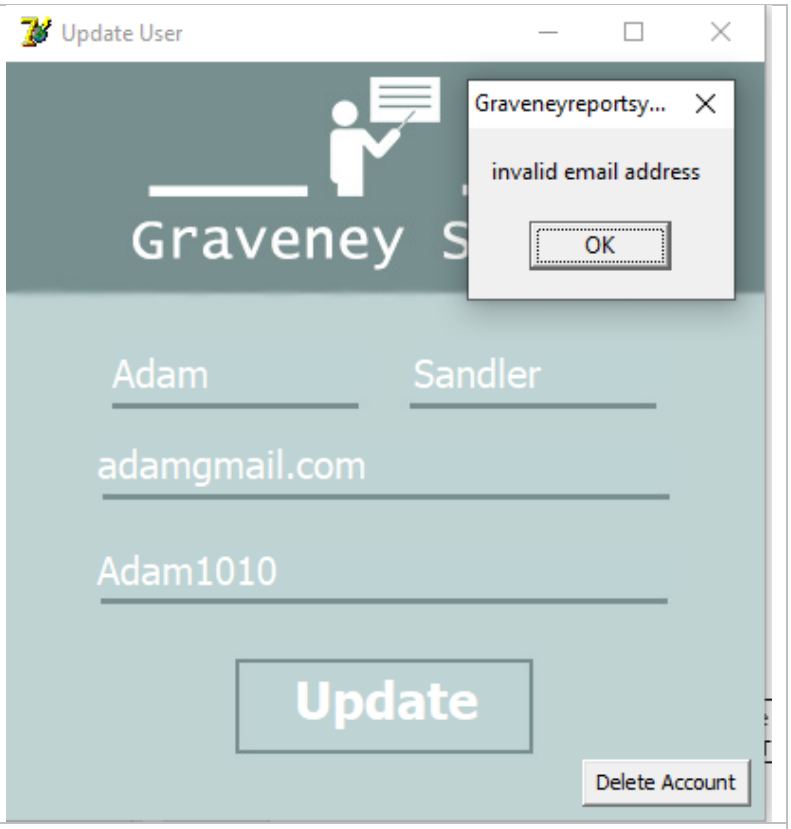
Test No 6.3.1 result: checking if system can update account email…

Typing values Email was Adam@gmail.com Before…	
Summary: Test Successful – The System is able to update an account with valid values. It changed the email within the database. Users is informed when the account was updated.	

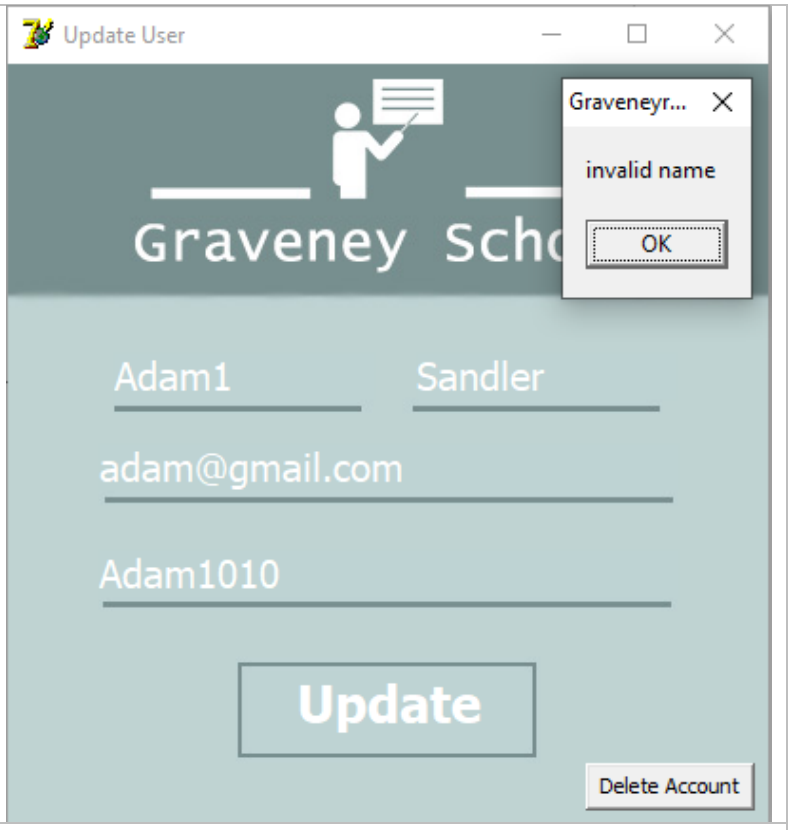
Test No 6.3.2 result: checking if system will update account email, when the new email entered is registered to another account…

Typing values	
Summary: Test Successful – The System was able to read the database and find out that the account already exists to another user. The system did not update the email and informed the user why.	

Test No 6.3.3 result: checking if system will update account email, when the new email entered is invalid…

Typing values	
Summary: Test Successful – The Update Setting Form is able to use signup forms validation for email, hence all email validation is a success, since it's already been tested.	

Test No 6.4.1 result: checking if system will update account name, when the new first name entered is invalid…

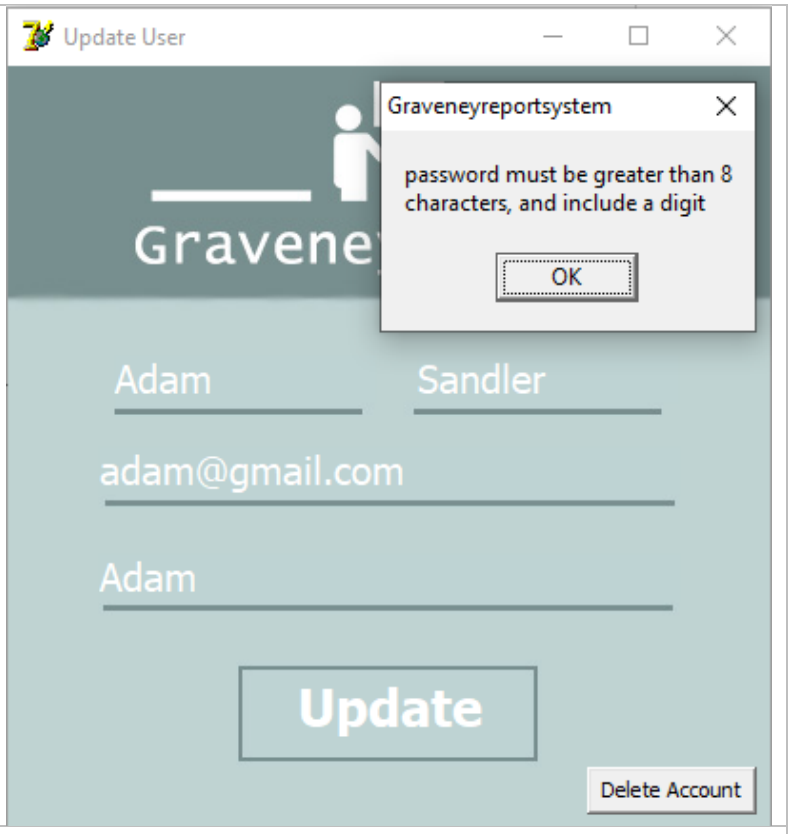
Typing values	
Summary: Test Successful – The Update Setting Form is able to use signup forms validation for name, hence all name validation is a success, since it's already been tested.	

Test No 6.4.2 result: checking if system will update account name, when the new last name entered is invalid…

Typing values	
Summary: Test Successful – The Update Setting Form is able to use signup forms validation for name, hence all name validation is a success, since it's already been tested.	

Test No 6.5 result: checking if system will update account name, when the new password entered is invalid…

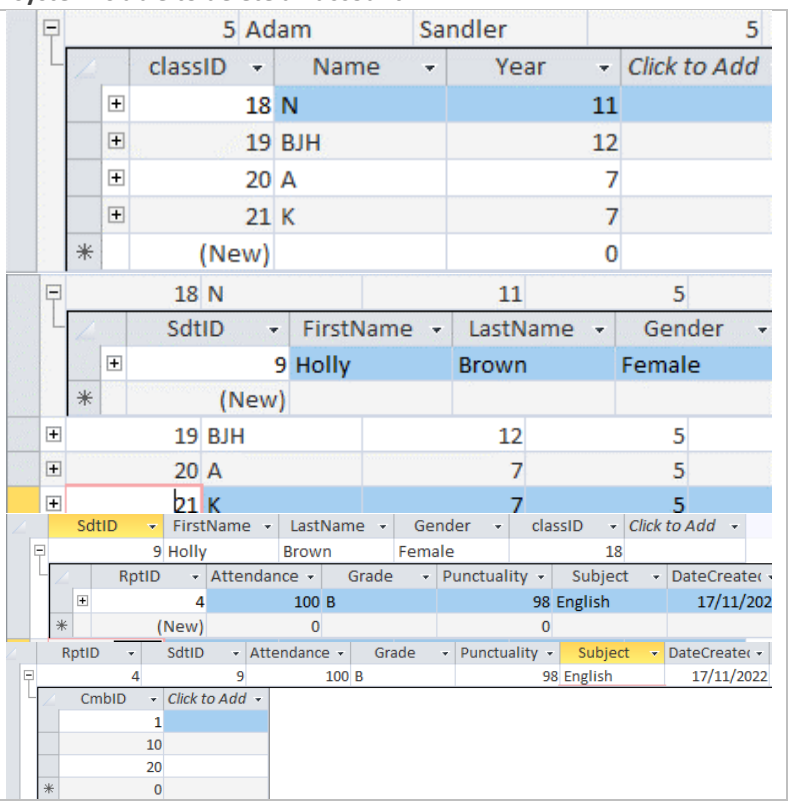
Typing values

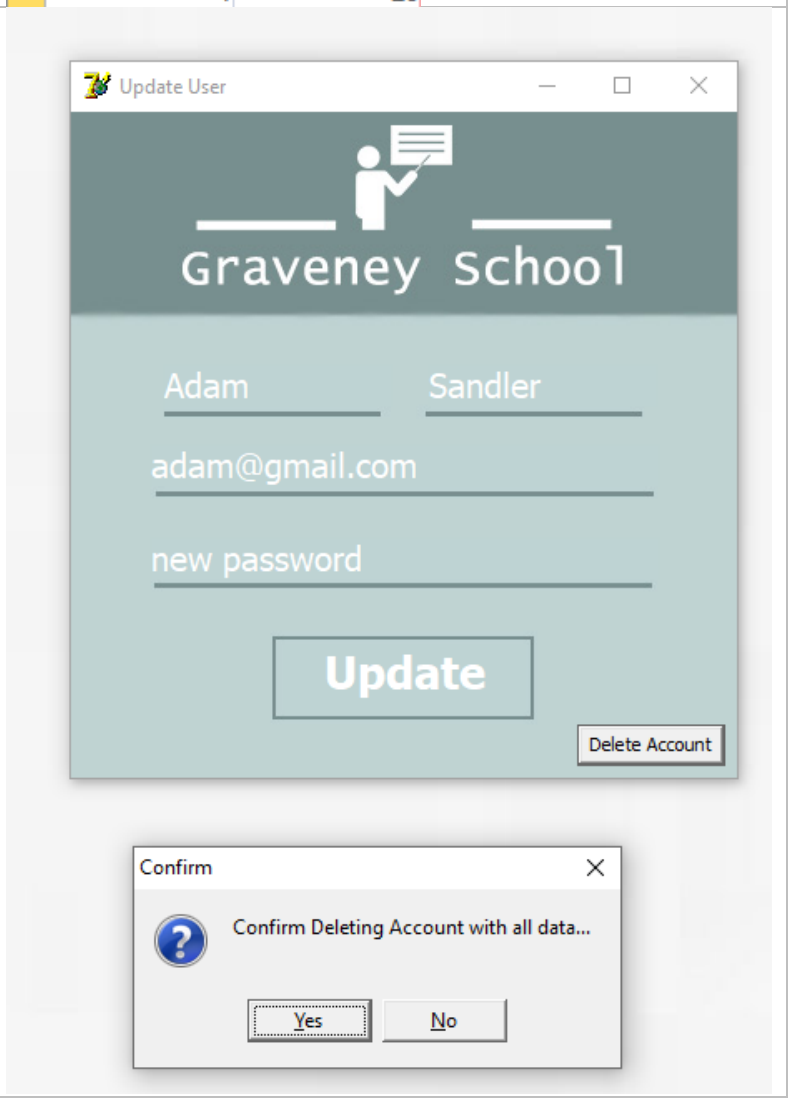


Summary: Test Successful – The Update Setting Form is able to use signup forms validation for password, hence all password validation is a success, since it's already been tested.

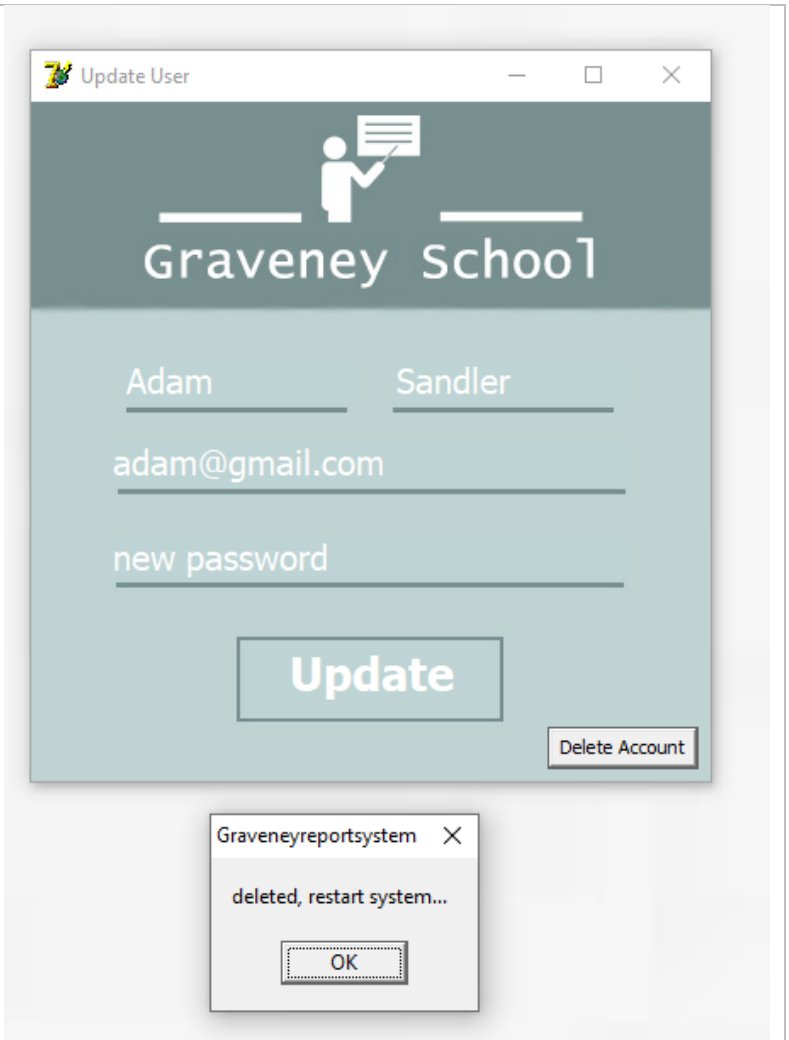
Test No 6.6.1 result: checking if system is able to delete an account…

Users' data within database before

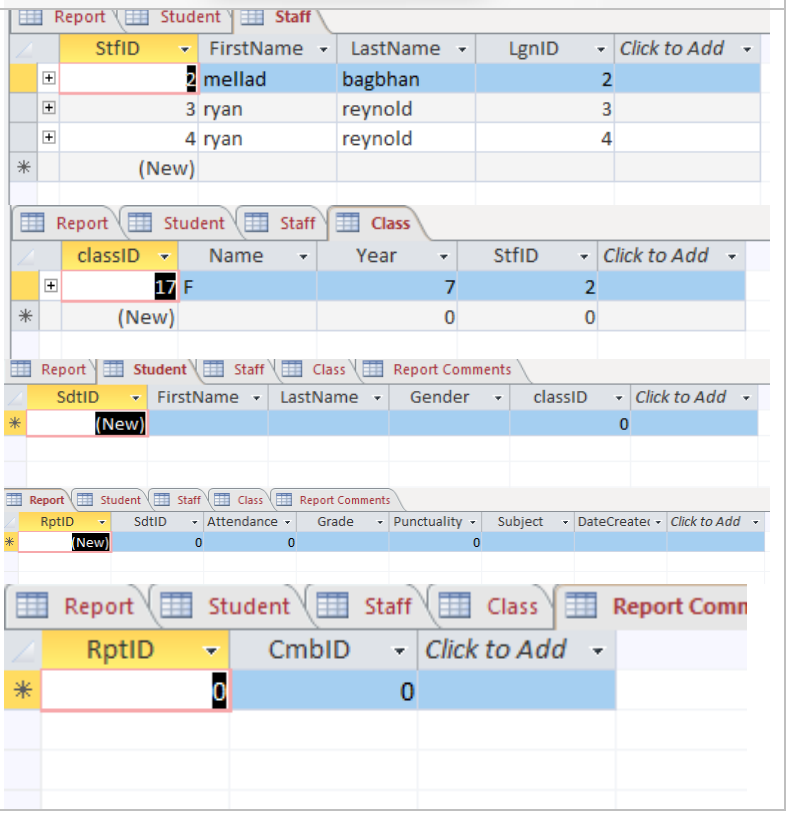


	RptID / CmbID 4 / 1 4 / 10 4 / 20
1) Pressing on the delete button	

2) Pressing yes then okay

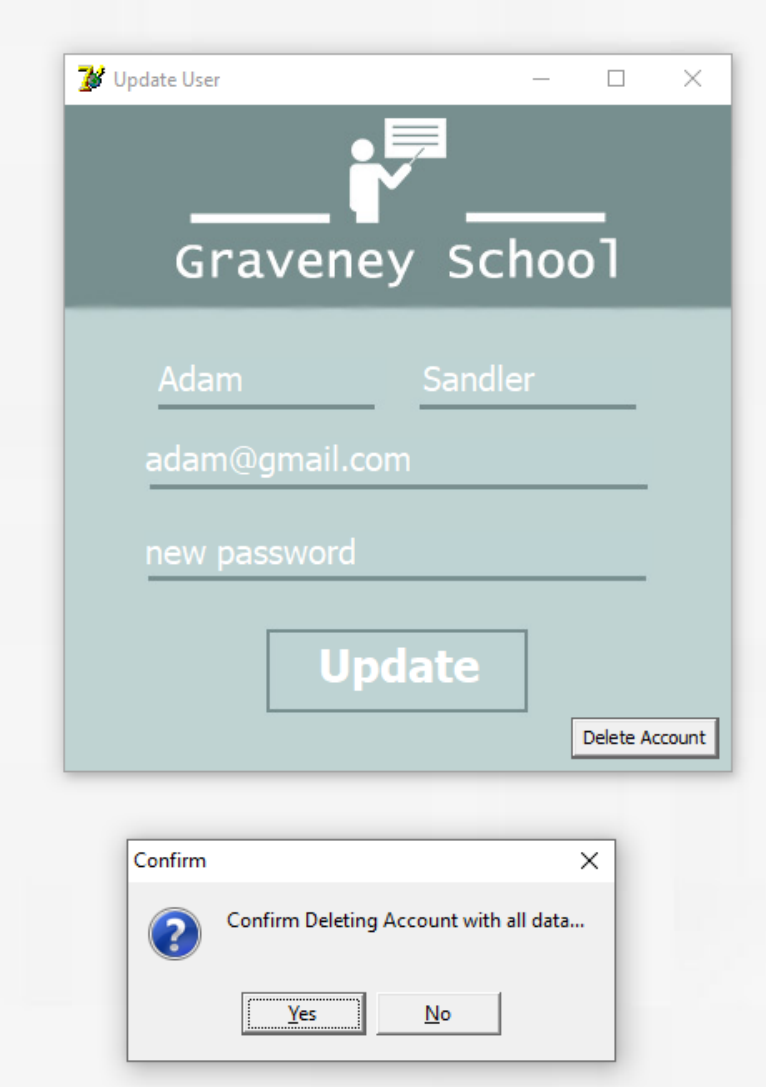


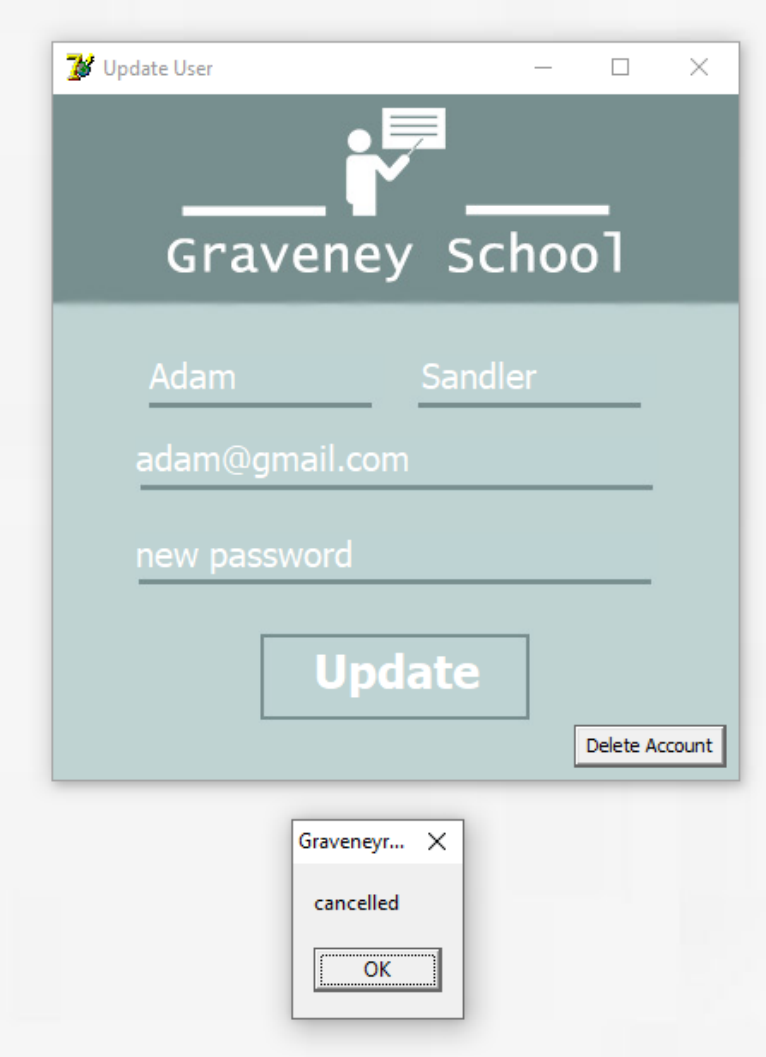
Database after



Summary: Test Successful – The Update Setting Form is able to delete all of user's data within the database, hence deleting the account. The system warns user before deleting the account.

Test No 6.6.2 result: checking if system will delete account, when user presses no...

Pressing delete button	

Pressing No	
Summary: Test Successful – The Update Setting Form is able to prevent the user from accidentally deleting their account. It also informs user when it does not delete it.	

Prototype 2

When showing the system to the stakeholders, they were not happy that the system stores users and students names without the correct use of cases.

Changes in Implemented Code for "SignUp Form" (unit_Signup)

Coded Section Before	Coded Section After
`if ( writeaccount (edtfName.Text, edtlName.Text, edtEmail.Text, edtpassword.Text) )then`	`if ( writeaccount (FirstLetterUppercase (edtfName.Text), FirstLetterUppercase (edtlName.Text), edtEmail.Text, edtpassword.Text) )then`
Justification: Now when the name is stored in the database, it stores it in the correct case	

```
  Public
    function FirstLetterUppercase(temp:string):string;
    …
…
function TfrmSignup.FirstLetterUppercase(temp:string):string;
begin
  temp := lowercase(temp);
  temp[1] := upcase(temp[1]);
  FirstLetterUppercase := temp;
end;
…
```

Justification: the string is first converted to lowercase. Then the first character of the string is converted to uppercase.

Changes in Implemented Code for "Create Student Form" (unit_CreateStudent)

Coded Section Before

```
frmCreateStudent.qryCreateStudent.SQL.Text :=
'insert into student(FirstName, LastName,Gender, ClassID) values('+
quotedStr(frmCreateStudent.lbledtStudent_FirstName.Text)
+','+quotedStr(frmCreateStudent.lbledtStudent_LastName.Text)+
','+quotedStr(frmCreateStudent.cboSelectGender.text)+','+classID+');';
```

Coded Section After

```
frmCreateStudent.qryCreateStudent.SQL.Text :=
'insert into student(FirstName, LastName,Gender, ClassID) values('+
quotedStr(frmSignup.FirstLetterUppercase(frmCreateStudent.lbledtStudent_FirstN
ame.Text))
+','+quotedStr(frmSignup.FirstLetterUppercase
(frmCreateStudent.lbledtStudent_LastName.Text))+
','+quotedStr(frmCreateStudent.cboSelectGender.text)+','+classID+');';
```

Justification: Now when the student name is stored in the database, it stores it in the correct case

Changes in Implemented Code for "Update User Form" (unit_UpdateSetting)

Coded Section Before

```
qryupdatesetting.SQL.Text:='update Staff set
FirstName='+quotedStr(edtfname.Text)+', LastName='+quotedStr(edtlname.Text)+'
where StfID='+inttostr(info.staffID)+';';
```

Coded Section After
`qryupdatesetting.SQL.Text:='update Staff set FirstName='+quotedStr(frmSignup.FirstLetterUppercase(edtfname.Text))+', LastName='+quotedStr(frmSignup.FirstLetterUppercase(edtlname.Text))+' where StfID='+inttostr(info.staffID)+';';`
Justification: Now when the name is updated in the database, it stores it in the correct case

What has been done

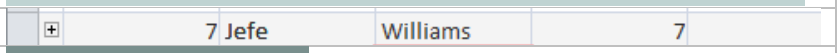
Algorithm has been created to prevent data being stored in the database with incorrect casing. The stakeholder required this to be done so the system looks for professional.

Summary: The user is able to create and sign in using an account. The user is able to update their details. When doing so, their name is checked by the system, making sure the casing is correct. If not, it is corrected. The main menu form has also been created, which will allow users to open different forms for the solution. The main menu form always displays the user's name in correct casing. The user is also able to create classes and students within it. The students name case is also corrected.

Testing of solution

Test carried out are 6.7

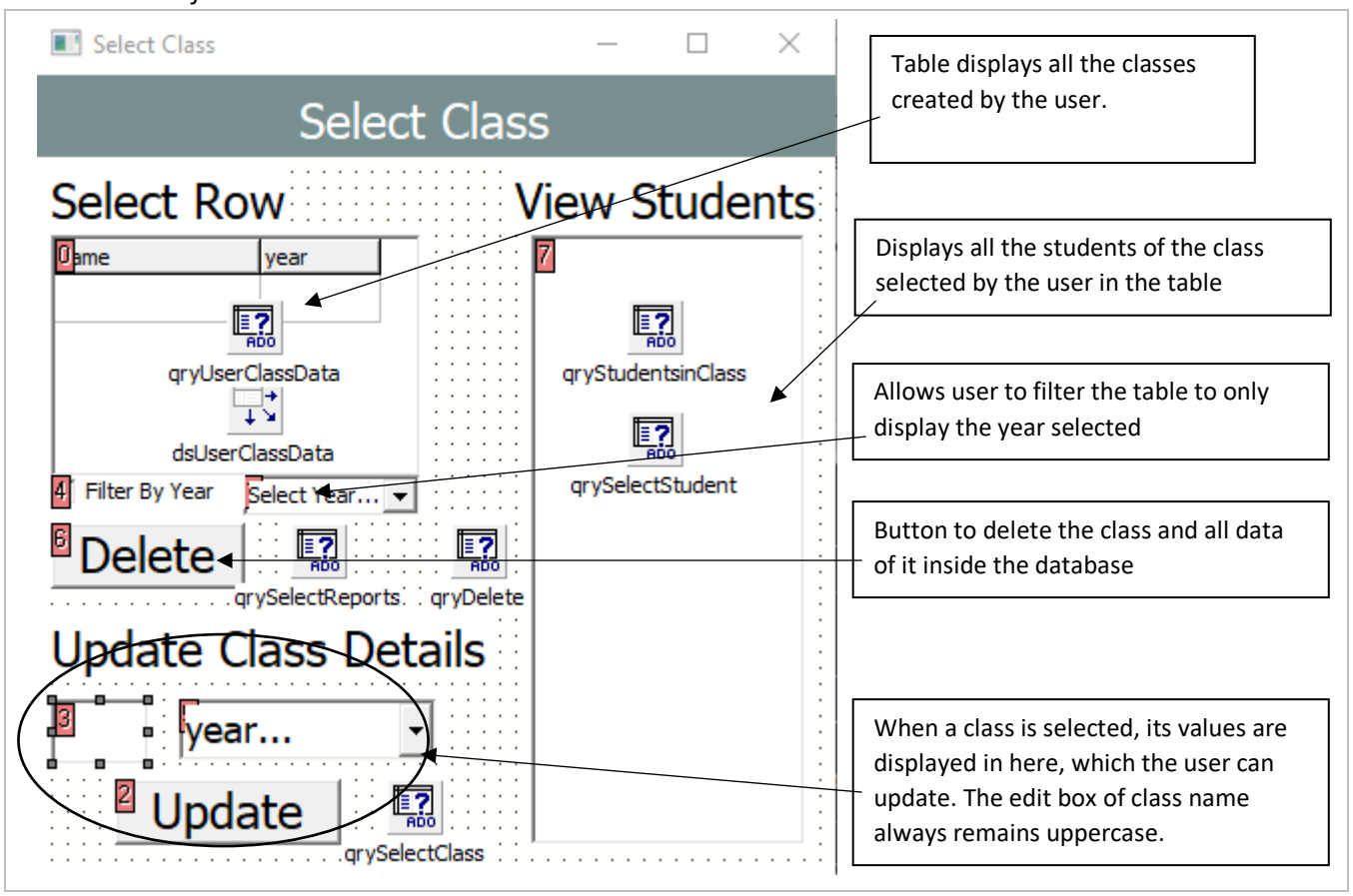
Test No 6.7 result: checking if algorithm works and corrects casing…

Typing in the values	
Stored as in database…	
Logged in main menu	
Summary: Test Successful – Algorithm works and makes sure the correct casing is used within the database.	

Stage 7: Implementing the "Select Class Form"

Prototype 1

Screenshot of "Select Class Form"



Changes in Implemented Code for "Main Menu Form" (unit_menu)

Coded Section Before	Coded Section After

`procedure TfrmMainMenu.btnSelectClassClick(Sender: TObject);`
`begin`
`  //open form`
`end;`
	`procedure TfrmMainMenu.btnSelectClassClick(Sender: TObject);`
	`begin`
	`  //open form`
	`frmSelectClass.qryUserClassData.SQL.Text := 'select ClassID, name, year from class where stfID='+inttostr(info.StaffID)+';';`
	`  frmSelectClass.qryUserClassData.Open;`
	`  frmSelectClass.ShowModal;`
	`end;`

`uses unit_login, unit_createClass, unit_createStudent, Unit_UpdateSetting;`	`uses unit_login, unit_createClass, unit_createStudent, Unit_UpdateSetting, unit_selectClass;`
Justification: Allow user to open the Select Class Form from main menu.	

Changes in Implemented Code for "Create Class Form" (unit_createClass)

Coded Section Before	Coded Section After

procedure TfrmMainMenu.btnSelectClassClick(Sender: TObject);
begin
  //open form
end;
	public
	  function ClassExists(formclassName:string; year:Integer):boolean; **//used in select class**
	  function verifyClassName(formclassName:string; year:Integer):boolean; **//used in select class**

function ClassExists(formclassName: string; year:Integer) :boolean;	function TfrmCreateClass.ClassExists(formclassName: string; year:Integer):boolean;
function TfrmCreateClass.verifyClassName (formclassName:string; year:Integer):boolean;	function TfrmCreateClass.verifyClassName (formclassName:string; year:Integer):boolean;
Justification: Allows the Select Class unit to use the functions from unit Create Class	

Implemented Code for Select Class Form (unit_SelectClass)

```
unit Unit_SelectClass;

interface

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

type
  TfrmSelectClass = class(TForm)
    imgImgBackground: TImage;
    lblSelectStudent: TLabel;
  published
    dbgrdUserClassData: TDBGrid;
    dsUserClassData: TDataSource;
    qryUserClassData: TADOQuery;
    cboSelectYear: TComboBox;
    btnUpdate: TButton;
    qrySelectClass: TADOQuery;
    edtName: TEdit;
    chkFilterYear: TCheckBox;
    cboUpdateYear: TComboBox;
    btnDelete: TButton;
    lstStudents: TListBox;
    qryStudentsinClass: TADOQuery;
    lblUpdateColumn: TLabel;
    lblSelectRow: TLabel;
    lblViewStudents: TLabel;
    qryDelete: TADOQuery;
    qrySelectStudent: TADOQuery;
    qrySelectReports: TADOQuery;
    procedure cboSelectYearChange(Sender: TObject);
    procedure btnUpdateClick(Sender: TObject);
    procedure dbgrdUserClassDataCellClick(Column: TColumn);
    procedure chkFilterYearClick(Sender: TObject);
    procedure btnDeleteClick(Sender: TObject);
  end;

var
```

```
  frmSelectClass: TfrmSelectClass;
  classIDSearched: string;
implementation

{$R *.dfm}

//toggle filtering feature for the table displaying all classes created by
user
procedure TfrmSelectClass.cboSelectYearChange(Sender: TObject);
begin
  if (chkFilterYear.Checked)then
  begin
    qryUserClassData.SQL.Text := 'select ClassID, name, year from class where
stfID='+inttostr(info.StaffID)+'and year='+inttostr(cboSelectYear.ItemIndex +
7)+';';
    qryUserClassData.Open;
  end;
end;

//updates the class
procedure TfrmSelectClass.btnUpdateClick(Sender: TObject);
var year:Integer;
begin
  if not(classIDSearched='')then
  begin
    year:=cboUpdateYear.ItemIndex+7;

    //Class Name validated according to its length and year
    if (frmcreateclass.verifyClassName(edtName.Text, year))then
    begin
      //checks if class exists or not
      if not(frmCreateClass.ClassExists(edtName.Text, year))then
      begin
        //updates the class
        qrySelectClass.SQL.Text:='update Class set name = ' +
quotedStr(edtname.Text) + ', year='+inttostr(year)+' where
classID='+classIDSearched+';';
        qrySelectClass.ExecSQL;
        qryUserClassData.Refresh;
        ShowMessage('class updated');
      end
      else
        //informs user class already exists
        ShowMessage('already exists...');
    end
    else
      ShowMessage('Invalid class name'); //invalid when doesnt follow graveney
class rules
  end
  else
    ShowMessage('cannot edit non-existing class');
end;

procedure TfrmSelectClass.dbgrdUserClassDataCellClick(Column: TColumn);
var yearValue:string;
  i:Integer;
begin
  lstStudents.Clear;
  //stores and writes the values of the class selected in the table to the
form
  classIDSearched:=vartostr(qryUserClassData['classID']);
```

```
  edtName.text:=vartostr(qryUserClassData['name']);
  yearValue:=vartostr(qryUserClassData['year']);
  if (yearValue='')then
    cboUpdateYear.ItemIndex:=-1
  else
    cboUpdateYear.ItemIndex:=strtoint(yearValue)-7;
  if (classIDSearched<>'')then
  begin
    //displays the students within the selected class
    qryStudentsinClass.SQL.Text := 'select FirstName+''+QuotedStr('
')+''+LastName as name from student where classID='+classIDSearched+';';
    qryStudentsinClass.Open;
    if (qryStudentsinClass.RecordCount>0)then //if 0, class has 0 students
    begin
      for i:=1 to qryStudentsinClass.RecordCount do
      begin

lstStudents.Items.Add(qryStudentsinClass.fieldbyname('name').AsString);
        qrystudentsinclass.Next;
      end;
    end;
  end;

end;

//allows user to filter the classes in table by year
procedure TfrmSelectClass.chkFilterYearClick(Sender: TObject);
begin
  if((chkFilterYear.Checked)and(cboSelectYear.ItemIndex >-1))then
    qryUserClassData.SQL.Text := 'select ClassID, name, year from class where
stfID='+inttostr(info.StaffID)+'and year='+inttostr(cboSelectYear.ItemIndex +
7)+';'
  else
    qryUserClassData.SQL.Text := 'select ClassID, name, year from class where
stfID='+inttostr(info.StaffID)+';';
  qryUserClassData.Open;
end;

//deletes students reports
procedure DeleteStudentReports(studentID:string);
var i,recCount:integer;
begin
  with frmSelectClass do
  begin
    qrySelectReports.SQL.text:='Select RptID FROM report where
SdtID='+StudentID+';';
    qrySelectReports.open;
    recCount:=qrySelectReports.recordCount;
    for i:=1 to recCount do
    begin
      qryDelete.SQL.text:='DELETE * FROM [report Comments] where
RptID='+qrySelectReports.fieldbyname('RptID').asstring+';';
      qryDelete.ExecSQL;
      qrySelectReports.Next;
    end;
    qryDelete.SQL.text:='DELETE * FROM report where SdtID='+StudentID+';';
    qryDelete.ExecSQL;
  end;
end;

//takes class id by parameters and deletes the class and the data
```

```
procedure DeleteClass(ClassID:string);
var i:integer;
begin
  with frmSelectClass do
  begin
    qrySelectClass.SQL.text:='Select SdtID FROM student  where
classID='+classID+';';
    qrySelectClass.open;
    for i:=1 to qrySelectClass.RecordCount do
    begin
      DeleteStudentReports(qrySelectClass.fieldbyname('SdtID').asstring);
      qrySelectclass.Next;
    end;
    qrySelectClass.SQL.text:='DELETE * FROM student where
classID='+classID+';';
    qrySelectClass.ExecSQL;
    qrySelectClass.SQL.text:='DELETE * FROM Class where classID='+classID+';';
    qrySelectClass.ExecSQL;
    qryUserClassData.Close;
    qryUserClassData.Open;
    lstStudents.Clear;
  end;
end;

//deletes all students and their data within the selected class, when button
clicked
procedure TfrmSelectClass.btnDeleteClick(Sender: TObject);
begin
  case MessageDlg('Confirm Deleting Class
'+vartostr(qryUserClassData['name'])+' of year
'+vartostr(qryUserClassData['year'])+' with
'+inttostr(lststudents.Items.Count)+' student(s)', mtConfirmation, [mbYes,
mbNo], 0) of
  mrYes:
    begin
      DeleteClass(classIDSearched);
      Showmessage('deleted');
    end;
  mrNo:
    begin
      ShowMessage('cancelled');
    end;
  end;
end;

end.
```

Validation carried out for "Select Class Form" (unit_SelectClass)

Validation of Class Name	The Class name must be valid to create a class. Classes of year 7 to 11 must be one character in length, whilst year 12 and 13's must be 3 characters in length. This is the format of Graveney School's class name.
Validation of Class Existence	The Class will only be created if it does not already exist.
Validates if form is empty	If the value of the name is null or if the user did not select a year for the class, the system displays a message to the user informing them to fill in the missing values

What has been done

The Select class form has been implemented. Criteria 5 has been met. It allows the user to update and delete classes, hence requirement 13.1 and 13.2 have been met. The select class form can also be open from the main

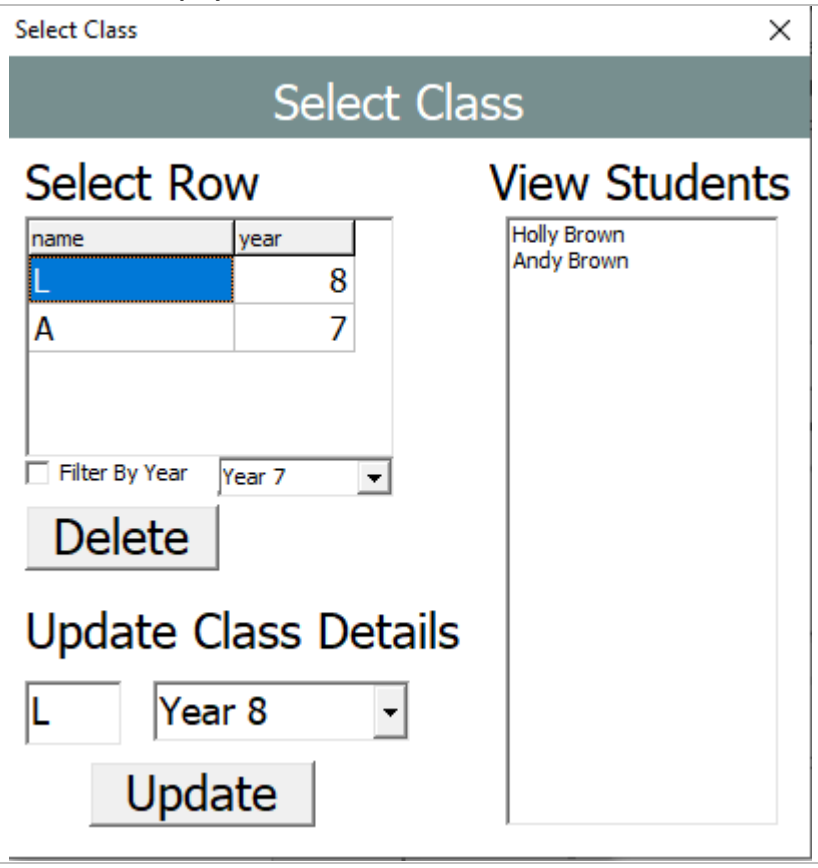
menu, hence requirement 8.3 has been partially met. The prototype was tested in runtime of the code. Screenshots of results of the tests are shown below.

Summary: The user is able to create and sign in using an account. The user is able to update their details. The main menu form has also been created, which will allow users to open different forms for the solution. They are also able to create classes and students within it. They can now also view all students within a class, delete the class along with the students and their data inside the class, and update classes.

Testing of solution

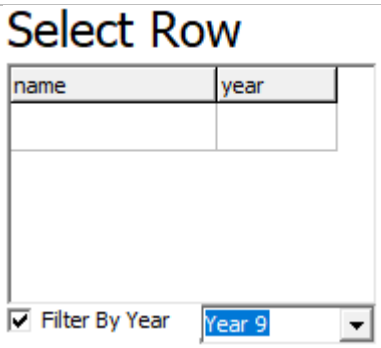
Test carried out are 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7, 7.8

Test No 7.1 result: Checking if the form can display students within the selected class…

Selecting the class	
Summary: Test Successful – The form is able to display all the students within the selected class	

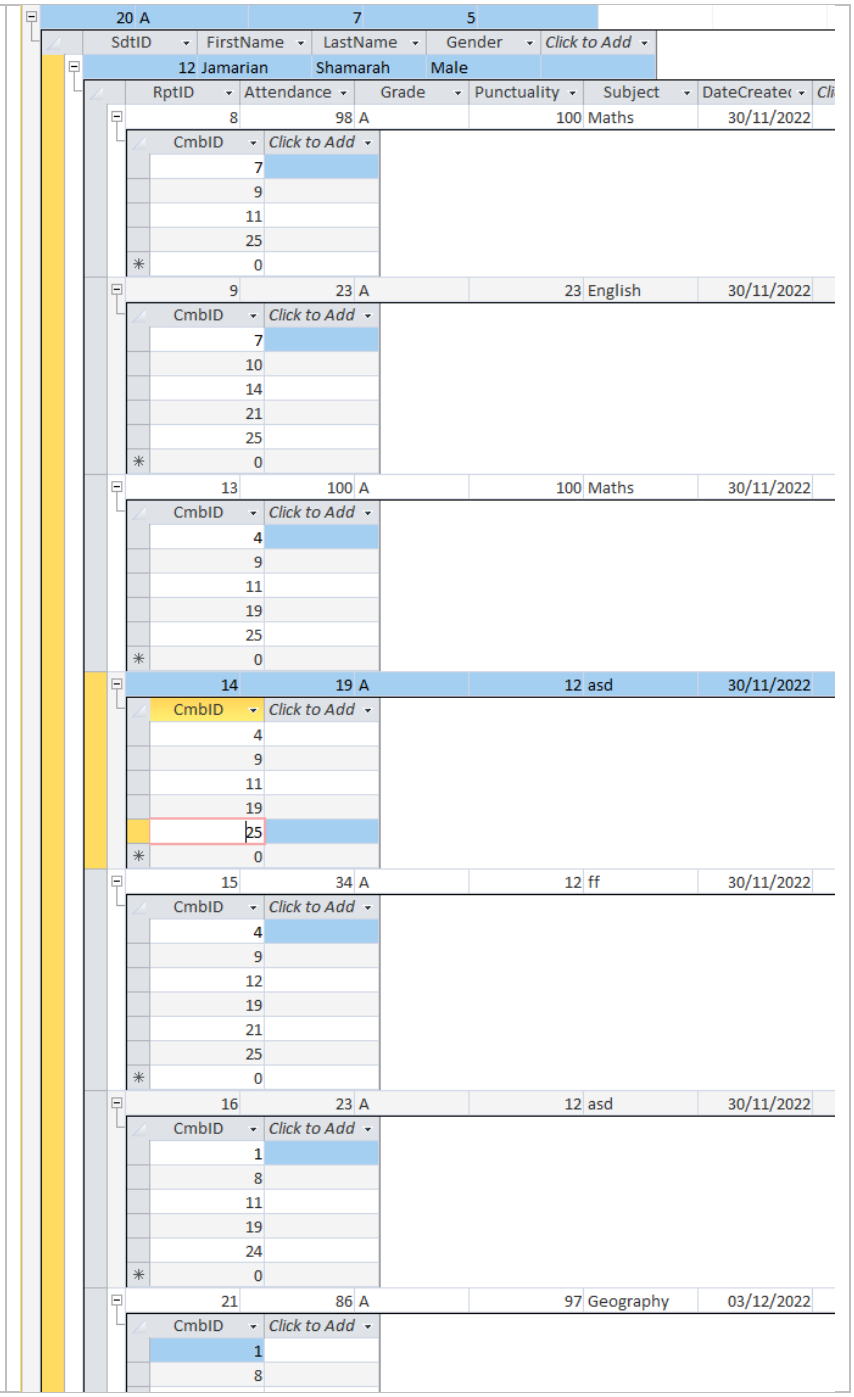
Test No 7.2 result: Checking if the form can allow users to filter the table rows by year…

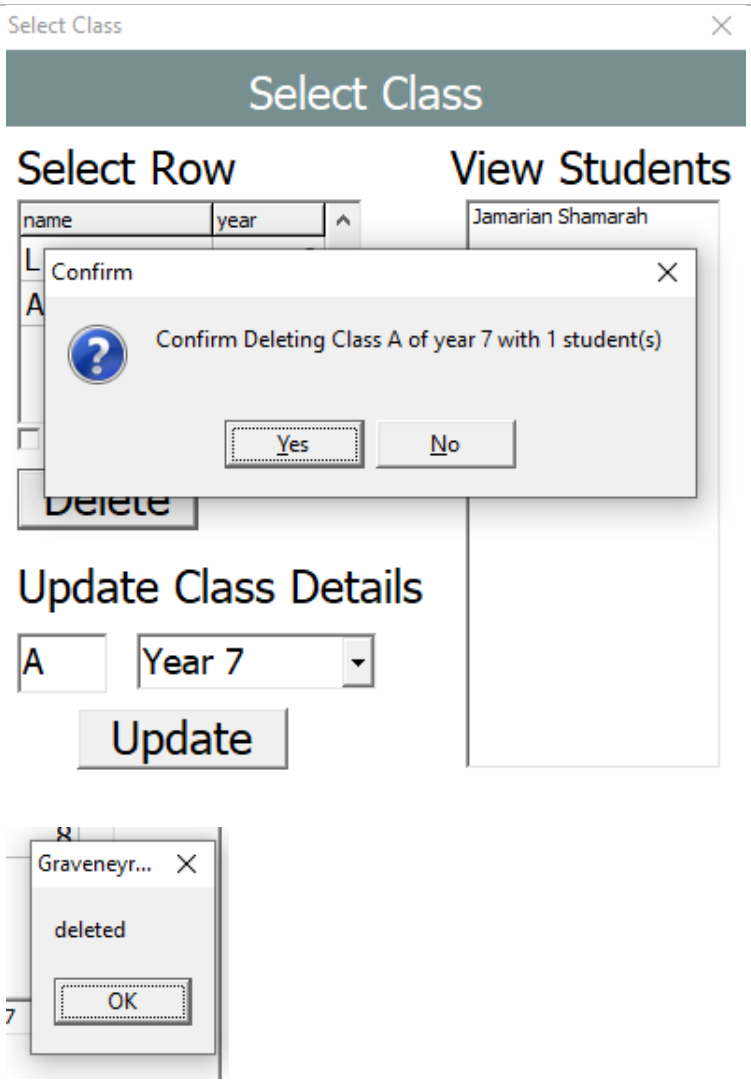
1) Selecting year 7	

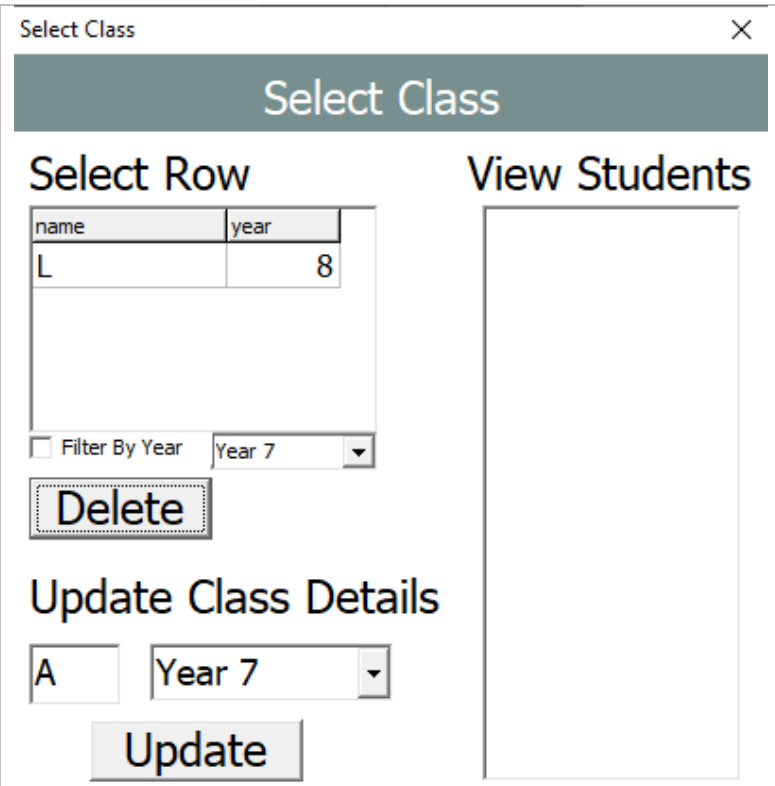
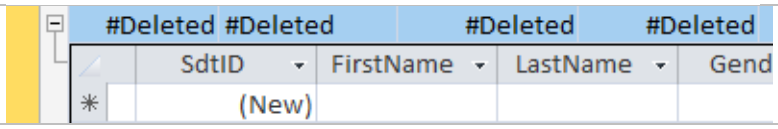
2) Selecting year 9	
Summary: Test Successful – The form is able to filter the table by year	

Test No 7.3 result: Checking if the delete button can delete the class and data correctly…

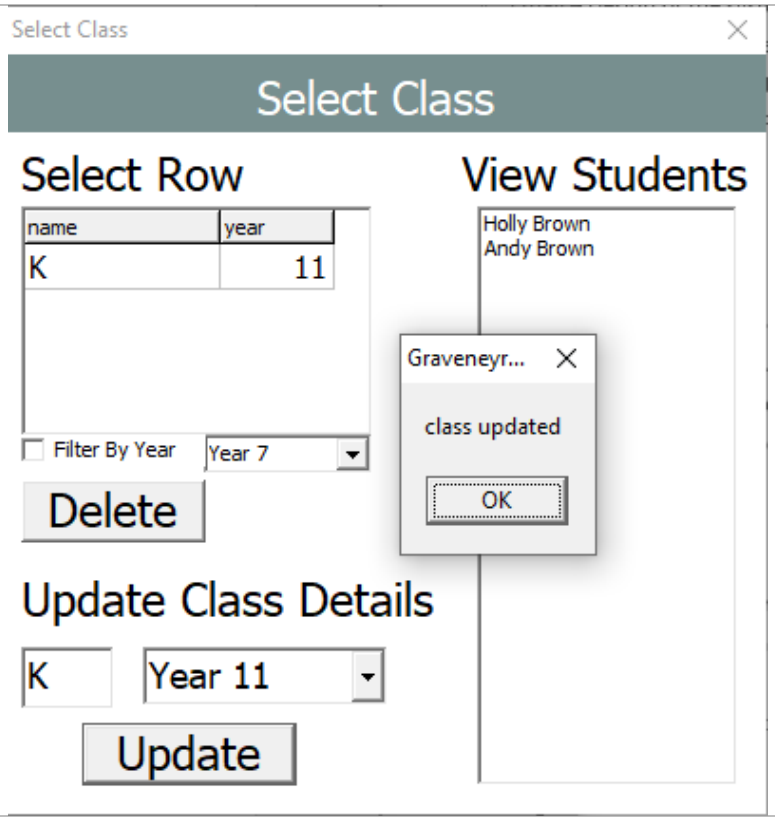
Data of class A year 7 before



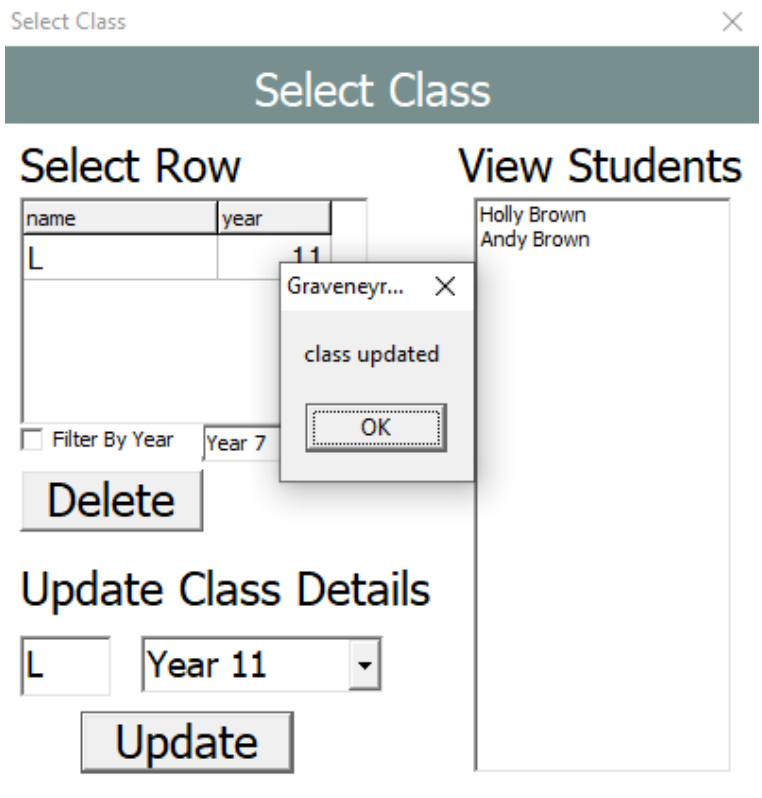
Deleting the class	

	
Checking if all data is deleted	
Summary: Test Successful – The system was able to successfully remove all the data of the class. It was also able to confirm deleting it, avoiding accidental miss click. It also informed that it has been deleted.	

Test No 7.4.1 result: Checking if the form can update both class name and year with valid values…

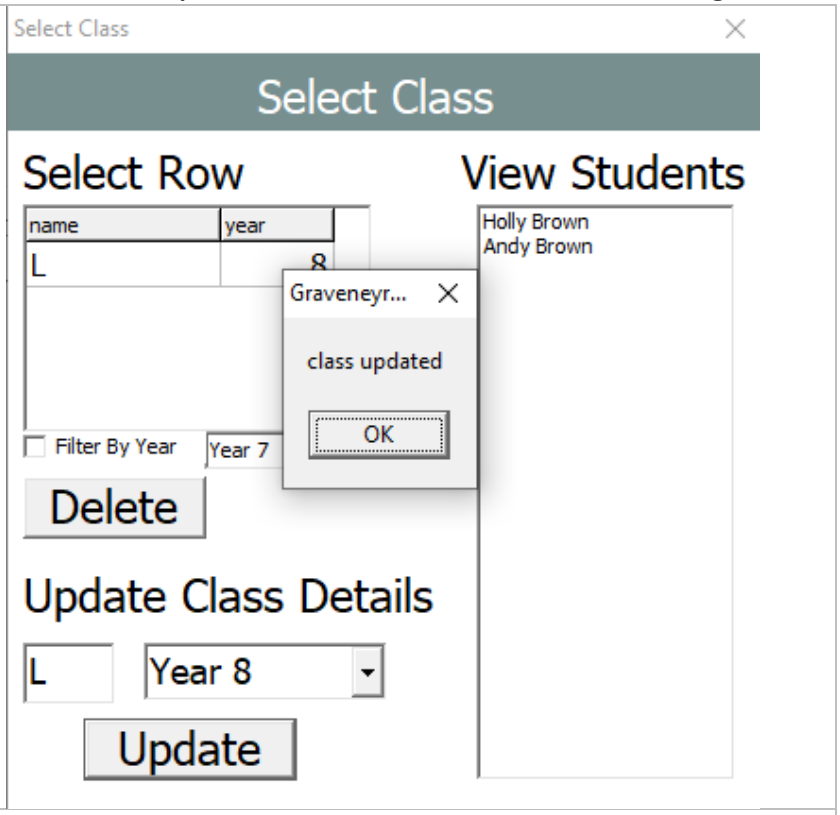
Inputting the values and clicking update	
Summary: Test Successful – The form is able to update the class with valid values. It also informs the user that the class has been updated.	

Test No 7.4.2 result: Checking if the form can update the class with the value of year remaining the same

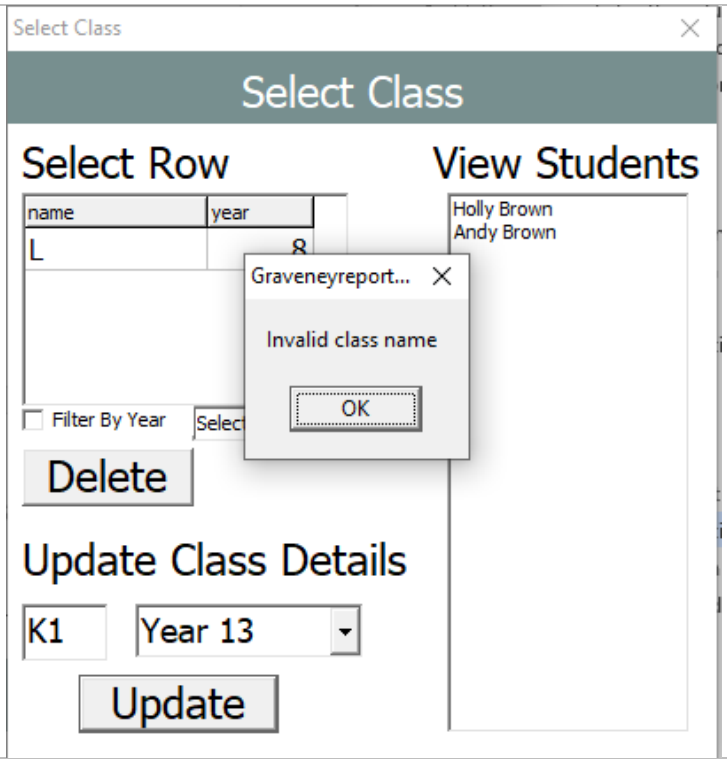
Inputting the values and clicking update	

Summary: Test Successful – The form is able to update the class with valid values. It also informs the user that the class has been updated.

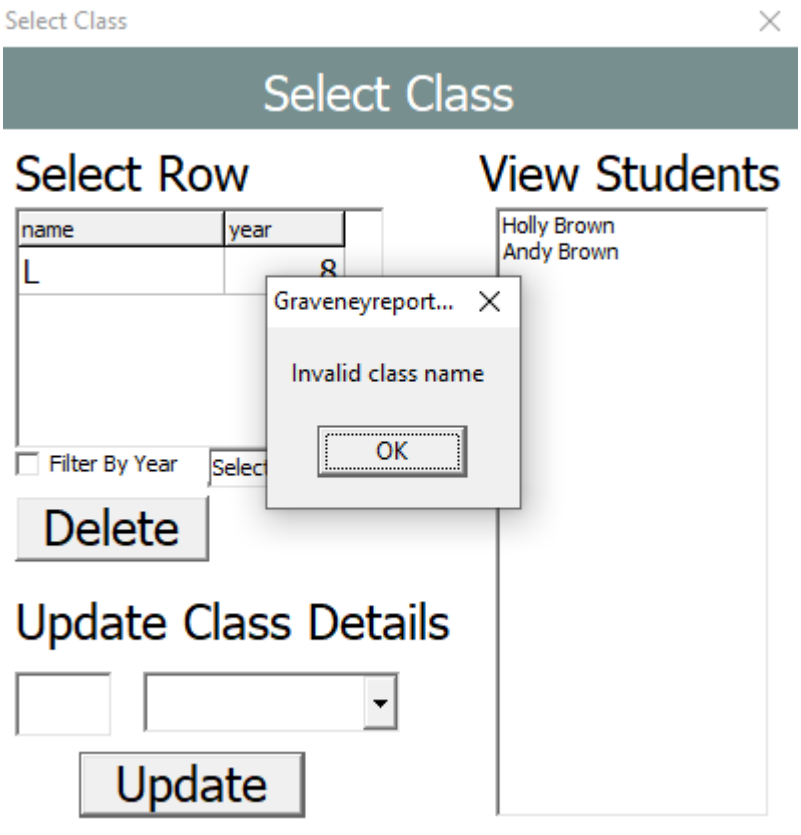
Test No 7.4.3 result: Checking if the form can update the class with the value of name remaining the same

Inputting the values and clicking update	
Summary: Test Successful – The form is able to update the class with valid values. It also informs the user that the class has been updated.	

Test No 7.5 result: Checking if the form can update the class with invalid values

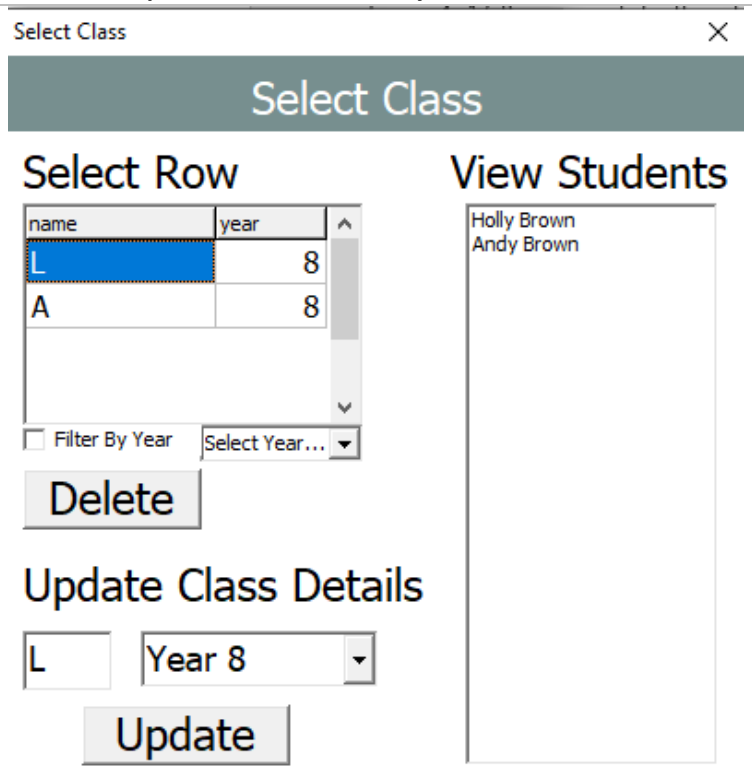
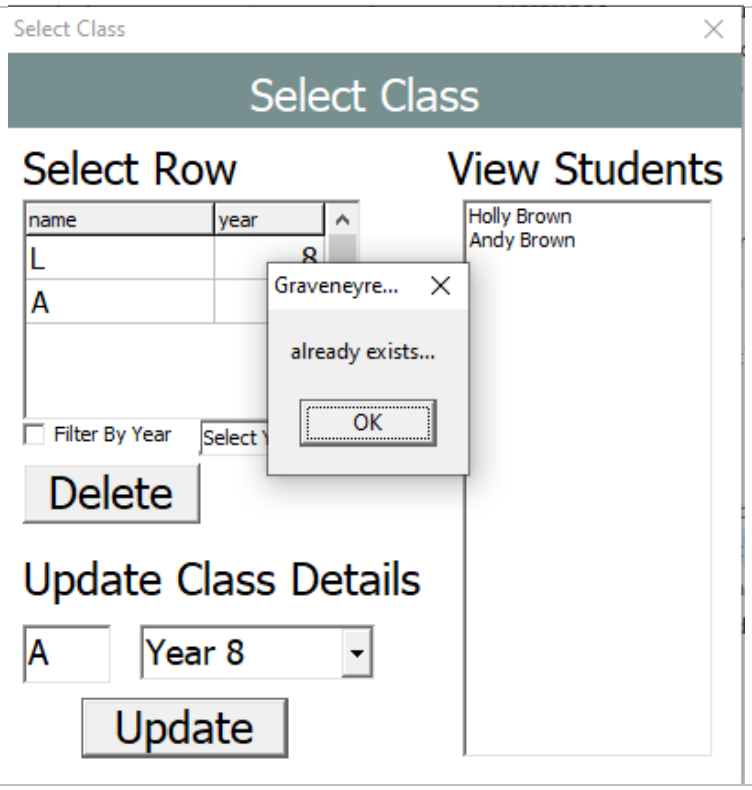
Inputting the values and clicking update	
Summary: Test Successful – The form is able to validate class values correctly before updating the class. It is also able to infrom the user the class has not been updated.	

Test No 7.6 result: Checking if the system can prevent user from updating the class into null values…

Inputting the values and clicking update	

Summary: Test Successful – The form is able to check if the values are null. The form is able to inform the user when the class has not been updated.

Test No 7.7 result: Checking if the form can prevent data redundancy…

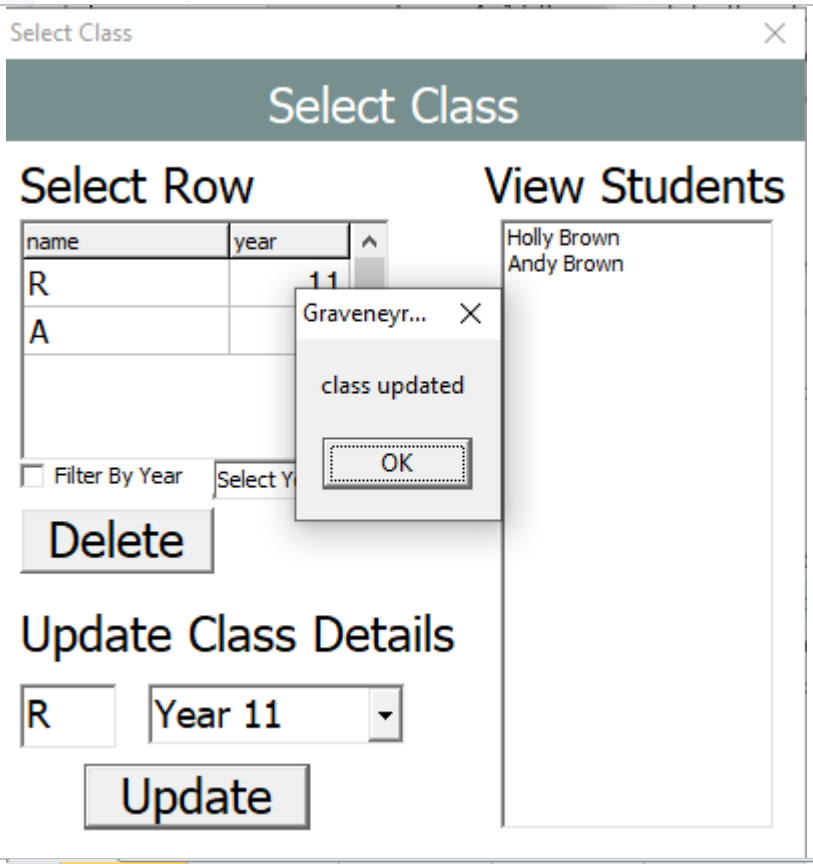
Before	
Inputting the values and clicking update	
Summary: Test Successful – The form is able to prevent data redundancy of user's data. It informs the user that the class has not been updated and the reason why.	

Test No 7.8 result: Checking if the system can prevent other users' data being used…

Database Before

classID	Name	Year	StfID	Click to Add
18	L	8	5	
21	R	11	2	
22	A	8	5	
(New)		0	0	

Inputting the values and clicking update



Database After

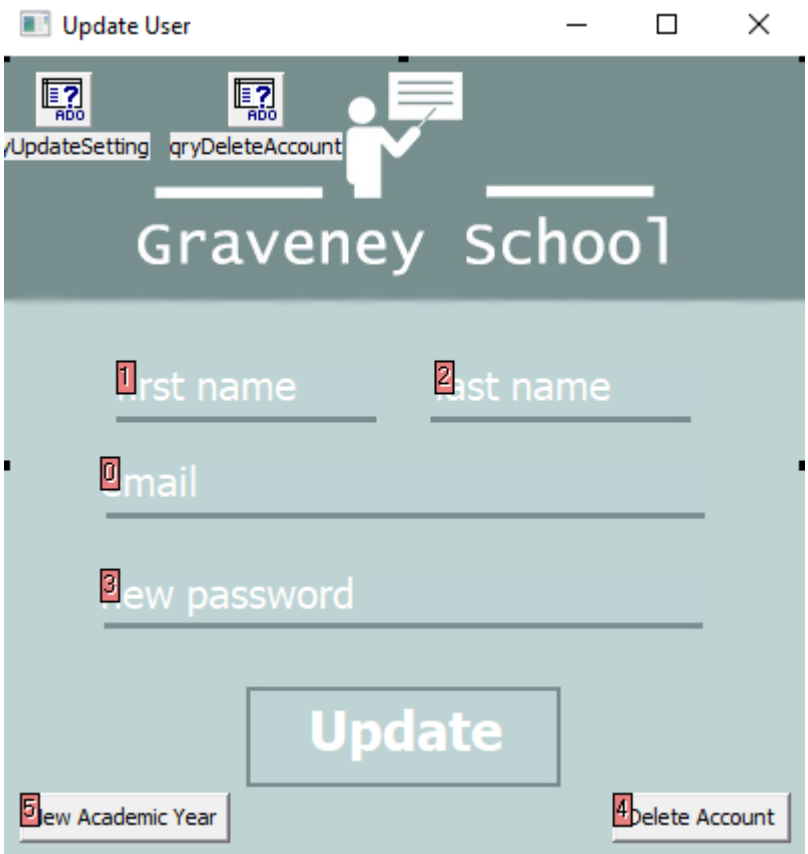
classID	Name	Year	StfID	Click to Add
18	R	11	5	
21	R	11	2	
22	A	8	5	
(New)		0	0	

Summary: Test Successful – The form is able to distinct between different staff data. The system only accesses the data of the user who is logged in. The system allows the user to update the class to the same values of another user's class.

Prototype 2

The user should be able to increment all years by pressing a single button. Prototype 2 implements the solution for it.

Updated Screenshot of "Update User Form"



Justification: There is a new button (btnIncrementClasses) created to meet requirement 13.3. Its function is to delete all year 11 and 13s classes and data and implement the year for other classes by one. This is to be used by the user on the start of a new academic year.

Changes in Implemented Code for Update Setting Form (unit_UpdateSetting)

```
…
procedure TfrmUpdateSetting.btnIncrementClassesClick(Sender: TObject);
begin
  case MessageDlg('Incrementing all classes by one will result in current Y11 and
Y13s being deleted...', mtConfirmation, [mbYes, mbNo], 0) of
    mrYes:
    begin
      with qryupdatesetting do
      begin
        sql.text := 'DELETE [Report Comments].*  '+
        'FROM (Staff INNER JOIN (Class INNER JOIN Student ON Class.classID =
Student.classID) '+
        'ON Staff.StfID = Class.StfID) INNER JOIN (Report INNER JOIN [Report
Comments] ON '+
        'Report.RptID = [Report Comments].RptID) ON Student.SdtID = Report.SdtID '+
        'WHERE ((Class.Year=11 Or Class.Year=13) AND
(Staff.stfID='+inttostr(info.StaffID)+'));';
        ExecSQL;

        sql.text := 'DELETE Report.* '+
```

```
        'FROM (Staff INNER JOIN (Class INNER JOIN Student ON Class.classID =
Student.classID) '+
        'ON Staff.StfID = Class.StfID) INNER JOIN Report ON Student.SdtID =
Report.SdtID '+
        'WHERE (((Class.Year)=11 Or (Class.Year)=13) AND
(Staff.stfID='+inttostr(info.StaffID)+'));';
        ExecSQL;

        sql.text := 'DELETE Student.* '+
        'FROM Staff INNER JOIN (Class INNER JOIN Student ON Class.classID =
Student.classID) '+
        'ON Staff.StfID = Class.StfID '+
        'WHERE (((Class.Year)=11 Or (Class.Year)=13) AND
(Staff.stfID='+inttostr(info.StaffID)+'));';
        ExecSQL;

        sql.text := 'DELETE Class.* '+
        'FROM Staff INNER JOIN Class ON Staff.StfID = Class.StfID '+
        'WHERE (((Class.Year)=11 Or (Class.Year)=13) AND
(Staff.stfID='+inttostr(info.StaffID)+'));';
        ExecSQL;

        sql.text := 'UPDATE Staff INNER JOIN Class ON Staff.StfID = Class.StfID SET
Class.[Year]'+
        ' = [class].[year]+1 where Staff.stfID='+inttostr(info.StaffID)+';';
        ExecSQL;
        showmessage('all classes have been incremented');
      end;

    end;
    mrNo:
    begin
      ShowMessage('cancelled');
    end;
  end;
end;
…
```

What has been done

The Update Setting form has been modified to meet criteria 6. It allows the user to increment the class years by one and delete all previous year 11 and 13s classes and its data, since they are the last academic years, hence requirement 13.3 has been met. The prototype was tested in runtime of the code. Screenshots of results of the tests are shown below.

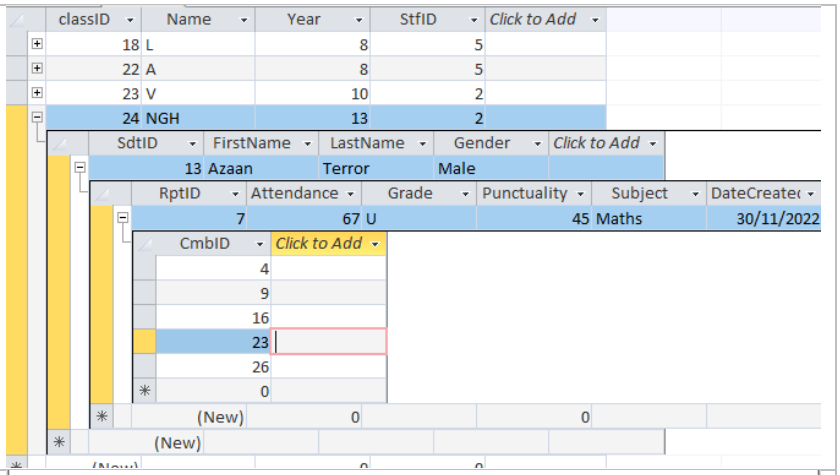
Summary: The user is able to create and sign in using an account. The user is able to update their details. The main menu form has also been created, which will allow users to open different forms for the solution. They are also able to create classes and students within it. They can also view all students within a class, delete the class along with the students and their data inside the class, and update classes. Users are able to increment all years by one and delete unneeded data at the start of a new academic year by pressing a single button.

Testing of solution

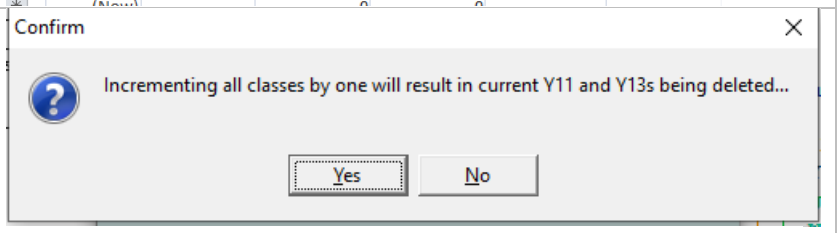
Test carried out are 7.9

Test No 7.9 result: Checking if the system can change data for a new academic year…

Database Before

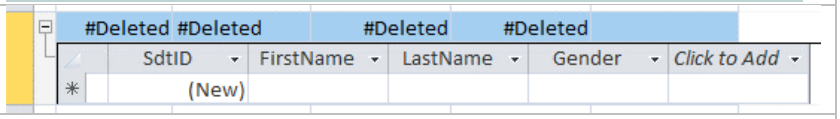


Pressing the button, logged into staff 2.





Database After

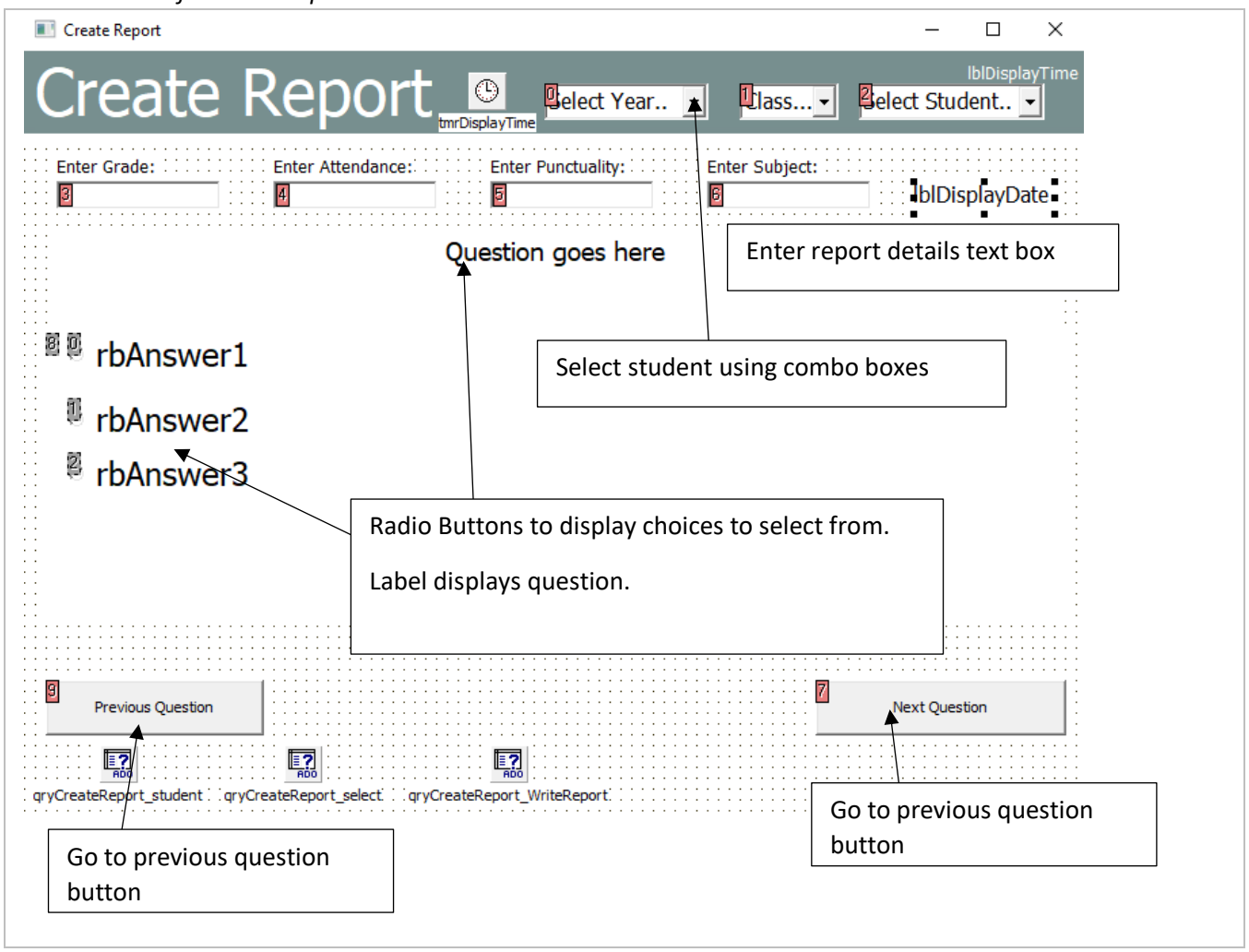


Summary: Test Successful – The form is able to let the user change its data by clicking a single button. It increments the years and can delete year 11 and 13 classes and their data.

Stage 8: Implementing the "Create Report Form"

Prototype 1

Screenshot of "Create Report Form"



Changes in Implemented Code for "Main Menu Form" (unit_menu)

```
Uses … ,ADODB; //needed to use querys
…
type YearArr = array[1..7] of string; //max length it will be is 7, can be less
…
  public
    function selectYear(qry, frm:Tobject):yearArr; //declared in frmMainMenu class
as public
…
function Tfrmmainmenu.selectYear(qry, frm:Tobject):yearArr;
var i:integer;
  arr:yearArr;
begin
  //selects all years which the user has created class for in ascending order
  with TForm(frm) do
  begin
    TADOQuery(qry).SQL.Text:='select year from class where StfID =
'+inttostr(info.staffID)+' group by year order by year;';
    TADOQuery(qry).Open;
    for i:=1 to TADOQuery(qry).recordcount do
```

```
    begin
      arr[i]:= 'Year '+ TADOQuery(qry).fieldbyname('year').AsString;
      TADOQuery(qry).Next;
    end;
    TADOQuery(qry).Close;
    selectyear:=arr;
  end;
end;
```

Coded Section Before	Coded Section After

`procedure TfrmMainMenu.btnCreateStudentClick(Sender : TObject);`
`var i:Integer;`
`begin`
`  with frmCreateStudent do`
`  begin`
`    cboSelectYear.Clear;`
`    qryCreateStudent.SQL.Text:='select year from class where StfID = '+inttostr(info.staffID)+' group by year order by year;';`
`    qryCreateStudent.Open;`
`  end;`
`  for i:=1 to frmcreatestudent.qrycreatestudent.recordcount do`
`  begin`
 	
`frmCreateStudent.cboSelectYear.items.Add('Year' +frmCreateStudent.qryCreateStudent.fieldbyname ('year').AsString);`
 	
`frmCreateStudent.qryCreateStudent.Next;`
`  end;`
 	
`frmCreateStudent.qryCreateStudent.Close;`
`  **//open form**`
`  frmCreateStudent.ShowModal;`
`end;`
	`//Create student button - label component used as a button`
	`procedure TfrmMainMenu.btnCreateStudentClick (Sender: TObject);`
	`var i:Integer;`
	`  arr:yearArr;`
	`begin`
	`  arr := arr := selectyear(frmCreateStudent.qryCreateStudent, frmcreatestudent);`
	`frmcreatestudent.cboSelectyear.Clear;`
	`  for i:=1 to length(arr) do`
	`  begin`
	`    if (arr[i]='') then`
	`      break`
	`    else`
	`      frmCreateStudent.cboSelectYear.items.Add(arr[i]);`
	`  end;`
	`  frmCreateStudent.ShowModal;`
	`end;`

`procedure TfrmMainMenu.btnCreateReportClick(Sender: TObject);`
`begin`
`  **//open form**`
`end;`
	`//Create Report button - label component used as a button`
	`procedure TfrmMainMenu.btnCreateReportClick(Sender: TObject);`
	`var i:Integer;`
	`  arr:yearArr;`
	`begin`
	`  FreeAndNil(frmCreateReport);`
	`  frmCreateReport := TfrmCreateReport.Create(frmCreateReport);`
	`  with frmCreateReport do`
	`  begin`
	`    arr := selectyear(qryCreateReport_Select, frmcreatereport);`
	`    cboSelectyear.Clear;`
	`    for i:=1 to length(arr) do`
	`    begin`

	if (arr[i]='') then
	break
	else
	
	cboSelectYear.items.Add(arr[i]);
	end;
	questionToDisplay(questionCount);
	ShowModal;
	end;
	end;

uses unit_login, unit_createClass, unit_createStudent, Unit_UpdateSetting, unit_selectClass;	uses unit_login, unit_createClass, unit_createStudent, Unit_UpdateSetting, unit_selectClass, unit_CreateReport;
Justification: Allow user to open the Create Report Form from main menu. It makes sure the data of the form is cleared before opening it.	

Changes in Implemented Code for Create Student Form (unit_CreateStudent)

function
getClassYear(year:string):string;
begin
year := year[Length(year)];
if (StrToInt(year)<7)then
year:=IntToStr(StrToInt(year)+10);
getClassYear:=year;
end;
	Public
	…
	Function getClassYear (year:string)
	:string; **//used in Select Student**
	
	function
	TfrmCreateStudent.getClassYear(year:string):
	string;
	begin
	year := year[Length(year)];
	if (StrToInt(year)<7)then
	year:=IntToStr(StrToInt(year)+10);
	getClassYear:=year;
	end;
	…
	procedure writeStudent;
	var classID:String;
	begin
	//Getting ClassID of the selected class
	frmCreateStudent.qrycreatestudent.SQL.Text
	:= 'Select classID from class where
	year='+frmCreateStudent.getClassYear(frmCrea
	teStudent.cboSelectYear.Text)+' and
	stfID='+inttostr(info.staffID)+' and
	name='+quotedStr(frmCreateStudent.cboselectc
	lass.Text)+';';

Justification: Made Public to be used in Create Report unit	

Implemented Code for Create Report Form (unit_CreateReport)

```
unit Unit_CreateReport;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, jpeg, ExtCtrls, StdCtrls, DB, ADODB, unit_login, unit_createStudent,
  unit_signup;

type //Question info
  Tquestion = record
    number:integer;
```

```
    text:string;
  end;

type
  TStudentInfo = class
  private
    fName, lName, year, gender, id, cName:string;
  public
    procedure writeSelectedStudent(selectedIndex:integer); // writes the details of
the selected student
    procedure resetSelectedStudent; //clears the data of the selected student
variable
    constructor Create; overload;
  end;

type
  Tquiz = class
  private
    question:Tquestion;
    answerNumber:integer;
  public
    procedure WriteQuestion(Value:string;number:integer);
    function getQuestionNumber:integer;
    procedure displayAnswerChoice(a1, a2,a3:string);
    procedure inputAnswer(Value:integer);
    function outputAnswer:integer;
    constructor Create; overload;
  end;

type
  TfrmCreateReport = class(TForm)
  Published
    qryCreateReport_WriteReport: TADOQuery;
    ImgBackground: TImage;
    Label1: TLabel;
    cboSelectYear: TComboBox;
    qryCreateReport_student: TADOQuery;
    cboSelectClass: TComboBox;
    cboSelectStudent: TComboBox;
    btnPreviousQuestion: TButton;
    qryCreateReport_select: TADOQuery;
    lbledtSdtGrade: TLabeledEdit;
    lbledtSdtAttendance: TLabeledEdit;
    lbledtSdtPunctuality: TLabeledEdit;
    lbledtSdtSubject: TLabeledEdit;
    lblDisplayDate: TLabel;
    tmrDisplayTime: TTimer;
    lblDisplayTime: TLabel;
    btnNextQuestion: TButton;
    lblDisplayQuestion: TLabel;
    pnlAnswerSection: TPanel;
    rbAnswer1: TRadioButton;
    rbAnswer2: TRadioButton;
    rbAnswer3: TRadioButton;
    procedure cboSelectYearChange(Sender: TObject);
    procedure cboSelectClassChange(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure tmrDisplayTimeTimer(Sender: TObject);
    procedure cboSelectStudentChange(Sender: TObject);
    procedure btnQuestionClick(Sender: TObject);
    procedure rbAnswerClick(Sender: TObject);
```

```
  private
    studentSelect:TStudentInfo; //Selected Student Info
    reportCommentNumber:array[1..6] of integer; //comment ID stored to link to
reports
    //1- opening, 2-behavior, 3-progress, 4-strength, 5-weakness, 6-closing comment
    ReportID:string; //used to link comments to reports
    procedure SelectAnAnswer(AnswerNum:integer;Bool:boolean);
    procedure GenerateReport;
  public
    questionCount:integer; //used in main menu unit
    behavior, effortinClass, homework, understandingOfContent, WorkingTarget,
contribution:Tquiz; //used in main menu unit
    procedure QuestionToDisplay(questionNum:integer); //used in main menu unit
    function matchQuestiontoVar(questionNum:integer):Tquiz;
  end;

var
  frmCreateReport: TfrmCreateReport;


implementation

{$R *.dfm}

//creates new quiz question with null values
constructor Tquiz.create;
begin
  question.text := '';
  question.number := 0;
  answerNumber := 0;
end;

//writes values to the question of the quiz, displays it to form
procedure Tquiz.WriteQuestion(Value:string;number:integer);
begin
  question.text := Value;
  question.number := number;
  frmCreateReport.lbldisplayquestion.Caption := inttostr(number) + ') ' + value;
end;

//writes possible answers to form
procedure Tquiz.displayAnswerChoice(a1,a2,a3:string);
begin
  frmCreateReport.rbAnswer1.Caption := a1;
  frmCreateReport.rbAnswer2.Caption := a2;
  frmCreateReport.rbAnswer3.Caption := a3;
end;

//read current question number
function Tquiz.getQuestionNumber:integer;
begin
  getQuestionNumber := question.number;
end;

//write answer, either 1, 2 , 3; depends on which rb selected
procedure Tquiz.inputAnswer(value:integer);
begin
  answerNumber := value;
end;

//output answer number selected 1,2 or 3
```

```
function Tquiz.outputAnswer:integer;
begin
  outputAnswer := answerNumber;
end;

//Which question to display depending on question number
procedure Tfrmcreatereport.QuestionToDisplay(questionNum:integer);
begin
  if (questionNum = 1) then //behavior
  begin
    behavior.WriteQuestion('How is the students behavior been this term?',1);
    behavior.displayAnswerChoice('Excellent','Okay(distracted time to
time)','Poor(distraction to other students)');
  end
  else if (questionNum = 2) then //effort in class
  begin
    effortInclass.WriteQuestion('Students effort in class is ...',2);
    effortInclass.displayAnswerChoice('Excellent','Okay','Poor(e.g. does not get
though tasks in class)');
  end
  else if (questionNum = 3) then //homework
  begin
    homework.WriteQuestion('Students homeworks are...',3);
    homework.displayAnswerChoice('Excellent','Okay(e.g. hands it in late or not to
the best standards)','Poor(e.g. Never completes homeworks)');
  end
  else if (questionNum = 4) then //understanding of content
  begin
    understandingOfContent.WriteQuestion('How much of the content does the student
understand? ',4);
    understandingOfContent.displayAnswerChoice('All of it','Some/ Most of it','few
of it');
  end
  else if (questionNum = 5) then //Working Target
  begin
    WorkingTarget.WriteQuestion('The student is currently working...',5);
    WorkingTarget.displayAnswerChoice('Above their expected grade of the term','At
their expected grade of the term','Below their expected grade of the term');
  end
  else if (questionNum = 6) then //contribution
  begin
    contribution.WriteQuestion('Does the student contribute in class?',6);
    contribution.displayAnswerChoice('All the time','Some/ Most of the times','Not
at all');
  end;
end;

//checks what question number matches to which quiz variable it represents
function TfrmCreateReport.matchQuestiontoVar(questionNum:integer):Tquiz;
begin
  case questionNum of
    1:matchQuestiontoVar := behavior;
    2: matchQuestiontoVar := effortInclass;
    3:matchQuestiontoVar := homework;
    4: matchQuestiontoVar := understandingOfContent;
    5:matchQuestiontoVar := WorkingTarget;
    6: matchQuestiontoVar := contribution;
  end;
end;

//Creates object of class type TStudentInfo
```

```
constructor TStudentInfo.create;
begin
  fName := '';
  lName := '';
  year := '';
  cName := '';
  gender := '';
  id := '';
end;

//stores the selected students details in the system
procedure TStudentInfo.writeSelectedStudent(selectedIndex:integer);
var i:integer;
begin
  with frmCreateReport.qrycreatereport_student do
  begin
    SQL.Text:='SELECT student.sdtID, student.FirstName, student.LastName,
student.Gender, class.year, class.name FROM Class INNER JOIN Student'+
    ' ON Class.classID = Student.classID WHERE
class.name='+quotedStr(frmCreateReport.cboselectclass.Text)+
    ' and class.stfID = '+inttostr(info.staffID)+' and class.year = '+
frmcreatestudent.getClassYear(frmCreateReport.cboselectyear.Text)+' ORDER BY
Student.firstname;';
    open;
    for i:=1 to selectedIndex do
    begin
      next;
    end;
    fName := fieldbyname('firstname').asstring;
    lName := fieldbyname('lastname').asstring;
    year := fieldbyname('year').asstring;
    gender := fieldbyname('gender').asstring;
    id := fieldbyname('sdtID').asstring;
    cname := fieldbyname('name').asstring;
  end;
end;

//Clears the stored data of the selected student, to prevent reports created on
students who were previosly selected
procedure  TStudentInfo.resetSelectedStudent;
begin
  fName := '';
  lName := '';
  year := '';
  cName := '';
  gender := '';
  id := '';
end;

//whenever the year selected is changed, the system outputs the correspoding classes
procedure TfrmCreateReport.cboSelectYearChange(Sender: TObject);
begin
  cboSelectStudent.Clear; //deletes any previous students in the combo box
  cboSelectClass.Clear; //deletes any previous classes in the combo box
  try
    frmcreatestudent.selectClasses (frmCreateReport.cboSelectClass,
    frmCreateReport.cboselectYear); // finds the classes in relation to it
  except //if user inputs invalid value, it resets the combo box
    cboselectyear.Text:='Select Year...';
    cboselectclass.Clear;
  end;
```

```
  //resets all values
  cboselectstudent.clear;
  cboselectstudent.itemindex:=-1;
  cboselectstudent.text:='Select Student..';
  cboselectclass.itemindex:=-1;
  cboselectclass.text:='Class...';
  studentSelect.resetSelectedStudent;
end;

//whenever the class selected is changed, the system needs to display the
correspoding students
procedure TfrmCreateReport.cboSelectClassChange(Sender: TObject);
var i:integer;
begin
  cboSelectStudent.Clear; //deletes any previous students in the combo box
  qryCreateReport_student.SQL.Text:='SELECT student.FirstName, student.LastName FROM
Class INNER JOIN Student'+
  ' ON Class.classID = Student.classID WHERE
class.name='+quotedStr(cboselectclass.Text)+' and class.stfID =
'+inttostr(info.staffID)+' and class.year =
'+frmcreatestudent.getClassYear(cboSelectYear.text)+' ORDER BY Student.firstname;';
  qrycreatereport_student.open;

  //adds the students to combo box in relation to the class selected
  for i:=1 to qryCreateReport_student.RecordCount do
  begin
    cboselectstudent.Items.Add(
    qryCreateReport_student.fieldbyname('firstname').AsString + ' ' +
    qryCreateReport_student.fieldbyname('lastname').AsString);
    qryCreateReport_student.Next;
  end;

  cboselectstudent.itemindex:=-1;
  if (qryCreateReport_student.RecordCount=0)then
    cboselectstudent.text:='No Student(s)' //when no students are in that class
  else
    cboselectstudent.text:='Select Student..'; //when there are students in the
class
  qryCreateReport_student.close;
  studentSelect.resetSelectedStudent; //resets the values of student
end;

//writes the values of the student to the variable, frequently used in system.
procedure TfrmCreateReport.cboSelectStudentChange(Sender: TObject);
begin
  studentSelect.writeSelectedStudent(cboSelectStudent.ItemIndex);
end;

//Gets date, needed to sort reports created; a student will have multiple reports.
procedure TfrmCreateReport.FormCreate(Sender: TObject);
var Date:TDateTime;
begin
  Date:=now;
  lblDisplayDate.Caption:=DatetoStr(Date); //writes it to the label
  //updates every 0.5 second(s)

  //Creates All objects needed
  questionCount := 1;
  Behavior := Tquiz.create;
  effortInClass := Tquiz.create;
  homework := Tquiz.create;
```

```
  understandingOfContent := Tquiz.create;
  workingTarget := Tquiz.create;
  contribution := Tquiz.create;
  studentSelect := TStudentInfo.Create;
end;

//displays time for the user, lets user manage their time when writing multiple
reports
procedure TfrmCreateReport.tmrDisplayTimeTimer(Sender: TObject);
begin
  lblDisplayTime.Caption:=TimetoStr(time);
end;

//selects one of the choices; used if user is going back and forward to display
answers. Also used to deselect the selected answer when going to a new question
procedure TfrmCreateReport.selectAnAnswer(AnswerNum:integer;Bool:Boolean);
begin
  case AnswerNum of
    1:rbAnswer1.Checked := Bool;
    2:rbAnswer2.Checked := Bool;
    3:rbAnswer3.Checked := Bool;
  end;
end;

//validates form
function isReportFrmValid:boolean;
var isInvalid:boolean;
begin
  isInvalid := False;
  with frmCreateReport do
  begin
    //makes sure the form is not empty
    if ((cboSelectStudent.ItemIndex = -1) or (lbledtSdtSubject.Text = '') or
(lbledtSdtGrade.Text = '') or (lbledtSdtAttendance.Text = '') or
(lbledtSdtPunctuality.text = '') or (matchQuestiontoVar(questionCount).outputAnswer
= 0))then
    begin
      isInvalid := true;
      showmessage('Complete the Form..');
    end
    else
    begin
      //Grade can only be one character (e.g. A or 9)
      if (length(lbledtSdtGrade.Text)=1)then
      begin
        //makes sure it has a numerical or alphabetical value only
        if not(frmsignup.stringisAlpha(lbledtSdtGrade.Text))then
        begin
          try
            strtoint(lbledtSdtGrade.Text)
          except
            //informs user
            showmessage('Grade must be Alphabetical or Numerical');
            isInvalid := True;
          end;
        end
      end
      else
      begin
        //informs user
        showmessage('Grade is a single character');
```

```
        isInvalid := True;
      end;

      //subject name is alhphabetical letters only
      if not(frmsignup.stringisAlpha(lbledtSdtSubject.text))then
      begin
        //informs user
        showmessage('Subject Name must be Alphabetical with no spaces');
        isInvalid := True;
      end
      else
      begin
      //validating attendance and punctuality
        try
          //percentage is a number which is less than 100 and greater than 0
          if ((strtoint(lbledtSdtAttendance.Text) > 100) or
(strtoint(lbledtSdtAttendance.Text) < 0) or (strtoint(lbledtSdtPunctuality.text) >
100) or (strtoint(lbledtSdtPunctuality.text) < 0)) then
          begin
            //informs user
            isInvalid := true;
            showmessage('Puntuality or Attendance is invalid');
          end;
        except
          //informs user
          isInvalid := true;
          showmessage('Puntuality or Attendance is invalid');
        end;
      end;
    end;
  end;
  //outputs result, true is valid, false is invalid.
  isReportFrmValid := not(isInvalid);
end;

//algorithm to work out which opening comment to select from the quiz answers
procedure OpeningComment; //always results in an answer for opening comment
begin
  with frmCreateReport do
  begin
    if ((effortInClass.outputAnswer = 1) and (behavior.outputAnswer = 1)) then
    begin
      if (understandingOfContent.outputAnswer = 3) then
        reportCommentNumber[1] := 5
      else if (homework.outputAnswer = 1) then
        reportCommentNumber[1] := 1
      else if (understandingOfContent.outputAnswer = 1) then
        reportCommentNumber[1] := 2
      else
        reportCommentNumber[1] := 3;
    end
    else if (understandingOfContent.outputAnswer = 3) then
    begin
      if (workingTarget.outputAnswer = 3) then
        reportCommentNumber[1] := 6
      else
        reportCommentNumber[1] := 7;
    end
    else
      reportCommentNumber[1] := 4;
  end;
```

```
end;

//algorithm to work out which Closing comment to select from the quiz answers
procedure ClosingComment; // Always generates a comment
begin
  with frmCreateReport do
  begin
    if (workingTarget.outputAnswer = 3) then
      reportCommentNumber[6] := 26
    else
    begin
      if (behavior.outputAnswer = 1) then
        reportCommentNumber[6] := 24
      else
        reportCommentNumber[6] := 25;
    end;
  end;
end;

//algorithm to work out which closing and opening comments to select from the quiz
answers
procedure strengthsAndWeakness; // Does not always generate
begin
  with frmCreateReport do
  begin
    if (understandingOfContent.outputAnswer = 1)then
      reportCommentNumber[4] := 19
    else
    begin
      if (effortInClass.outputAnswer = 1)then
        reportCommentNumber[4] := 20;
    end;
    if (workingTarget.outputAnswer = 3) then
      reportCommentNumber[5] := 23
    else
    begin
      if (homework.outputAnswer > 1) then
        reportCommentNumber[5] := 21
      else if (contribution.outputAnswer = 3) then
        reportCommentNumber[5] := 22;
    end;
  end;
end;

//algorithm to work out which progress comment to select from the quiz answers
procedure progress; // Always generates a comment
begin
  with frmCreateReport do
  begin
    if (workingTarget.outputAnswer = 3) then
    begin
      if (behavior.outputAnswer > 1) then
        reportCommentNumber[3] := 16
      else if (understandingOfContent.outputAnswer = 3)then
        reportCommentNumber[3] := 17
      else
        reportCommentNumber[3] := 18;
    end
    else
    begin
      if (homework.outputAnswer = 1) then
```

```
        reportCommentNumber[3] := 11
      else if (effortinclass.outputAnswer = 1) then
        reportCommentNumber[3] := 12
      else if (homework.outputAnswer > 1) then
        reportCommentNumber[3] := 14
      else if (understandingOfContent.outputAnswer > 1) then
        reportCommentNumber[3] := 15
      else
        reportCommentNumber[3] := 13;
    end;
  end;
end;

//writes the report to the database
procedure writeReport(attendance, grade, punctuality, subject, date:string);
var i:integer;
begin
  with frmCreateReport do
  begin
    qryCreateReport_writereport.SQL.Text := 'insert into
Report(SdtID,Attendance,Grade,Punctuality,Subject,dateCreated) '+
    'values ('+studentSelect.id+','+attendance+','+ quotedStr(grade)+','+
punctuality+','+ quotedStr(subject)+','+ quotedStr(date)+');';
    qryCreateReport_writereport.ExecSQL;
    qryCreateReport_select.SQL.Text := 'select RptID from Report where sdtID =
'+studentSelect.id+' order by RptID Desc;';
    qryCreateReport_select.Open;
    ReportID := qryCreateReport_select.fieldbyname('RptID').AsString;
    qryCreateReport_select.close;
    for i:=1 to 6 do
    begin
      if (reportCommentNumber[i] <> 0)then //if its 0, then no comment was selected
for that topic
      begin
        qryCreateReport_writereport.SQL.Text := 'insert into [Report
Comments](RptID, CmbID) values('+reportID+','+inttostr(reportCommentNumber[i])+');';
        qryCreateReport_writereport.ExecSQL;
      end;
    end;
  end;
end;

//Runs the procedures, generating the full comment and writes it to the database
procedure TfrmCreateReport.GenerateReport;
begin
  if (isReportFrmValid)then
  begin
    reportCommentNumber[2] := behavior.outputAnswer + 7;
    OpeningComment;
    ClosingComment;
    strengthsAndWeakness;
    progress;
    showmessage('Report Generated');
    writeReport(lbledtsdtattendance.Text, lbledtsdtgrade.Text,
lbledtsdtpunctuality.Text, lbledtsdtsubject.Text, lblDisplayDate.Caption);
  end;
end;

//using the buttons previous, next/ generate
procedure TfrmCreateReport.btnQuestionClick(Sender: TObject);
begin
```

```
  if not(Tbutton(sender).Caption = 'Generate Report')then  //if not end of quiz
  begin
    if (Tbutton(sender).Name = 'btnNextQuestion')then //if button next is pressed
    begin
      if (matchQuestiontoVar(questionCount).outputAnswer = 0)then
        showmessage('Complete Question First') //make sure the question is answered
      else
      begin
        //store answer and go to next question
        selectAnAnswer(matchQuestiontoVar(questionCount).outputAnswer,false);
        questionCount := questionCount + 1;
        questionToDisplay(questionCount);
      end;
    end
    else
    begin
      //goes to the previous question
      selectAnAnswer(matchQuestiontoVar(questionCount).outputAnswer,false);
      questionCount := questionCount - 1;
      questionToDisplay(questionCount);
    end;
    //outputs the answer the user may have selected before
    if (matchQuestiontoVar(questionCount).outputAnswer <> 0)then
      selectAnAnswer(matchQuestiontoVar(questionCount).outputAnswer,true);
    if (questionCount > 1)then
    begin
      btnPreviousQuestion.Visible := true; //show option to go back since its not
the first question
      if (questionCount = 6)then
        btnNextquestion.Caption := 'Generate Report' //last question
      else
      begin
        btnNextquestion.Caption := 'Next Question';
      end
    end
    else
      btnPreviousQuestion.Visible := false; // if first question then hide the
option to go back
  end
  else
  begin
    GenerateReport; //if caption is generate report, then ...
  end;
end;

//writes the answer number selected
procedure TfrmCreateReport.rbAnswerClick(Sender: TObject);
var Index:integer;
begin
  index := strtoint(TRadioButton(sender).Name[9]);
  matchQuestiontoVar(questionCount).inputAnswer(index);
end;
end.
```

Highlighted Grey Code is changed in Prototype 2.

Validation carried out for "Create Report Form" (unit_CreateReport)

Validation of Selecting year box	When the user types a value which is invalid, it does not process it and changes the value of the combo box back to select year,

	otherwise it displays all the classes within the year selected in the class combo box.
Makes sure form is not empty	Makes sure no value is left empty before creating the report. Displays message if there is a value empty.
Validating Grade Value	Value of grade must only be a single alphabetical letter or a single digit, otherwise invalid and informs user and why.
Validating subject name value	Subject name must be one word, e.g. Computing, otherwise informs user that it's invalid and why.
Validating attendance and punctuality	Vale must be greater than/ equal to 0 but less than/ equal to 100, since it's a percentage. If invalid, it informs the user that either one of them are invalid and why.

What has been done

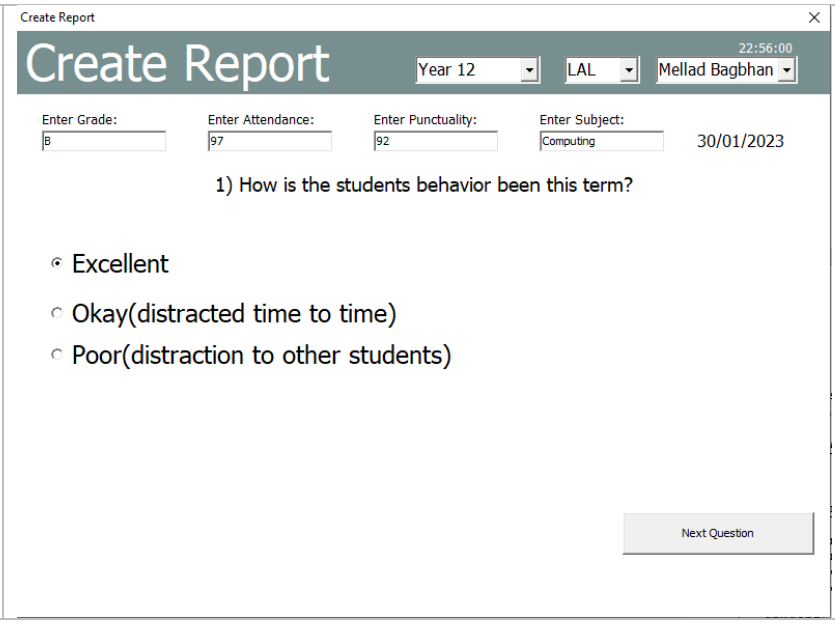
The Create Report form has been implemented. Criteria 9 has been met. It generates the student report by asking the user Six Multiple Choice questions. It allows the user to go back and forward to change any answers. After filling in the form and answering all six questions, it generates the comment and lets the user know the report has been created. Requirement 14.1 and 14.2 have been met. The Create Report form can be open from the main menu, hence requirements 8.3 has been partially met. The prototype was tested in runtime of the code. Screenshots of results of the tests are shown below.

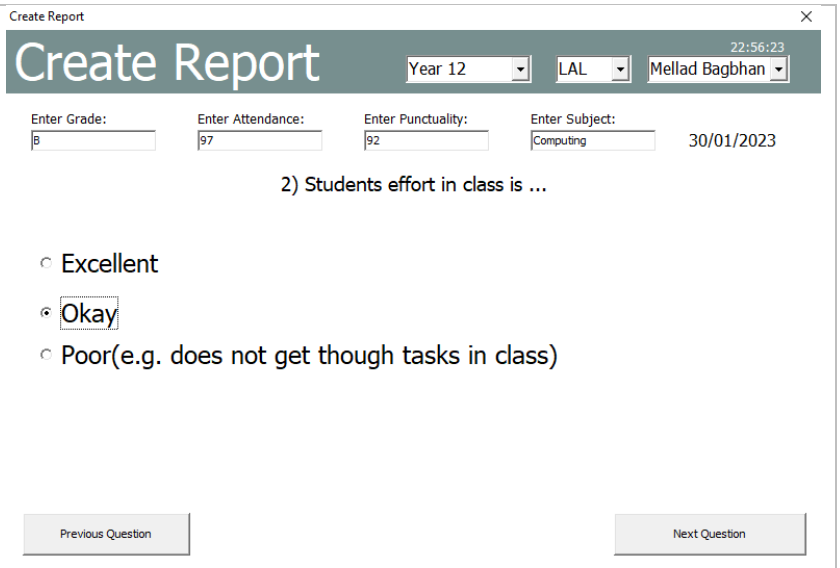
Summary: The user is able to create and sign in using an account. The user is able to update their details. They are also able to increment all years by one and delete unneeded data at the start of a new academic year by pressing a single button, within the update setting form. The main menu form has also been created, which will allow users to open different forms for the solution. Users are able to create classes and students within it. They can view all students within a class, delete the class along with the students and their data inside the class, and update classes. Users are now also able to create a report of an existing student.

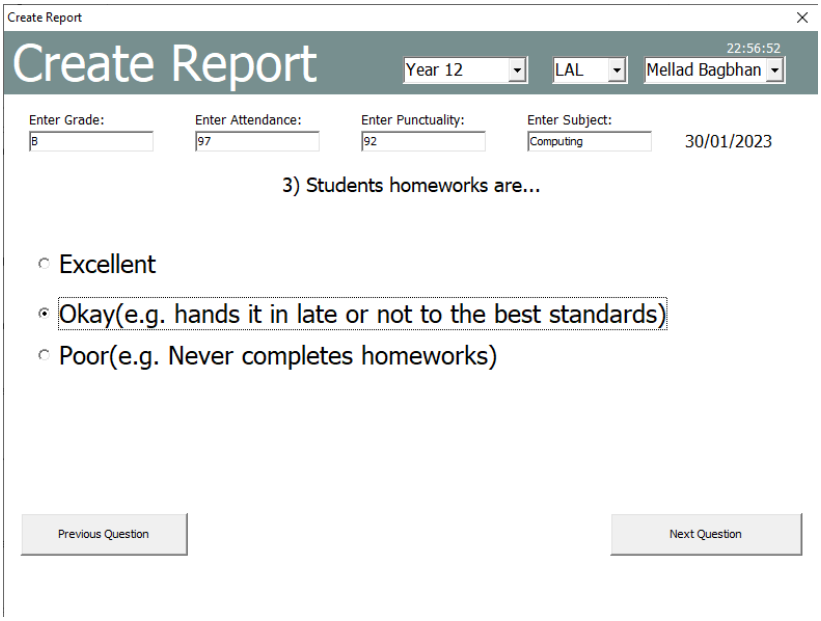
Testing of solution

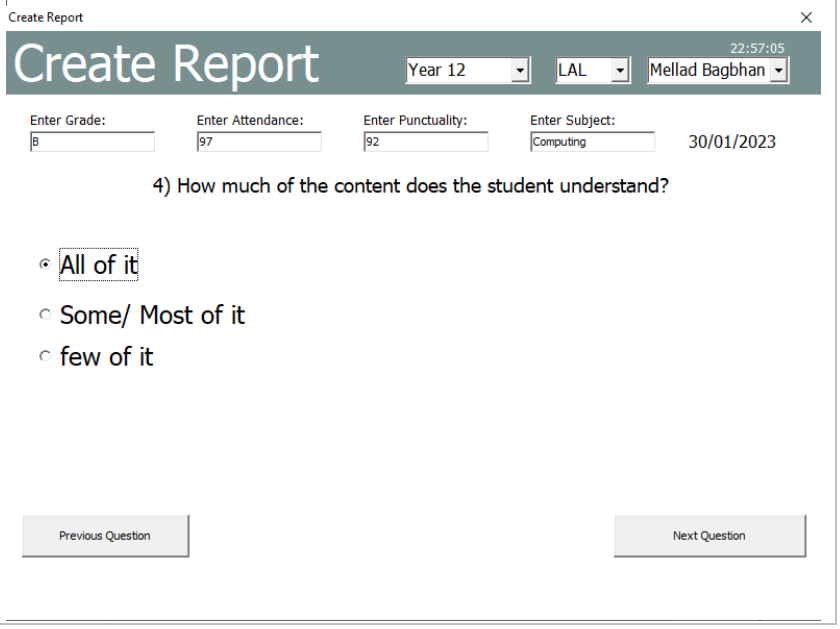
Test carried out are 8.1, 8.2, 8.3, 8.4, 8.5

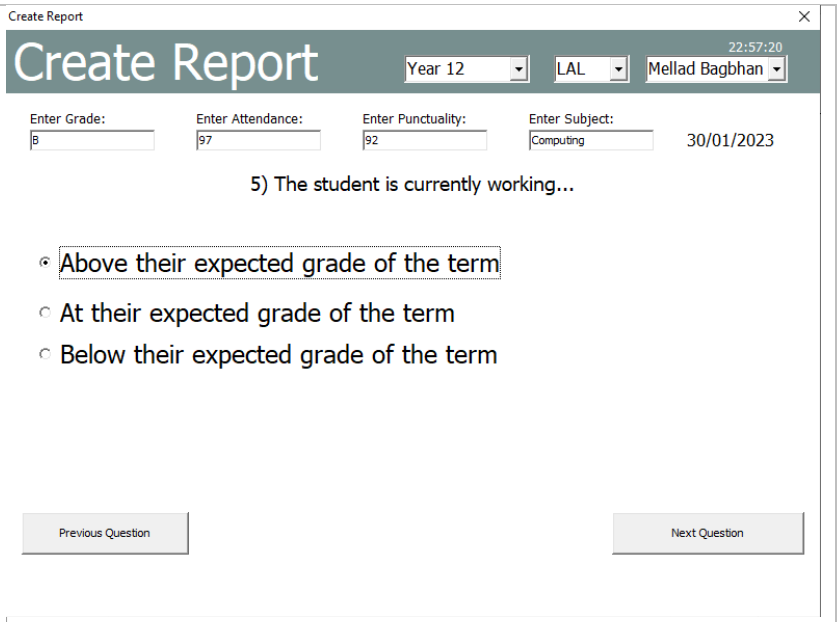
Test No 8.1 result: Checking if the form can generate a valid report …

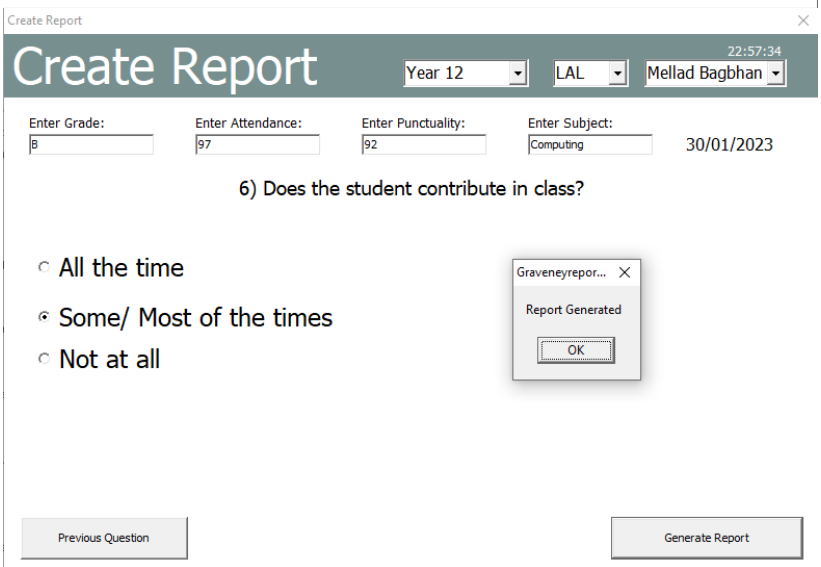
Inputting data	


Create Report
Create Report
22:56:23
Year 12
LAL
Mellad Bagbhan
Enter Grade:
B
Enter Attendance:
97
Enter Punctuality:
92
Enter Subject:
Computing
30/01/2023
2) Students effort in class is ...
Excellent
Okay
Poor(e.g. does not get though tasks in class)
Previous Question
Next Question

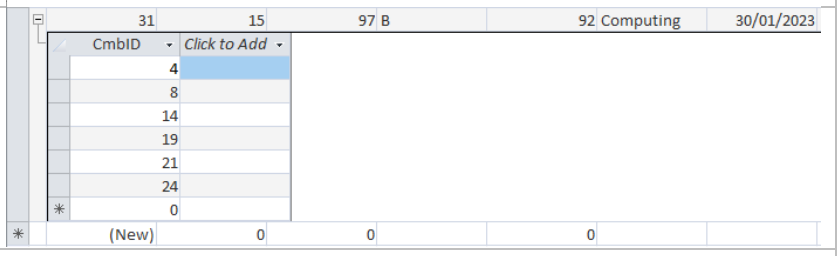

Create Report
Create Report
22:56:52
Year 12
LAL
Mellad Bagbhan
Enter Grade:
B
Enter Attendance:
97
Enter Punctuality:
92
Enter Subject:
Computing
30/01/2023
3) Students homeworks are...
Excellent
Okay(e.g. hands it in late or not to the best standards)
Poor(e.g. Never completes homeworks)
Previous Question
Next Question


Create Report
Create Report
22:57:05
Year 12
LAL
Mellad Bagbhan
Enter Grade:
B
Enter Attendance:
97
Enter Punctuality:
92
Enter Subject:
Computing
30/01/2023
4) How much of the content does the student understand?
All of it
Some/ Most of it
few of it
Previous Question
Next Question





Verifying program comment algorithm works



Summary: Test Successful – The form is able to generate a report with valid inputted values by the user. It is able to follow the algorithm of generating the comment successfully. The comment Ids translates to…

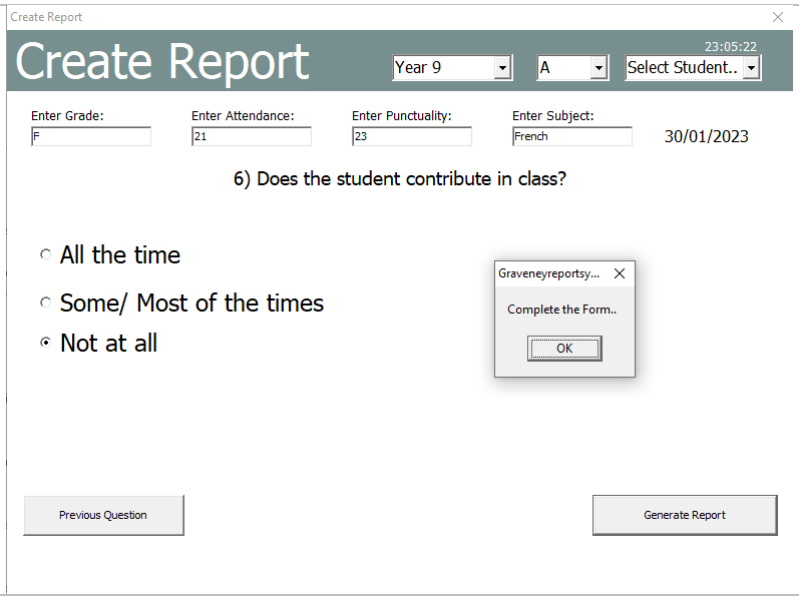
"Mellad has worked well this year but can be a bit reserved during discussion work, being willing to answer a question but rarely offering up ideas of her own. She has behaved in an appropriate manner, being a polite, helpful and well mannered student; a good role model to other students. The progress made by Mellad would be improved if she made a greater effort to complete homework tasks to the best of her ability and ensure that it was handed in by the set deadline. She quickly gets to grips with new concepts and is able to make links with other areas of the curriculum, showing higher order thinking. She has a tendency to hand in work after the deadline, better organisation is required. Overall Mellad has made a good level of progression this year and has met her target grades. I wish you well for the future Mellad."

Multiple additional tests were taken place to make sure the algorithm works properly…

⊞	3	10	98	A	100	English	30/11/2022
⊞	4	9	98	A	100	English	30/11/2022
⊞	5	9	98	C	100	Maths	30/11/2022
⊞	10	9	89	A	67	History	30/11/2022
⊞	11	9	56	D	45	Tecnology	30/11/2022
⊞	12	10	100	B	100	Computing	30/11/2022
⊞	17	9	100	A	100	Art	30/11/2022
⊞	18	10	100	A	100	Computing	30/11/2022
⊞	19	10	98	A	98	English	03/12/2022
⊞	20	10	100	A	12	Maths	03/12/2022
⊞	22	9	79	B	100	Science	03/12/2022
⊞	23	10	23	A	23	sd	03/12/2022
⊞	24	14	100	A	100	Maths	06/12/2022
⊞	25	10	12	A	100	English	13/12/2022
⊞	26	10	100	A	56	B	15/12/2022
⊞	27	10	23	A	12	Maths	16/12/2022
⊞	28	15	100	A	100	Computing	30/01/2023
⊞	29	16	90	A	100	ComputerScie	30/01/2023
⊞	30	15	97	B	92	Computing	30/01/2023

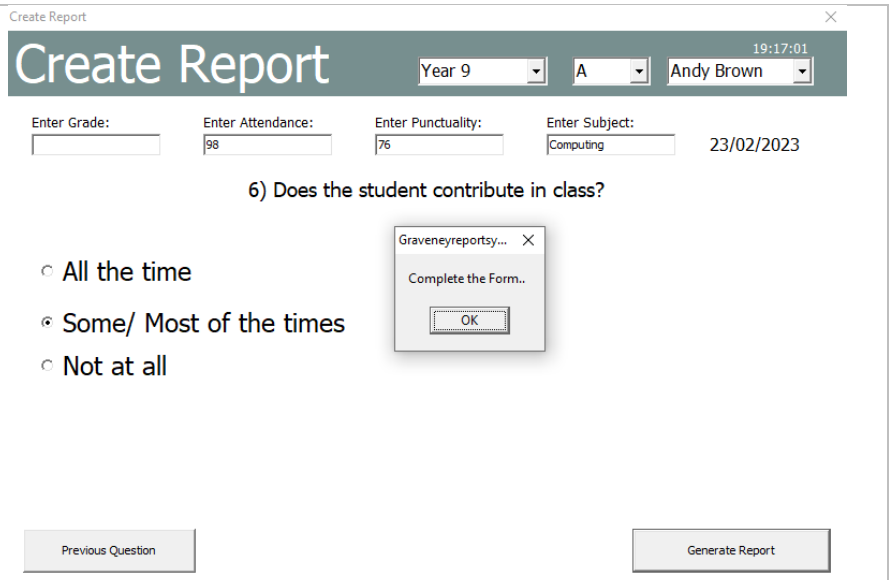
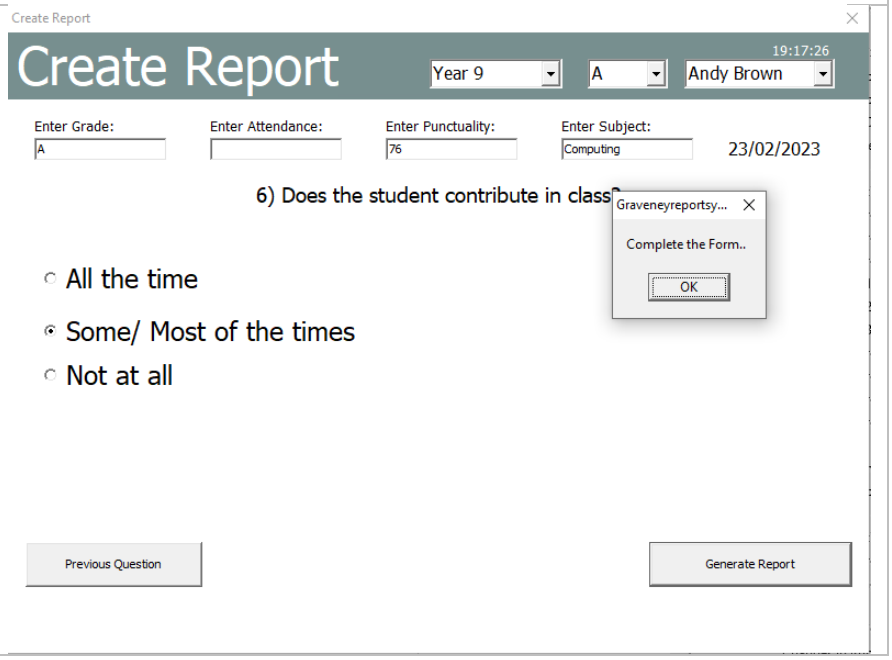
Test No 8.2.1 result: Checking if the form makes sure student is selected before creating report …

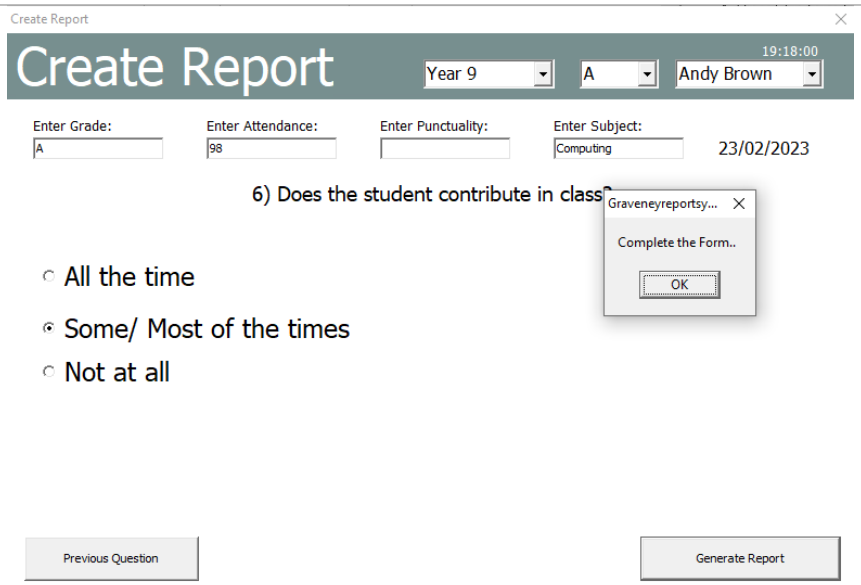
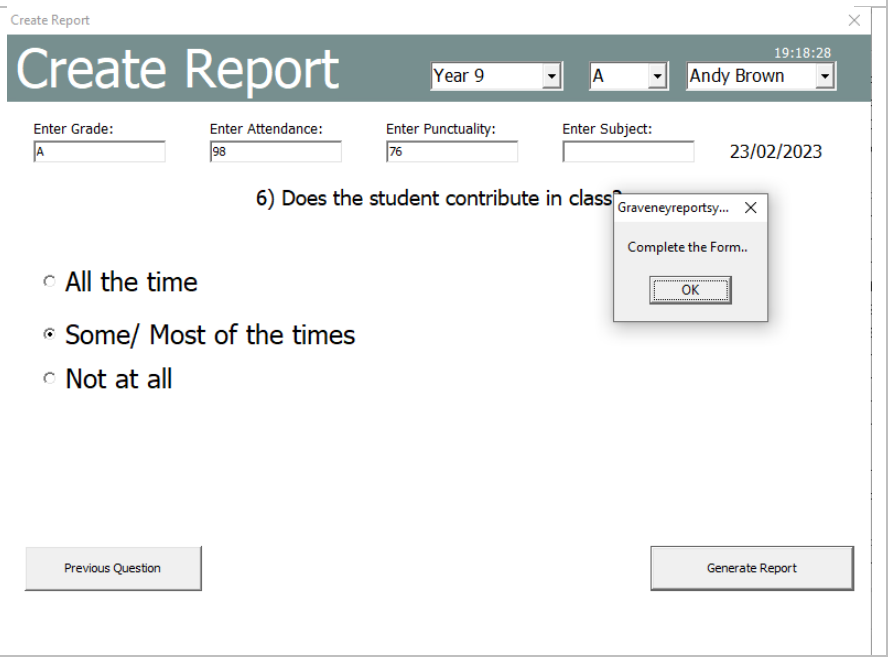
Checking Output when student is not selected

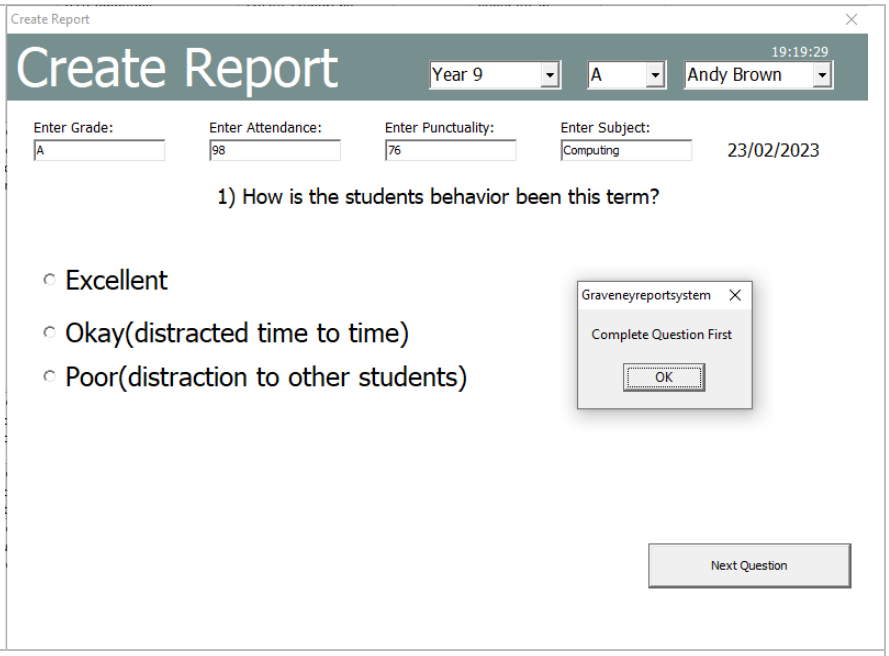


Summary: Test Successful – The form is able to check if the user has selected a student. The reason for this is to avoid SQL Errors and to prevent the report being created for a student the user created the report for previously.

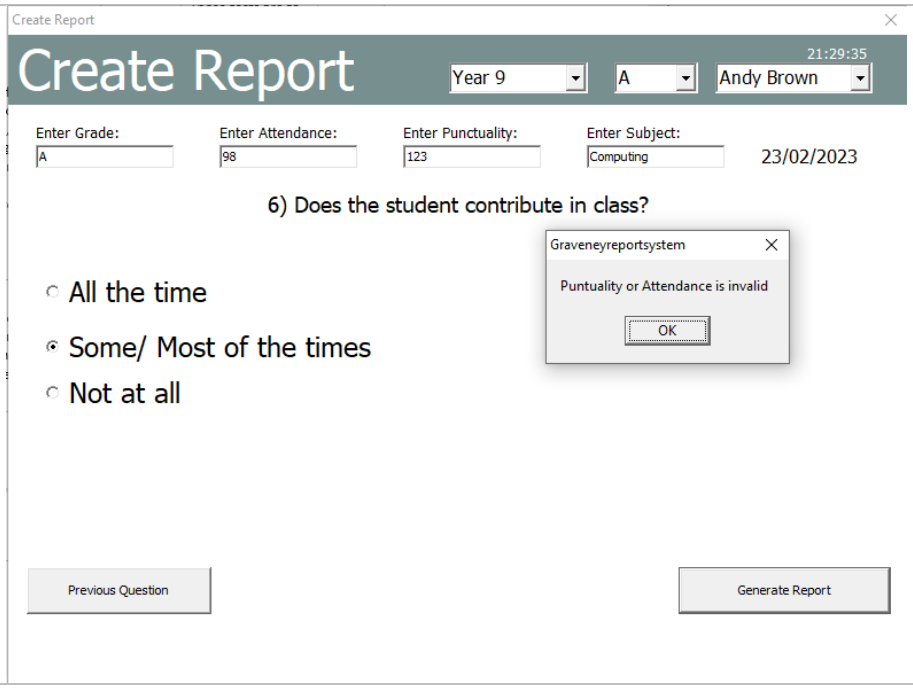
Test No 8.2.2 result: Checking if the form makes sure form is complete before creating report …

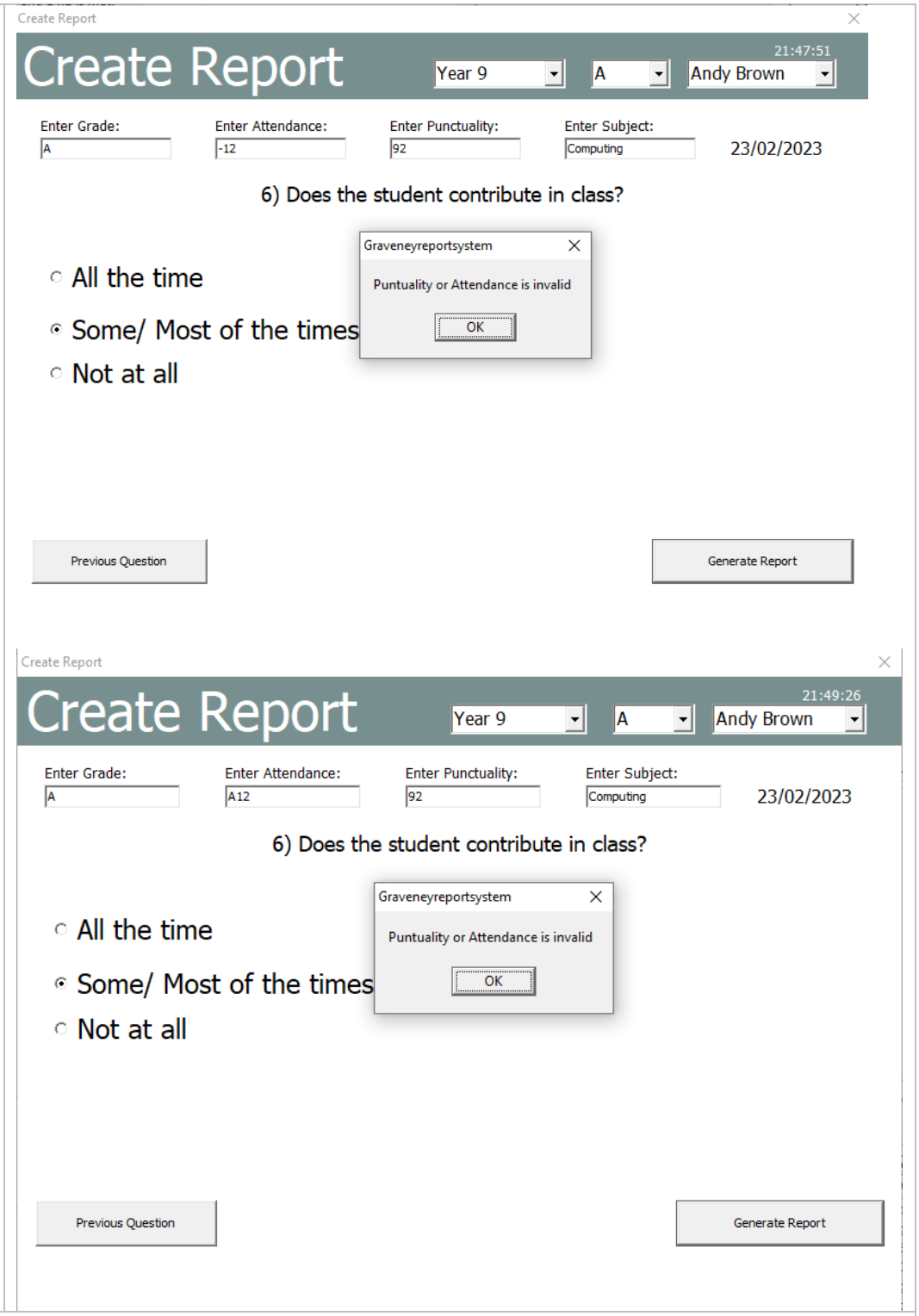
Inputting values except grade	
Inputting values except Attendance	

Inputting values except Punctuality	
Inputting values except Subject	

Inputting values except Answer to question	
Summary: Test Successful – The form is able to check if the user has completed the form before creating the report.	

Test No 8.3 result: Checking if the form is able to validate the attendance and punctuality …

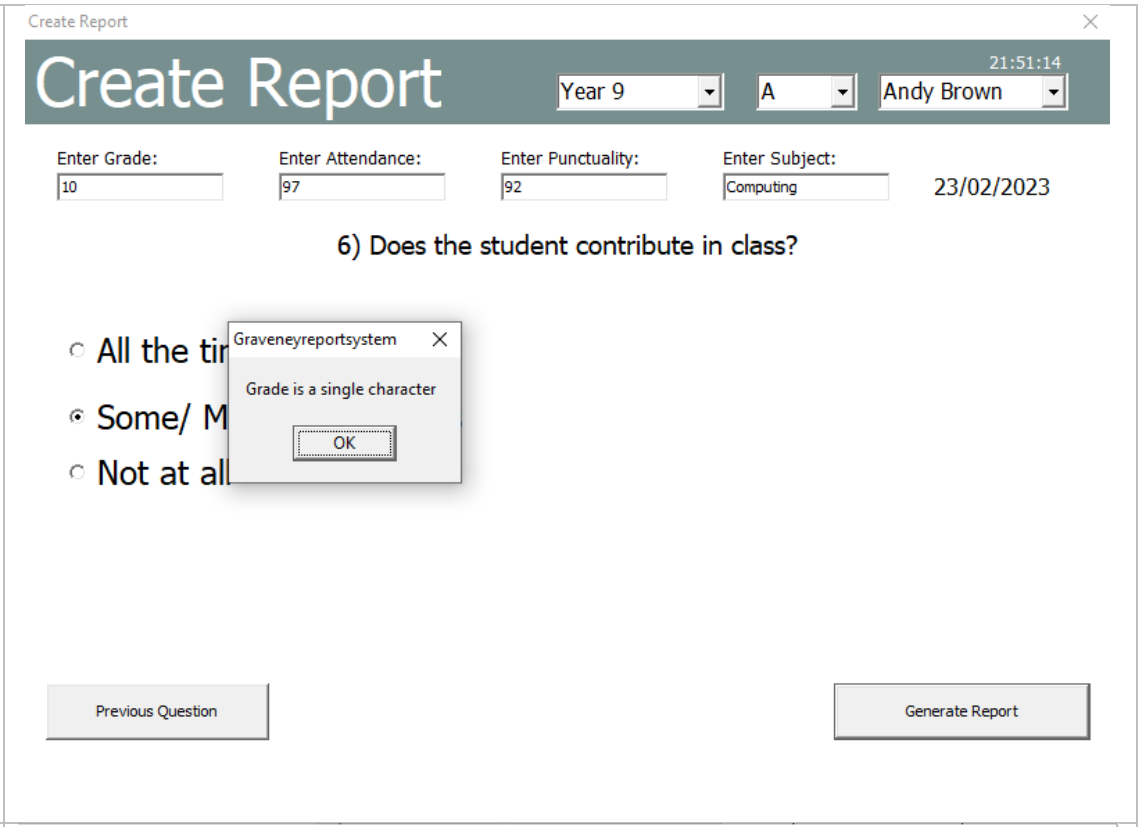
Inputting invalid value for punctuality	

Inputting invalid value for attendance	

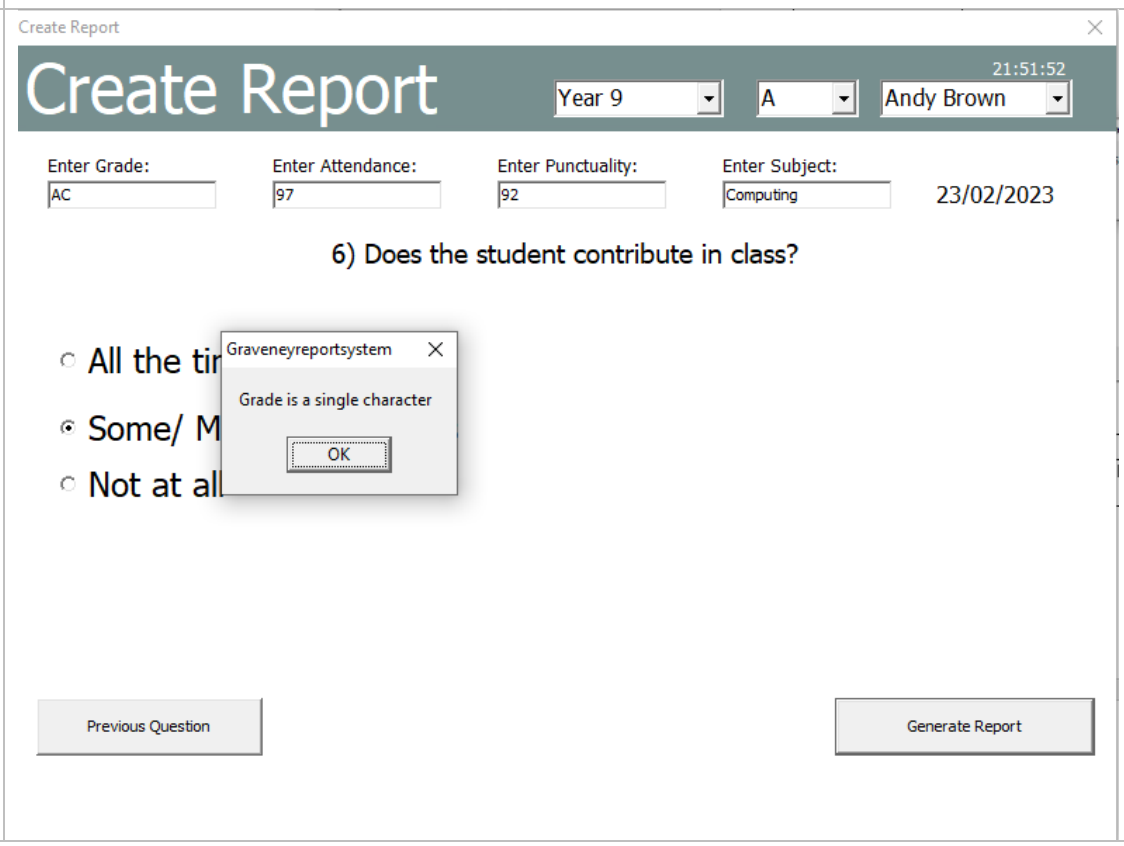
Summary: Test Successful – The form is able to check if the user has entered the attendance or punctuality with an invalid value, where the percentage is either greater than 100 or non-numerical, i.e. invalid data type

Test No 8.4 result: Checking if the form is able to validate the Grade entered …

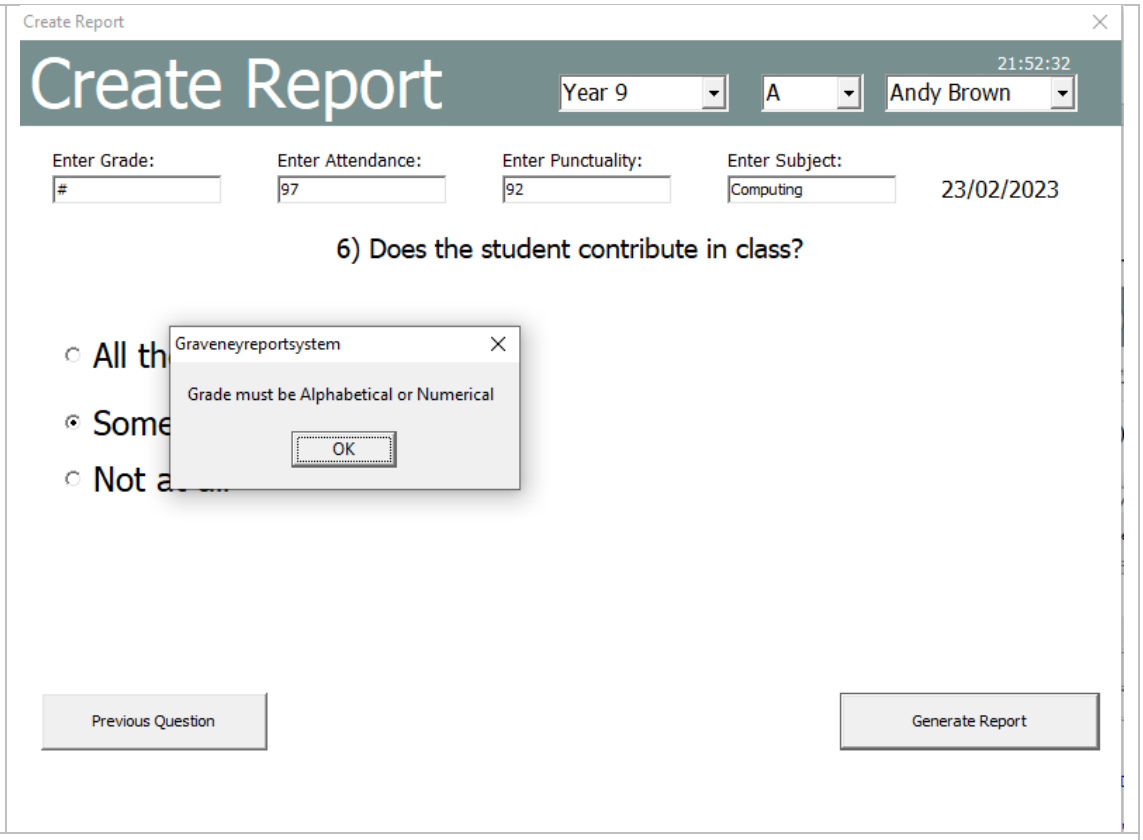
1)



2)



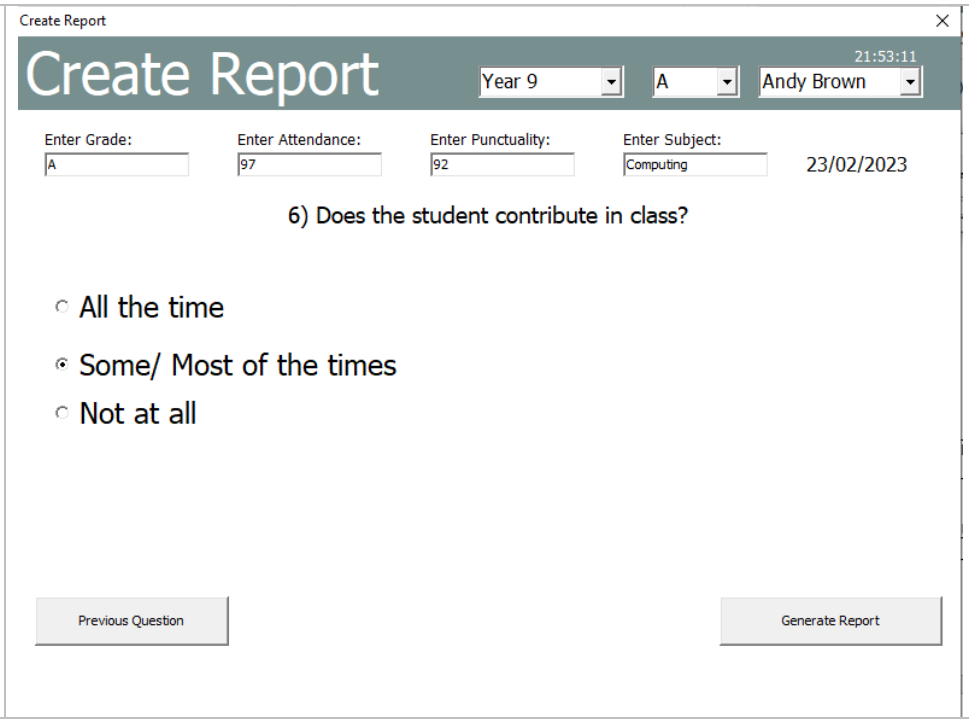
3)

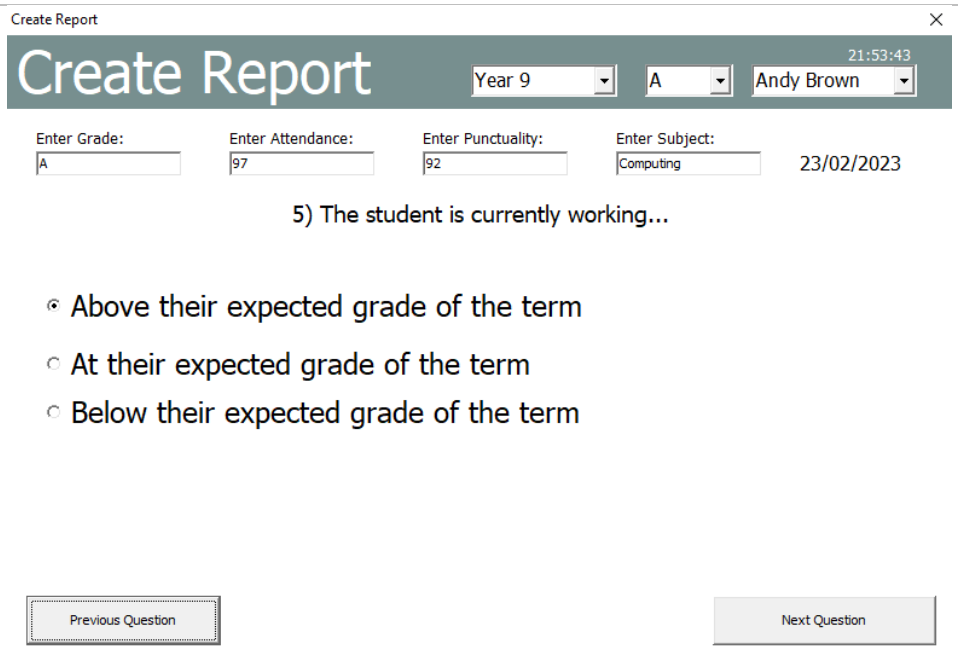
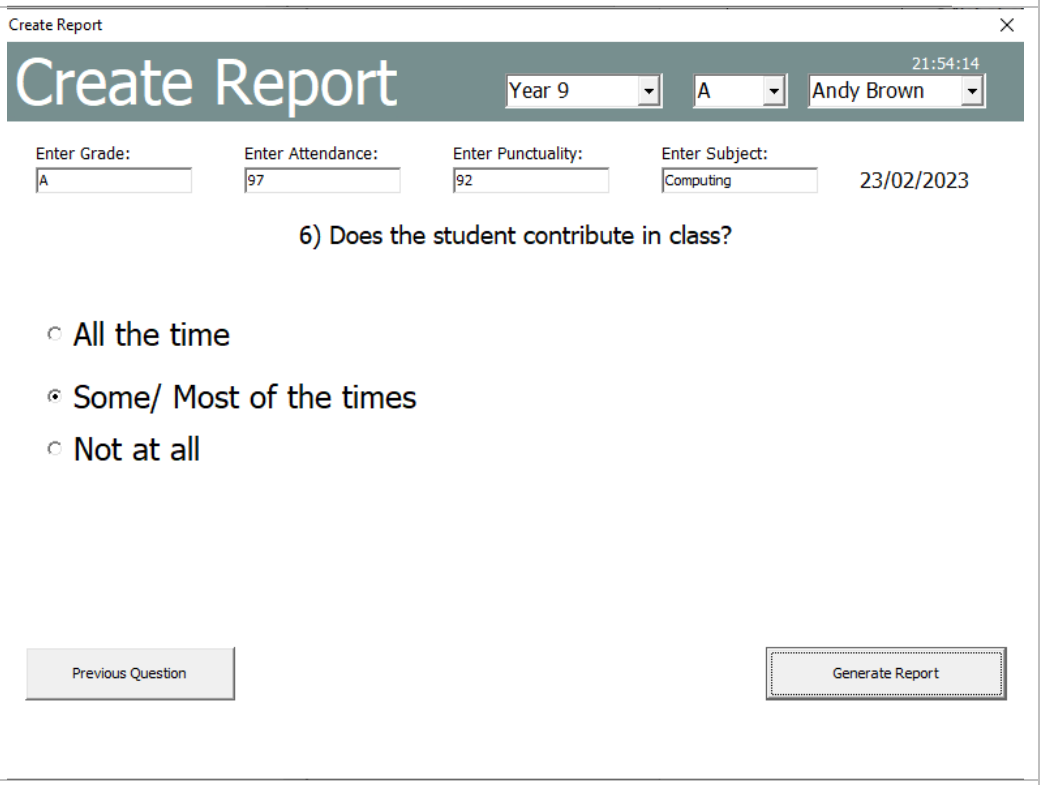


Summary: Test Successful – The form is able to check if the value of the grade is invalid on submission of the form. It makes sure the grade is a single character and either a integer or alphabet

Test No 8.5 result: Checking if the forms previous question button functions as intended to…

Before Pressing Button



After Pressing Button	
Press Next	
Summary: Test Successful – The form is able to display the previous question, with the previous answer selected, to let the user change their answer or view what they selected previously. It also stores any answers ahead of the current question it's at, if answered.	

Prototype 2

Prototype 2 aims to show the report once it's created. It also allows users to export the report as a PDF and copy the comment to clipboard. Prototype 2 also allows the user to input the grade A* which is a need informed by the stakeholders.

Changes in Implemented Code for Create Report Form (unit_CreateReport)

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, jpeg, ExtCtrls, StdCtrls, DB, ADODB, unit_login, unit_createStudent,
  unit_signup, unit_reportComment;
```

```
private
  reportCommentNumber:array[1..6] of integer; //comment ID stored to link to reports
  //1- opening, 2-behavior, 3-progress, 4-strength, 5-weakness, 6-closing comment
  procedure SelectAnAnswer(AnswerNum:integer;Bool:boolean);
  procedure GenerateReport;
public
  studentSelect:TStudentInfo; //Selected Student Info
  questionCount:integer; //used in main menu unit
  behavior, effortinClass, homework, understandingOfContent, WorkingTarget,
contribution:Tquiz; //used in main menu unit
  ReportID:string; //used to link comments to reports, used by unit_reportcomment
  procedure QuestionToDisplay(questionNum:integer); //used in main menu unit
  function matchQuestiontoVar(questionNum:integer):Tquiz;
end;
```

```
procedure TfrmCreateReport.GenerateReport;
begin
  if (isReportFrmValid)then
  begin
    reportCommentNumber[2] := behavior.outputAnswer + 7;
    OpeningComment;
    ClosingComment;
    strengthsAndWeakness;
    progress;
    showmessage('Report Generated');
    writeReport(lbledtsdtattendance.Text, lbledtsdtgrade.Text,
lbledtsdtpunctuality.Text, lbledtsdtsubject.Text, lblDisplayDate.Caption);
    frmReportComment.displayReport(reportID);
    frmReportComment.show;
    frmCreateReport.Close;
  end;
end;
```

Justification: Displays the report in the report comment form once created and closes the create report form.

Changes in Implemented Code for Create Report Form (unit_CreateReport)

<u>Before</u>

```
//informs user
showmessage('Grade is a single
character');
isInvalid := True;
```

<u>After</u>

```
// Allows grade A*
if not(lbledtSdtGrade.Text = 'A*')then
begin
  //informs user
  showmessage('Grade is a single
character');
  isInvalid := True;
end;
```

Justification: Allows user to input the grade A* in reports

Changes in Database table, Report

The report table field, Grade, valiation was set so the length must be one (2.5.1 Data Dictionary). Since the A* grade is a length of two, the new validation is that the length must be below two and greater than zero, i.e. either one or two.

Implemented Code for Report Comment Form (unit_ReportComment)

```
unit Unit_ReportComment;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, unit_login, DB, ADODB, unit_signup, ClipBrd,
unit_ExportReport;

type
  TfrmReportComment = class(TForm)
    qryfrmReportComment_Select: TADOQuery;
    MemoReportComment: TMemo; //Displays Report Comment
    lblSdtGrade: TLabel;
    btnCopyMemoReportComment: TButton;
    procedure btnCopyMemoReportCommentClick(Sender: TObject); //Copy the Report
Comment
    procedure btnExportClick(Sender: TObject); //Export the Report Comment
  published
    //Student Details
    btnExport: TButton;
    lblSdtName: TLabel;
    lblSdtAttendance: TLabel;
    lblSdtPunctuality: TLabel;
    lblSdtSubject: TLabel;
  private
    pronoun:array[1..3] of string; //Array of 3 items to store students pronouns
    procedure writePronouns(gender:string);
  public
    punctuality, attendance, grade, subject, studentName, dateCreated,
classNameYr:string; //Report Details
    procedure displayReport(reportID:string);
  end;

var
  frmReportComment: TfrmReportComment;

implementation

{$R *.dfm}

// Stores the Students pronouns according to their gender
procedure TfrmReportComment.writePronouns(gender:string);
begin
  if (gender = 'Male')then
  begin
    pronoun[1] := 'he';
    pronoun[2] := 'his';
    pronoun[3] := 'him';
  end
  else
  begin
    pronoun[1] := 'she';
    pronoun[2] := 'her';
```

```
    pronoun[3] := 'her';
  end;
end;

//Dislays the Report
procedure TfrmReportComment.displayReport(reportID:string);
var comment, gender, studentFirstName:string;
  i:integer;
begin
  //Query gets Student's, Report's, and Classes Details, Corresponding to the
Report ID
  qryfrmReportComment_select.SQL.Text := 'select
Student.FirstName,Student.LastName,'+
  'Student.Gender, Report.Attendance,Report.Grade, Report.Punctuality,'+
  ' Report.Subject, Report.DateCreated, Class.Name,Class.Year '+
  'from Class inner join (Student inner join Report on Student.SdtID =
Report.SdtID)'+
  ' on Class.classID = Student.classID where report.RptID = '+reportID+';';
  qryfrmReportComment_select.Open;
  punctuality := qryfrmReportComment_select.fieldbyname('punctuality').asstring;
  subject := qryfrmReportComment_select.fieldbyname('subject').asstring;
  attendance := qryfrmReportComment_select.fieldbyname('attendance').asstring;
  grade := qryfrmReportComment_select.fieldbyname('grade').asstring;
  studentName := qryfrmReportComment_select.fieldbyname('FirstName').asstring +
  ' ' + qryfrmReportComment_select.fieldbyname('LastName').asstring;
  dateCreated := qryfrmReportComment_select.fieldbyname('dateCreated').asstring;
  classNameYr := qryfrmReportComment_select.fieldbyname('year').AsString +
  qryfrmReportComment_select.fieldbyname('name').AsString;

  gender := qryfrmReportComment_select.fieldbyname('Gender').asstring;
  studentFirstName := qryfrmReportComment_select.fieldbyname('FirstName').asstring;

  //Displays Student Name, and Report's Details to the form
  lblsdtName.caption :=
qryfrmReportComment_select.fieldbyname('FirstName').asstring;
  lblSdtSubject.Caption := 'Subject: '+subject;
  lblSdtpunctuality.Caption := 'Punctuality: ' + punctuality;
  lblSdtAttendance.Caption := 'Attendance: ' + attendance;
  lblSdtgrade.Caption := 'Grade: ' + grade;

  //Query gets the Comments of the reports
  qryfrmReportComment_select.SQL.Text := 'select [Comment Bank].comment from
[Comment
  Bank] '+
  'inner join [Report Comments] on [Comment Bank].cmbID = [Report Comments].CmbID
  where rptID = '+reportID+' order by [Report Comments].CmbID;';
  qryfrmReportComment_select.open;

  //Stores the Comment to the variable comment
  for i:=1 to qryfrmReportComment_select.RecordCount do
  begin
    comment := comment + ' ' +
               qryfrmReportComment_select.FieldByName('Comment').AsString;
    qryfrmReportComment_select.next;
  end;

  writePronouns(gender); //Stores the correct pronouns in array pronoun,
correspoding to the students gender

  //Edits the comment to include pronouns and name, and have correct casing.
  comment := stringReplace(comment, '{pronoun1}', pronoun[1], [rfReplaceAll]);
```

```
  comment := stringReplace(comment, '{pronoun2}', pronoun[2], [rfReplaceAll]);
  comment := stringReplace(comment, '{pronoun3}', pronoun[3], [rfReplaceAll]);
  comment := stringReplace(comment, '{name}', StudentFirstName, [rfReplaceAll,
rfIgnoreCase]);
  comment := stringReplace(comment, '{Pronoun1}',
             frmsignup.FirstLetterUppercase(pronoun[1]), [rfReplaceAll]);
  comment := stringReplace(comment, '{Pronoun2}',
             frmsignup.FirstLetterUppercase(pronoun[2]), [rfReplaceAll]);
  comment := stringReplace(comment, '{Pronoun3}',
             frmsignup.FirstLetterUppercase(pronoun[3]), [rfReplaceAll]);
  memoReportComment.lines.Text := comment; //Displays the comment on the form
end;

procedure TfrmReportComment.btnCopyMemoReportCommentClick(Sender: TObject);
begin
  clipboard.AsText := memoReportComment.Lines.Text; //writes the comment to the
users clipboard
end;

//Allows user to Export report as PDF
procedure TfrmReportComment.btnExportClick(Sender: TObject);
begin
  frmExportReport.ShowModal; //Shows the Export Report Form
end;

end.
```

Implemented Code for Export Report Form (Unit_ExportReport)

```
unit Unit_ExportReport;

interface

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

type
  TfrmExportReport = class(TForm)
  published
    dirlstSave: TDirectoryListBox;
    lblLocation: TLabel;
    drvcbbSave: TDriveComboBox;
    lbledtFileName: TLabeledEdit;
    save: TButton;
    procedure saveClick(Sender: TObject);
end;
var
  frmExportReport: TfrmExportReport;

implementation

uses Unit_ReportComment;

{$R *.dfm}

procedure TfrmExportReport.saveClick(Sender: TObject);
  var pdf:TPdfDocumentGDI;
      page:TpdfPage;
      i:integer;
begin
```

```
  //create PDF with auther being the system
  pdf:=TpdfdocumentGDI.create();
  pdf.info.author:='GraveneyReportSystem';
  page:=pdf.addpage;
  page.PageLandscape := true; //make page landscape

  //Create a horizontal line from 80px below page
  pdf.VCLCanvas.MoveTo(0,80);
  pdf.VCLCanvas.LineTo(pdf.VCLCanvassize.cx,80);

  //IN HEADING...
  //..write Graveney school in the middle
  pdf.vclcanvas.font.Size := 45;
  pdf.VCLCanvas.TextOut((pdf.VCLCanvassize.cx - pdf.VCLCanvas.TextWidth('Graveney
School'))div 2,5,'Graveney School');

  //..write Student Name on Left
  pdf.vclcanvas.font.Size := 20;
  pdf.VCLCanvas.TextOut(5,10 ,'student: '+frmReportComment.studentName);
  pdf.VCLCanvas.TextOut(5,45 ,'Class: '+frmReportComment.classNameYr);

  //..write report info
  pdf.vclcanvas.font.Size := 15;
  pdf.VCLCanvas.TextOut(pdf.VCLCanvassize.cx -
(pdf.VCLCanvas.TextWidth('Attendance: ') + 50),5 ,'Attendance:
'+frmReportComment.attendance);
  pdf.VCLCanvas.TextOut(pdf.VCLCanvassize.cx -
(pdf.VCLCanvas.TextWidth('Attendance: ') + 50),30 ,'Punctuality:
'+frmReportComment.punctuality);
  pdf.VCLCanvas.TextOut(pdf.VCLCanvassize.cx -
(pdf.VCLCanvas.TextWidth('Attendance: ') + 50),55 ,'Grade:
'+frmReportComment.grade);

  //Below heading...
  //..write subject and teacher name
  pdf.VCLCanvas.Font.Size := 30;
  pdf.VCLCanvas.TextOut(15,100 ,frmReportComment.subject+': '+info.firstname+ '
'+info.lastName);

  //..write Report Comment
  pdf.vclcanvas.font.Size := 15;
  for i:=1 to frmreportcomment.memoreportcomment.Lines.Count do
  begin
     pdf.VCLCanvas.TextOut(25,140+i*20
,frmreportcomment.memoreportcomment.Lines.Strings[i-1]);
  end;

  //..write Date on bottom
  pdf.VCLCanvas.Font.Size := 20;
  pdf.VCLCanvas.TextOut(pdf.VCLCanvasSize.cx -
(pdf.VCLCanvas.TextWidth('DD/MM/YYYY')+10) , pdf.VCLCanvasSize.cy - 30,
frmReportComment.dateCreated);

  //save the file
  pdf.SaveToFile(dirlstSave.DirLabel.caption + lbledtFileName.text+'.pdf');
//stores file in the location select by the user using dirlstsave
  showmessage('saved'); //lets the user know if the file has been saved
  pdf.Free;
  frmExportReport.Close;
end;
```

```
end.
```

What has been done

The Report Comment form and Export Report form have been implemented. They display the report to the user and allow the user to export the report as a PDF or/and copy the comment. The Create Report form has been re-implemented, so that when the user creates a report, the system is able to display the result to the user by opening the Report Comment form, and allows the user to export the report as a PDF or/and copy the comment. Criteria 10 for Create Report form has been met. Requirement 15.3 has been met. The prototype was tested in runtime of the code. Screenshots of results of the tests are shown below.

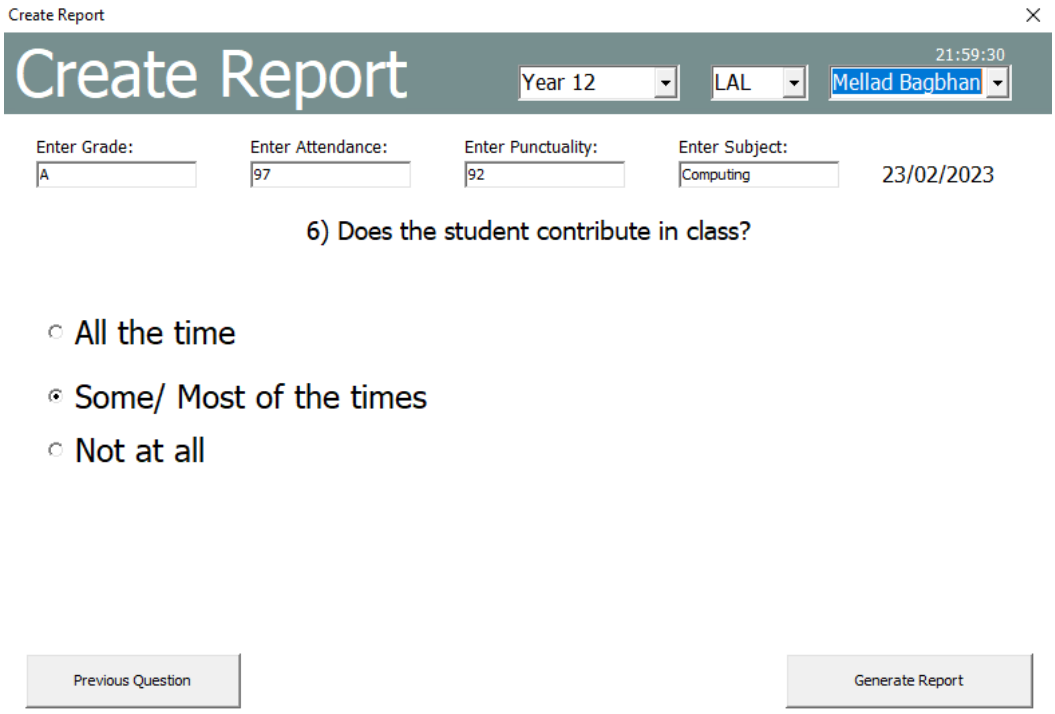
Summary: The user is able to create and sign in using an account. The user is able to update their details. They are also able to increment all years by one and delete unneeded data at the start of a new academic year by pressing a single button, within the update setting form. The main menu form has also been created, which will allow users to open different forms for the solution. The user is able to create classes and students within it. They can view all students within a class, delete the class along with the students and their data inside the class, and update classes. Users are able to create a report of an existing student, and view the report in a separate window. The user is able to export the report as a PDF or/and copy the comment.

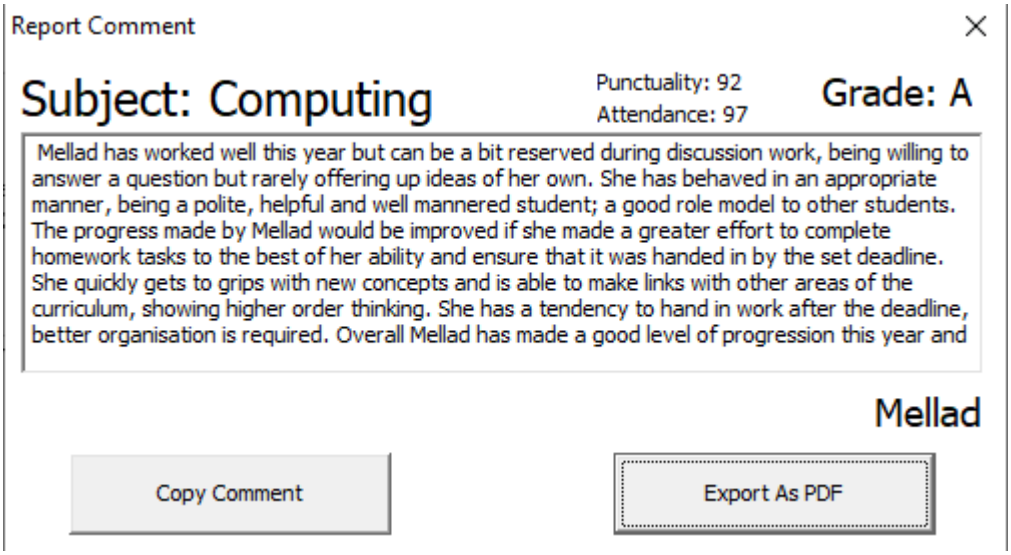
Testing of solution

Test carried out are 8.6, 8.7

Test No 8.6 result: Checking if the form can display the report correctly …

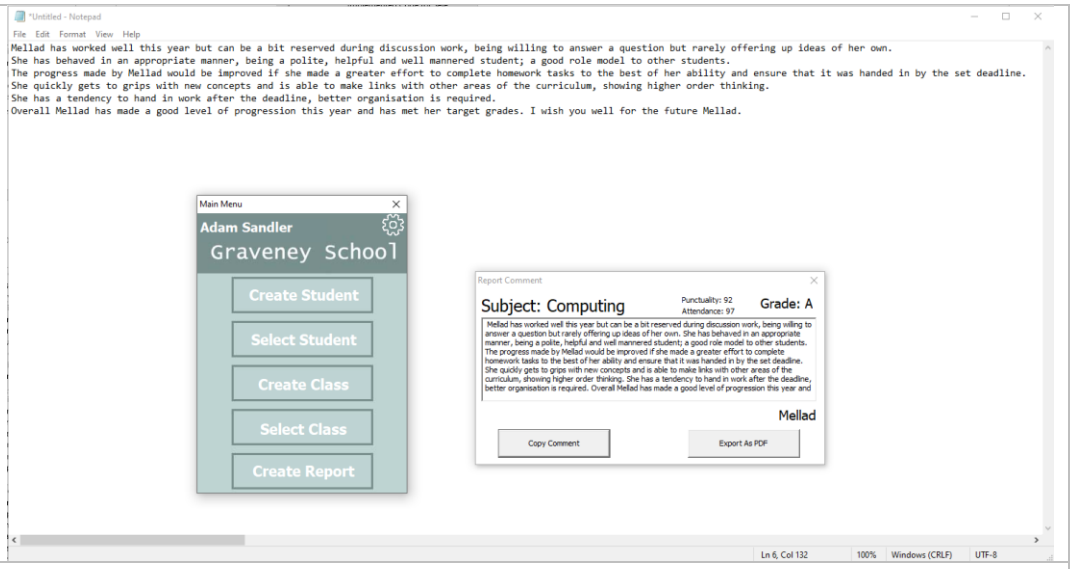
Creating a report...





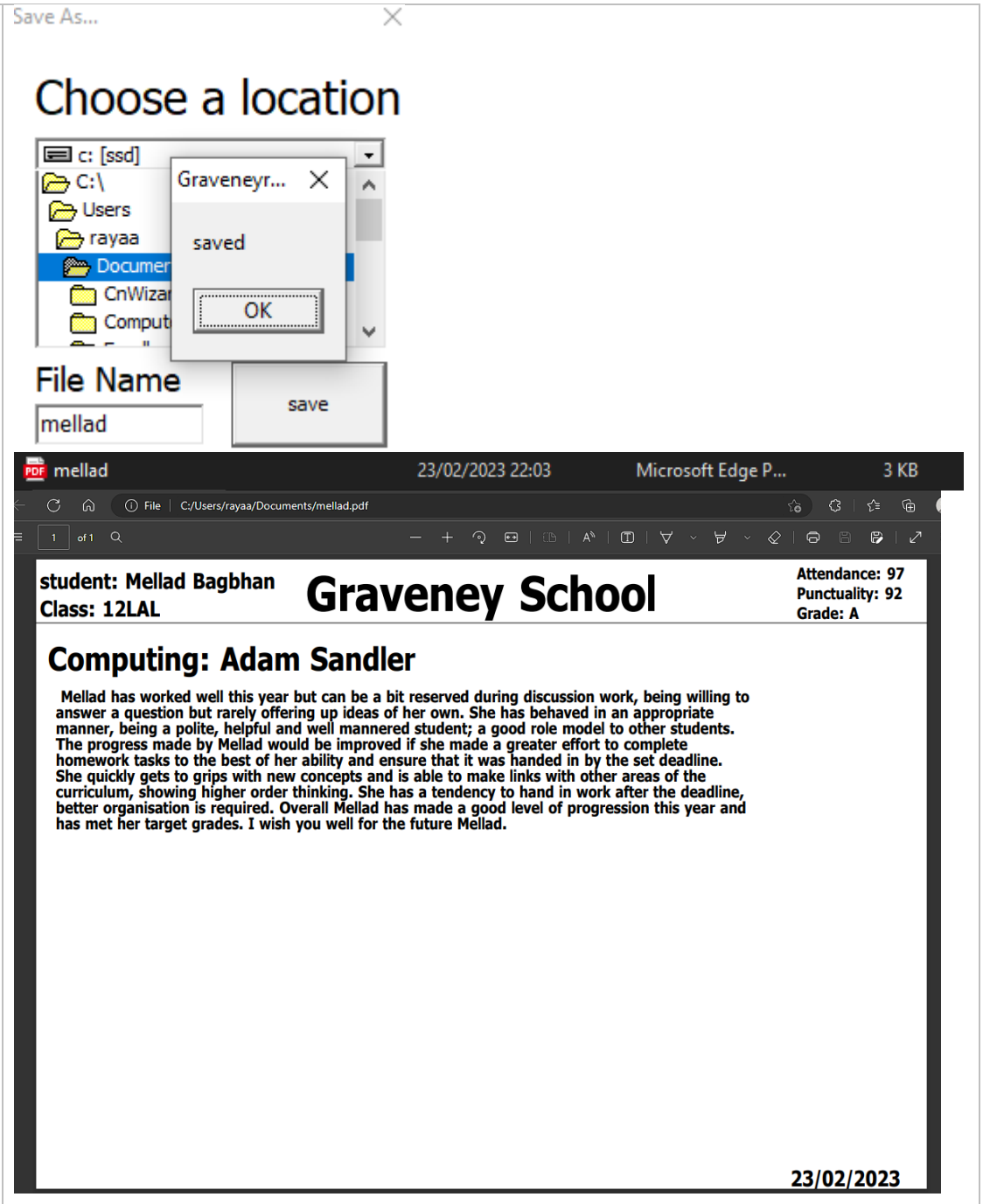
Summary: Test Successful – The form is able to output the report generated by the user.

Test No 8.7.1 result: Checking if the form allows the user to copy the comment …

Press copy	

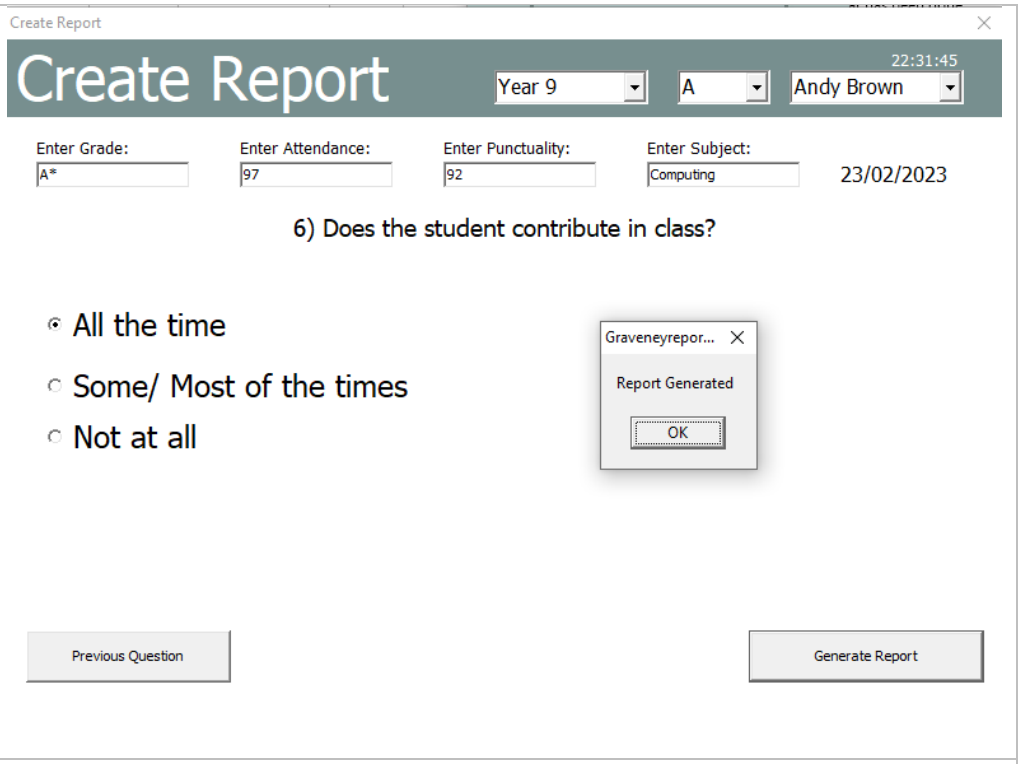
Summary: Test Successful – The form allows users to copy the report comment

Test No 8.7.2 result: Checking if the form allows the user to export the report as a pdf …

Press export and locate a location	

Summary: Test Successful – The form is allows users to export the report as a pdf

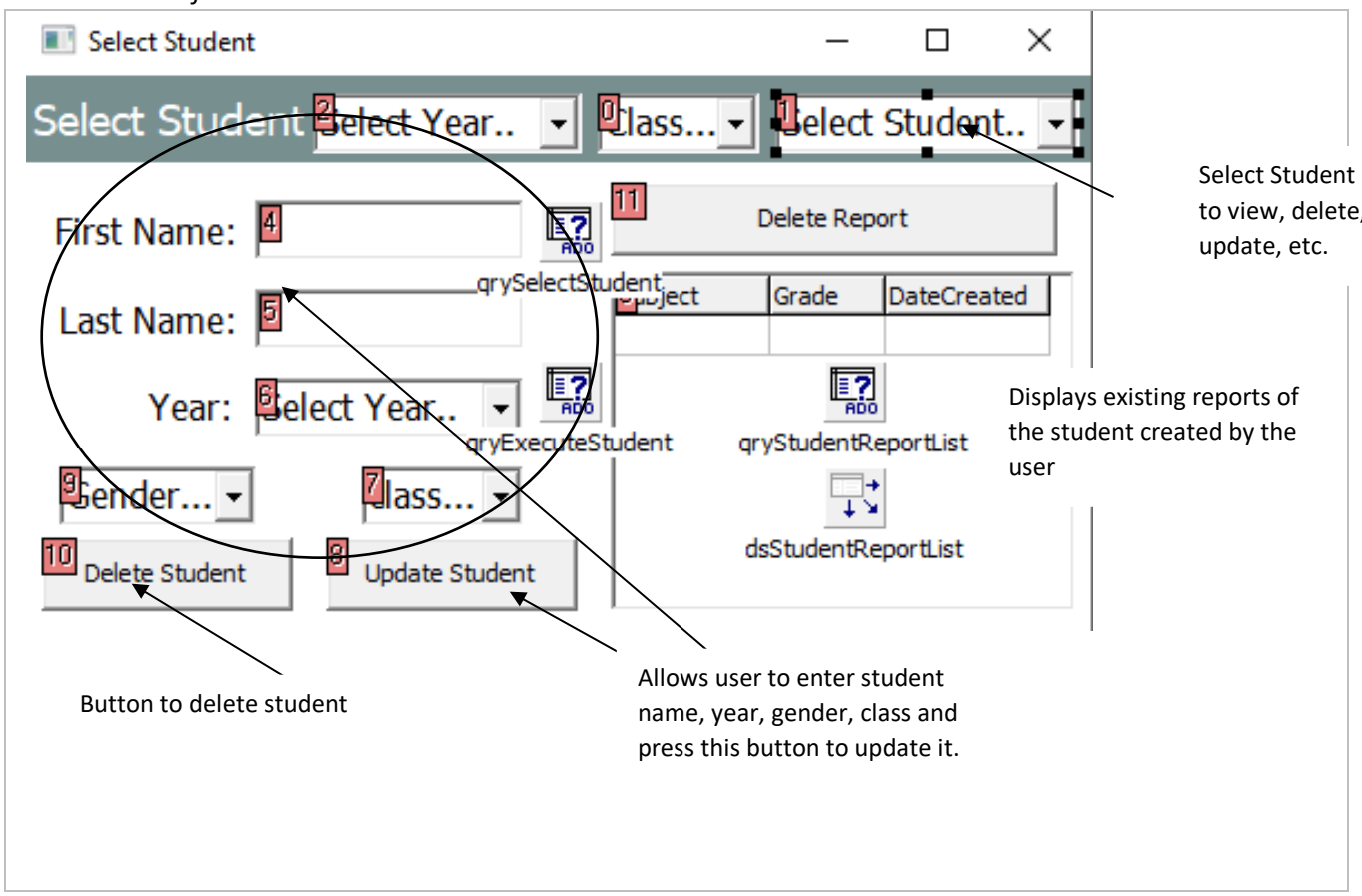
Test No 8.8 result: Checking if the form allows the user to input the grade A* …

Create report	
Summary: Test Successful – The form allows users input the grade A*	

Stage 9: Implementing the "Select Student Form"

Prototype 1

Screenshot of "Select Student Form"



Changes in Implemented Code for "Main Menu Form" (unit_menu)

Before	After

//Select student button - label component used as a button
procedure TfrmMainMenu.btnSelectStudentClick(Sender: TObject);
//open form
end;
	//Select student button - label component used as a button
	procedure TfrmMainMenu.btnSelectStudentClick(Sender : TObject);
	var i:integer;
	arr:yearArr;
	begin
	FreeAndNil(frmselectstudent);
	frmSelectStudent := TfrmSelectStudent.Create(frmselectstudent);
	// Displays years in combo box on select student form
	arr := selectyear(frmSelectStudent.qrySelectStudent, frmSelectStudent);
	frmSelectStudent.cboSelectyear.Clear;
	for i:=1 to length(arr) do
	begin
	if (arr[i]='') then
	break
	else

```
    begin

frmSelectStudent.cboSelectYear.items.Add(
arr[i]);

frmSelectStudent.cboChangeYear.items.Add(
arr[i]);
    end;
  end;

  //open form
  frmSelectStudent.ShowModal;
end;
```

Justification: Allow user to open the Select Student Form from main menu.

Changes in Implemented Code for Create Report Form (unit_CreateReport)

```
Public
  …
  procedure SelectClass(cbo, frm:Tobject; cName, cYear:string); //used in select
student
```

```
procedure TfrmCreateReport.SelectClass(cbo, frm:Tobject; cName, cYear:string);
var i:integer;
begin
  with TForm(frm) do
  begin
    TComboBox(cbo).Clear; //deletes any previous students in the combo box
    qryCreateReport_student.SQL.Text:='SELECT student.FirstName, student.LastName
FROM Class INNER JOIN Student'+
    ' ON Class.classID = Student.classID WHERE class.name='+quotedStr(cName)+' and
class.stfID = '+inttostr(info.staffID)+' and class.year =
'+frmcreatestudent.getClassYear(cYear)+' ORDER BY Student.firstname;';
    qrycreatereport_student.open;

    //adds the students to combo box in relation to the class selected
    for i:=1 to qryCreateReport_student.RecordCount do
    begin

TComboBox(cbo).Items.Add(qryCreateReport_student.fieldbyname('firstname').AsString+
' '+qryCreateReport_student.fieldbyname('lastname').AsString);
      qryCreateReport_student.Next;
    end;

    TComboBox(cbo).itemindex:=-1;
    if (qryCreateReport_student.RecordCount=0)then
      TComboBox(cbo).text:='No Student(s)' //when no students are in that class
    else
      TComboBox(cbo).text:='Select Student..'; //when there are students in the
class
  end;
  qryCreateReport_student.close;
end;

procedure TfrmCreateReport.cboSelectClassChange(Sender: TObject);
begin
  SelectClass(cboSelectStudent, frmCreateReport, cboSelectClass.Text,
cboSelectYear.text);
  studentSelect.resetSelectedStudent; //resets the values of student
end;
```

Justification: Making the algorithm of displaying students in a school class, public in unit create report. Its implemented into a public procedure to be used in select student unit.

Changes in Implemented Code for Create Report Form (unit_CreateReport)

```
…
Type arrStudentRecord = array[1..6] of string;

type
  TStudentInfo = class
  private
    fName, lName, year, gender, id, cName:string;
  public
    procedure writeSelectedStudent(selectedIndex:integer;selectedcName,
selectedYear:string); // writes the details of the selected student
    procedure resetSelectedStudent; //clears the data of the selected student
variable
    constructor Create; overload;
    function getData:arrStudentRecord;
  end;
```

```
procedure TStudentInfo.writeSelectedStudent(selectedIndex:integer;selectedcName,
selectedYear :string);
var i:integer;
begin
  with frmCreateReport.qrycreatereport_student do
  begin
    SQL.Text:='SELECT student.sdtID, student.FirstName, student.LastName,
student.Gender, class.year, class.name FROM Class INNER JOIN Student'+
    ' ON Class.classID = Student.classID WHERE
class.name='+quotedStr(selectedcName)+
    ' and class.stfID = '+inttostr(info.staffID)+' and class.year =
'+frmcreatestudent.getClassYear(selectedYear)+' ORDER BY Student.firstname;';
    open;
    for i:=1 to selectedIndex do
    begin
      next;
    end;
    fName := fieldbyname('firstname').asstring;
    lName := fieldbyname('lastname').asstring;
    year := fieldbyname('year').asstring;
    gender := fieldbyname('gender').asstring;
    id := fieldbyname('sdtID').asstring;
    cname := fieldbyname('name').asstring;
  end;
end;
```

```
procedure TfrmCreateReport.cboSelectStudentChange(Sender: TObject);
begin
  studentSelect.writeSelectedStudent(cboSelectStudent.ItemIndex,
cboSelectClass.Text, cboSelectYear.Text);
end;
```

```
function TstudentInfo.getData:arrStudentRecord;
begin
  getData[1] := id;
  getData[2] := fName;
  getData[3] := lName;
  getData[4] := gender;
  getData[5] := year;
  getData[6] := cname;
end;
```

Changes in Implemented Code for Select Class Form (unit_SelectClass)

```
Public
  …
  procedure DeleteStudentReports(studentID:string); //used in select student
```

```
//deletes students reports
procedure TfrmSelectClass.DeleteStudentReports(studentID:string);
var i,recCount:integer;
begin
  with frmSelectClass do
  begin
    qrySelectReports.SQL.text:='Select RptID FROM report where
SdtID='+StudentID+';';
    qrySelectReports.open;
    recCount:=qrySelectReports.recordCount;
    for i:=1 to recCount do
    begin
      qryDelete.SQL.text:='DELETE * FROM [report Comments] where
RptID='+qrySelectReports.fieldbyname('RptID').asstring+';';
      qryDelete.ExecSQL;
      qrySelectReports.Next;
    end;
    qryDelete.SQL.text:='DELETE * FROM report where SdtID='+StudentID+';';
    qryDelete.ExecSQL;
  end;
end;
```

Implemented Code for Create Report Form (unit_SelectStudent)

```
unit Unit_SelectStudent;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, Grids, DBGrids, jpeg, unit_login,
unit_createStudent, Unit_CreateReport,
  Unit_ReportComment, unit_signup, Unit_SelectClass, DB, ADODB;

type
  TfrmSelectStudent = class(TForm)
    ImgBackground: TImage;
    cboSelectClass: TComboBox;
    cboSelectStudent: TComboBox;
    cboSelectYear: TComboBox;
    dbgrdUserClassData: TDBGrid;
    lblYear: TLabel;
    lbledtStudent_FirstName: TLabeledEdit;
    lbledtStudent_LastName: TLabeledEdit;
    cboChangeYear: TComboBox;
    cboChangeClass: TComboBox;
    btnUpdateStudent: TButton;
    cboSelectGender: TComboBox;
    btnDeleteStudent: TButton;
    btnDeleteReport: TButton;
    lblSelectStudent: TLabel;
    dsStudentReportList: TDataSource;
    qryStudentReportList: TADOQuery;
    qrySelectStudent: TADOQuery;
    qryExecuteStudent: TADOQuery;
```

```
    procedure cboSelectClassChange(Sender: TObject);
    procedure cboSelectYearChange(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure cboSelectStudentChange(Sender: TObject);
    procedure cboChangeYearChange(Sender: TObject);
    procedure btnUpdateStudentClick(Sender: TObject);
    procedure btnDeleteReportClick(Sender: TObject);
    procedure btnDeleteStudentClick(Sender: TObject);
    procedure dbgrdUserClassDataDblClick(Sender: TObject);
  private
    studentSelect:TStudentInfo;
  public

  end;

var
  frmSelectStudent: TfrmSelectStudent;

implementation

{$R *.dfm}

procedure TfrmSelectStudent.cboSelectClassChange(Sender: TObject);
begin
  frmCreateReport.SelectClass(cboSelectStudent, frmSelectStudent,
cboSelectClass.Text, cboSelectYear.Text);
  studentSelect.resetSelectedStudent;
end;

procedure TfrmSelectStudent.cboSelectYearChange(Sender: TObject);
begin
  cboSelectStudent.Clear; //deletes any previous students in the combo box
  cboSelectClass.Clear; //deletes any previous classes in the combo box
  try

frmcreatestudent.selectClasses(frmSelectStudent.cboSelectClass,frmSelectStudent.cbo
selectYear); // finds the classes in relation to it
  except //if user inputs invalid value, it resets the combo box
    cboselectyear.Text:='Select Year...';
    cboselectclass.Clear;
  end;
  //resets all values
  cboselectstudent.clear;
  cboselectstudent.itemindex:=-1;
  cboselectstudent.text:='Select Student..';
  cboselectclass.itemindex:=-1;
  cboselectclass.text:='Class...';
  studentSelect.resetSelectedStudent;
end;

procedure TfrmSelectStudent.FormCreate(Sender: TObject);
begin
  studentSelect := TStudentInfo.Create;
end;

procedure TfrmSelectStudent.cboSelectStudentChange(Sender: TObject);
begin
  //write all student details to form
  studentSelect.writeSelectedStudent(cboSelectStudent.ItemIndex,
cboSelectClass.Text, cboSelectYear.Text);
  lbledtStudent_FirstName.Text := studentSelect.getdata[2];
```

```
  lbledtStudent_LastName.Text := studentSelect.getdata[3];
  cboSelectGender.ItemIndex :=
cboSelectGender.Items.IndexOf(studentSelect.getData[4]);
  cboChangeYear.text := studentSelect.getData[5];

frmcreatestudent.selectClasses(frmSelectStudent.cbochangeClass,frmSelectStudent.cbo
changeYear); // finds the classes in relation to it
  cboChangeClass.ItemIndex :=
cboChangeClass.Items.IndexOf(studentSelect.getdata[6]);

  //Display students reports
  with qryStudentReportList do
  begin
    SQL.Text := 'select RptID, Subject, DateCreated, Grade from Report where sdtID
= ' + studentSelect.getdata[1] + ';';
    Open;
  end;
end;

procedure TfrmSelectStudent.cboChangeYearChange(Sender: TObject);
begin

  try

frmcreatestudent.selectClasses(frmSelectStudent.cboChangeClass,frmSelectStudent.cbo
ChangeYear); // finds the classes in relation to it
  except //if user inputs invalid value, it resets the combo box
    cboChangeyear.Text:='Select Year...';
    cboChangeclass.Clear;
  end;
  //reset class values
  cboChangeclass.itemindex:=-1;
  cboChangeclass.text:='Class...';
end;

procedure TfrmSelectStudent.btnUpdateStudentClick(Sender: TObject);
var ClassID:string;
begin
  // Make sure student is selected first
  if not((cboSelectStudent.ItemIndex = -1) or (cboChangeClass.ItemIndex = -1) or
(lbledtStudent_FirstName.text = '') or
  (lbledtStudent_LastName.text = '') or (cboSelectGender.ItemIndex = -1))then
  begin
    if
((frmSignup.stringisAlpha(lbledtStudent_FirstName.Text))and(frmSignup.stringisAlpha
(lbledtStudent_LastName.Text)))then
    begin
      with qrySelectStudent do
      begin
        SQL.Text := 'select classID from class where Name =
'+quotedStr(cbochangeclass.text)+' and Year =
'+frmCreateStudent.getclassyear(cboChangeYear.Text)+
        ' and stfID = '+inttostr(info.StaffID)+';';
        Open;
        ClassID := FieldByName('classID').AsString;
        Close;
        sql.Text := 'select sdtid from student where FirstName =
'+quotedStr(lbledtStudent_FirstName.text)+' and LastName =
'+quotedStr(lbledtStudent_LastName.text)+
        ' and Gender = '+quotedStr(cboSelectGender.Text)+' and classID =
'+classID+';';
```

```
        Open;
        // checks if there is already a student, which exists, with the values the
student being updated with.
        if (recordcount > 0)then
          showmessage('Student with same identity already exists in that class...')
        else
        begin
          Close;
          // Updates the student
          SQL.Text := 'update Student set FirstName =
'+quotedStr(lbledtStudent_FirstName.text)+', LastName =
'+quotedStr(lbledtStudent_LastName.text)+
          ', Gender = '+quotedStr(cboSelectGender.Text)+', classID = '+classID+'
where SdtID = '+studentselect.getData[1]+';';
          ExecSQL;
          showmessage('Student updated...');
        end;
      end;
    end
    else
      ShowMessage('Invalid Name');
  end
  else
    ShowMessage('Complete the form...');
end;

procedure TfrmSelectStudent.btnDeleteReportClick(Sender: TObject);
begin
  try
    // attempts to delete selected report
    with qryExecuteStudent do
    begin
      sql.Text := 'delete from [Report Comments] where rptid
='+vartostr(qryStudentReportList['RptID'])+';';
      execsql;
      sql.Text := 'delete from report where rptid
='+vartostr(qryStudentReportList['RptID'])+';';
      execsql;
    end;
    showmessage('Report deleted!');

    // Reloads the Report Table
    with qryStudentReportList do
    begin
      SQL.Text := 'select RptID, Subject, DateCreated, Grade from Report where
sdtID = ' + studentSelect.getdata[1] + ';';
      Open;
    end;
  except
    // if report wasn't selected, traps the error and outputs a message
    ShowMessage('Select report first...');
  end;
end;

procedure TfrmSelectStudent.btnDeleteStudentClick(Sender: TObject);
var i:integer;
begin
  //attempts to delete student if selected along with all its data
  Try
    // Deletes all reports
    frmSelectClass.DeleteStudentReports(studentSelect.getdata[1]);
```

```
    // Deletes student
    qryExecuteStudent.sql.Text := 'delete from Student where sdtid
='+studentSelect.getdata[1]+';';
    qryExecuteStudent.execsql;
    showmessage('Student deleted!');
    frmSelectStudent.Close;
  except
    // If student is not selected
    ShowMessage('Select a student...');
  end;
end;

// Opens the report
procedure TfrmSelectStudent.dbgrdUserClassDataDblClick(Sender: TObject);
begin
  frmReportComment.displayReport(vartostr(qryStudentReportList['RptID']));
  frmReportComment.show;
end;

end.
```

Validation carried out for "Select Student Form" ()

Validates name when updating	Uses the same algorithm, in signup unit) used throughout the system to validate the students name, where the name must consist of only alphabets
Checks if the student being updated to already exists	Makes sure the user is not updating the student's data to an existing student's data and creating duplicates.

What has been done

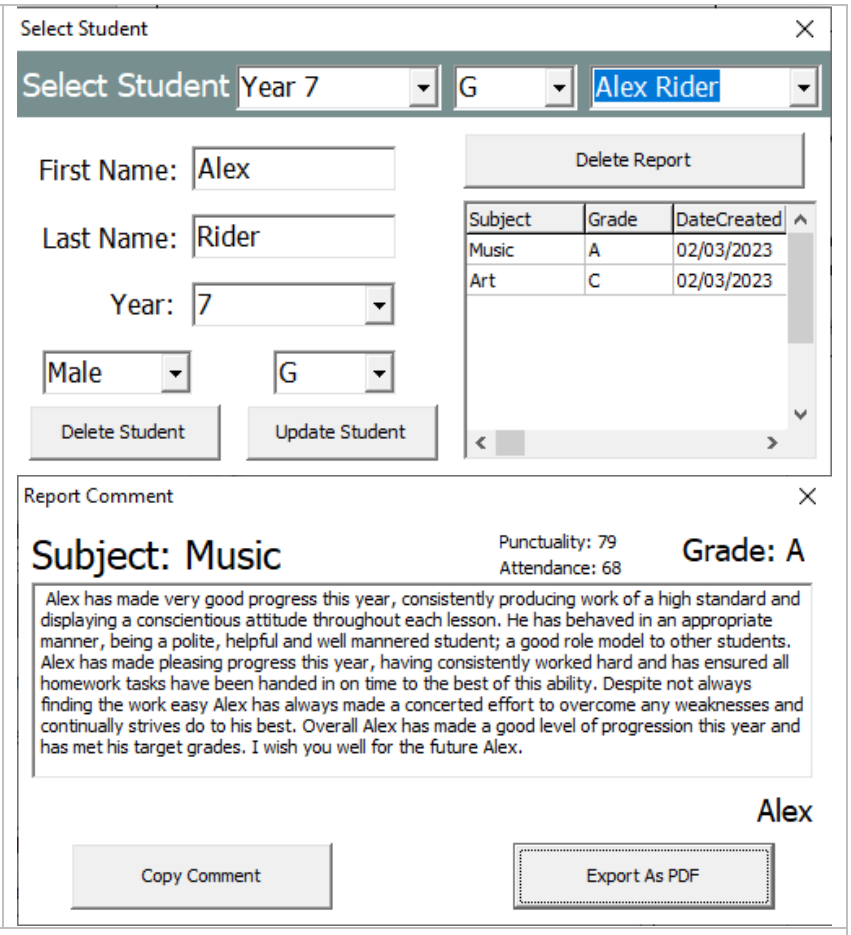
The Select Student form has been implemented. Criteria 11 have been met. Criteria 8 has partially been met. It is able to display existing reports of an individual, and allow the user to edit the student's details. It also allows users to delete students or delete their report(s). Requirements 11, 15 and 8.3 has been met.

Summary: The user is able to create and sign in using an account. The user is able to update their details. They are also able to increment all years by one and delete unneeded data at the start of a new academic year by pressing a single button, within the update setting form. The main menu form has also been created, which will allow users to open different forms for the solution. The user is able to create classes and students within it. They can view all students within a class, delete the class along with the students and their data inside the class, and update classes. Users are able to create a report of an existing student, and view the report in a separate window. The user is able to export the report as a PDF or/ and copy the comment. The user is able to view and/ or delete student's reports from past, and update and/ or delete students

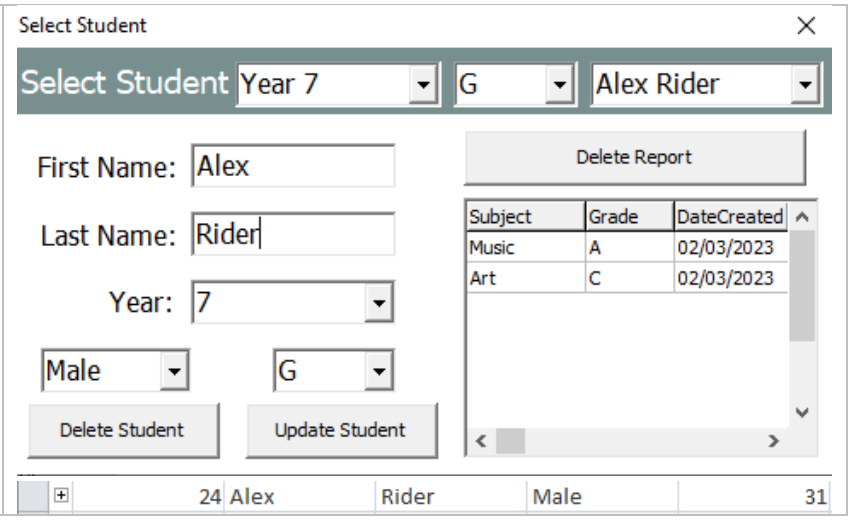
Testing of solution

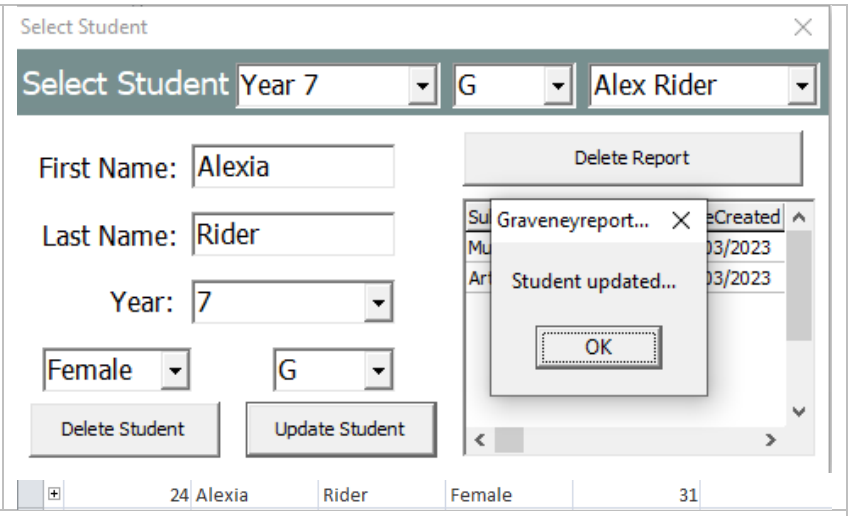
Test carried out are 9.1, 9.2, 9.3, 9.4

Test No 9.1 result: Checking if the form allows user to view and open student reports …

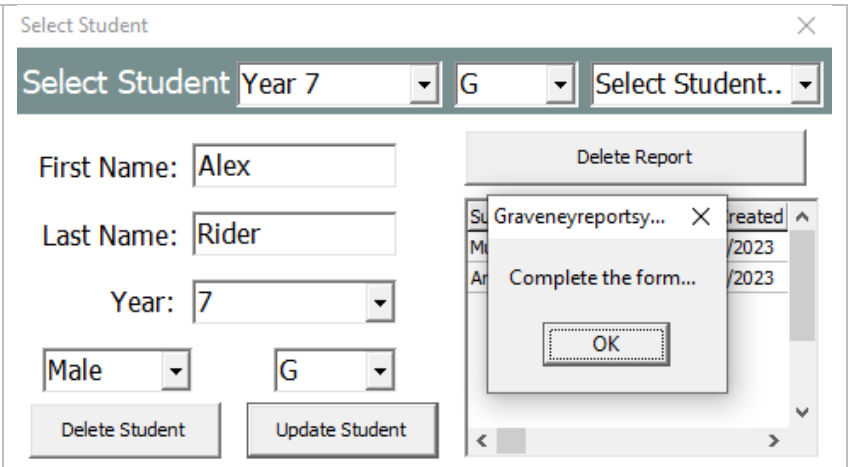
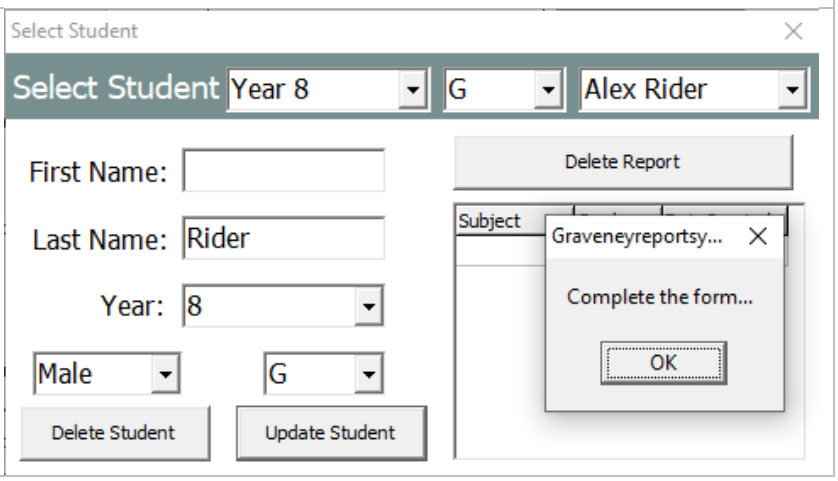
Selecting Student then open its report	
Summary: Test Successful – The user is able to open students previously created reports.	

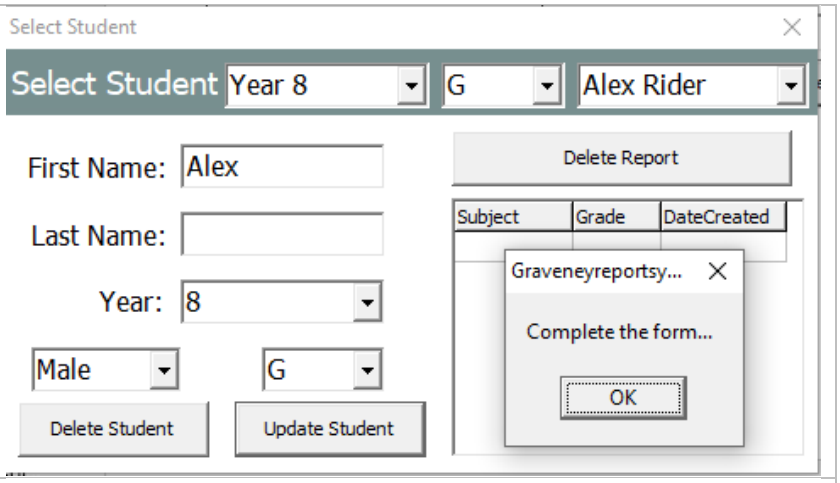
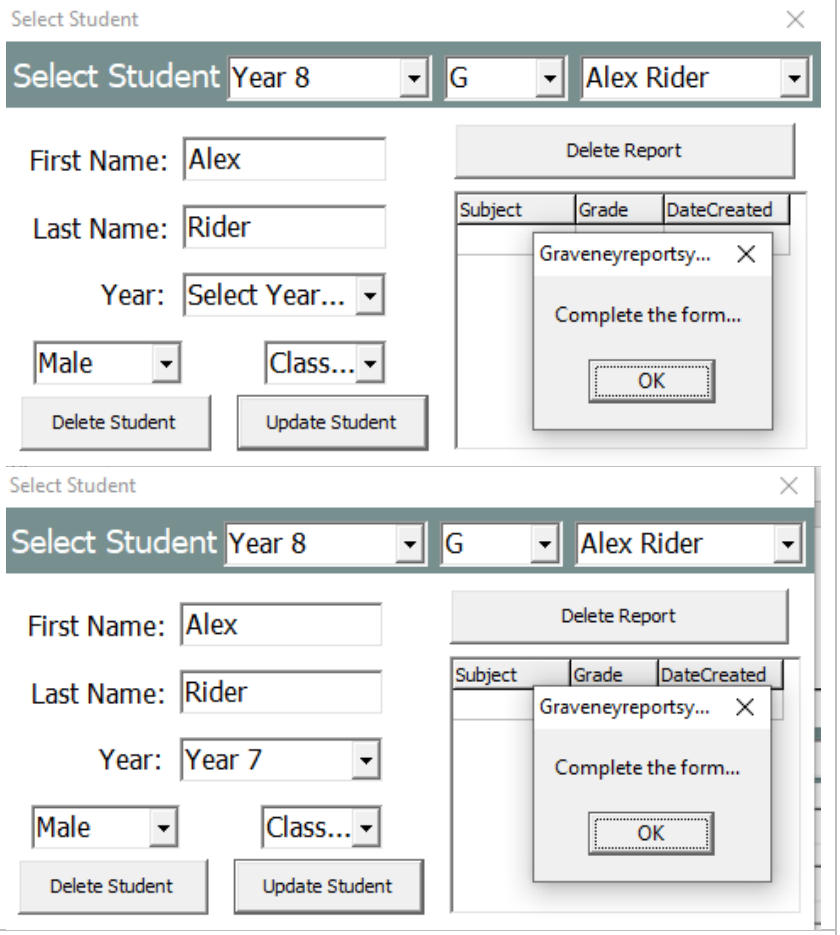
Test No 9.2.1 result: Checking if the form allows user to edit the student's values…

Before	

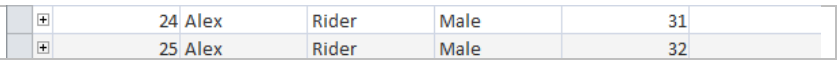
After	
Summary: Test Successful – The user is able to update the student's details, using valid values.	

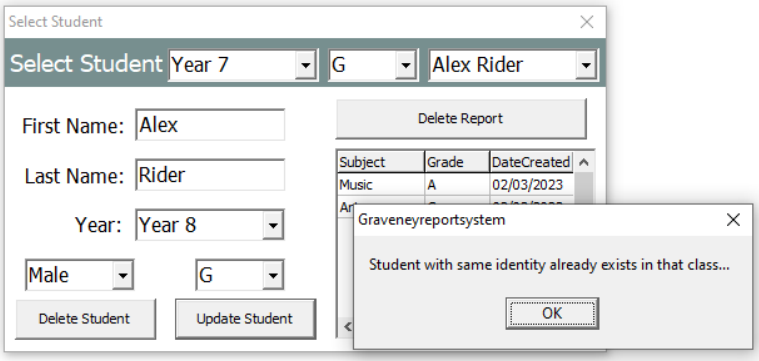
Test No 9.2.2 result: Checking result of attempting to update student values when form is not complete…

1) Attempting to update when student is not selected	
2) Name not completed	

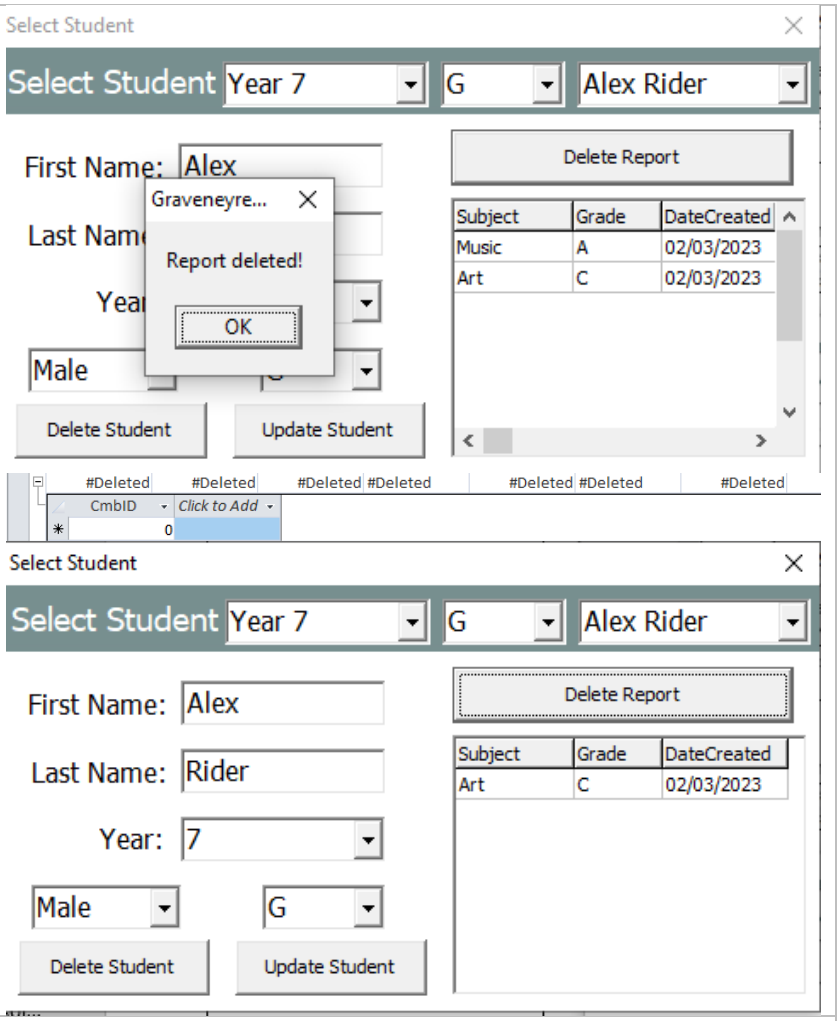
	
3) Year or class not selected	
Summary: Test Successful – The student is not updated and the user gets prompt with the message to complete the form.	

Test No 9.2.3 result: Checking result of attempting to update to an existing student…

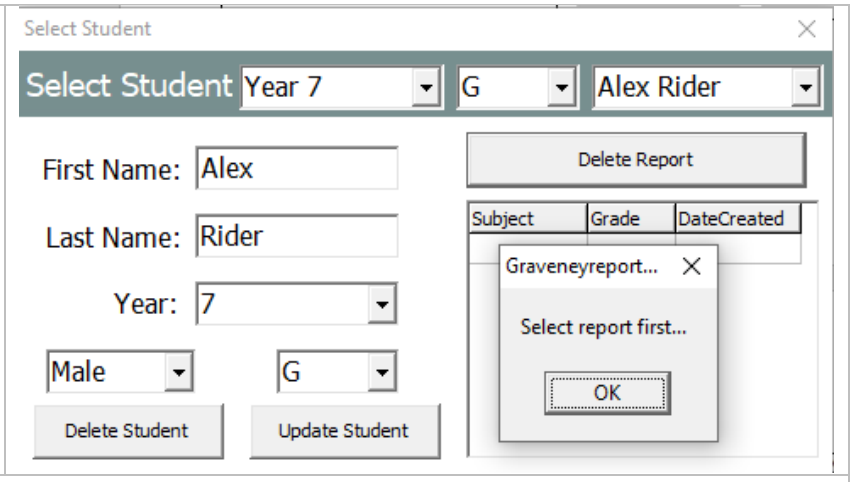
Students existing	

Attempting to update to that student	
Summary: Test Successful – The student from one class cannot be moved to another class with a student with his exact name. This is to prevent duplicates and confusion between students for user.	

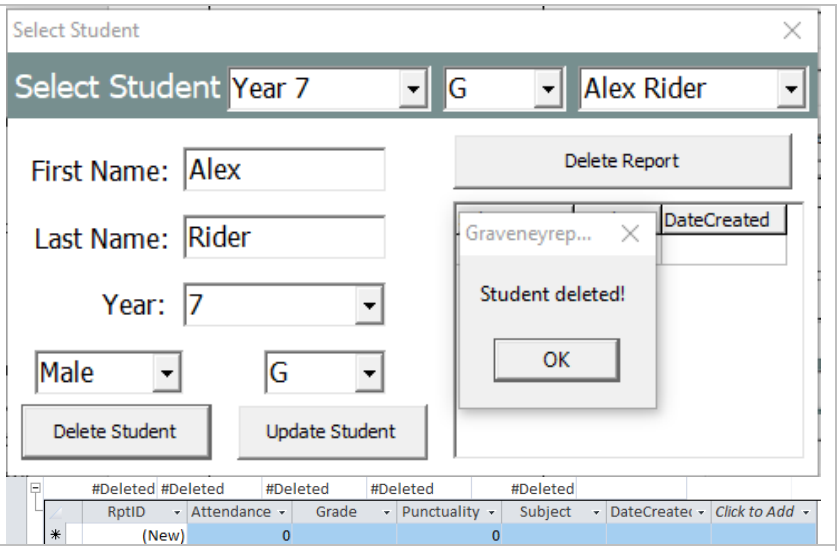
Test No 9.3.1 result: Checking if user is able to delete a report…

Select and delete report	
Summary: Test Successful – the user is able to delete reports individually of students.	

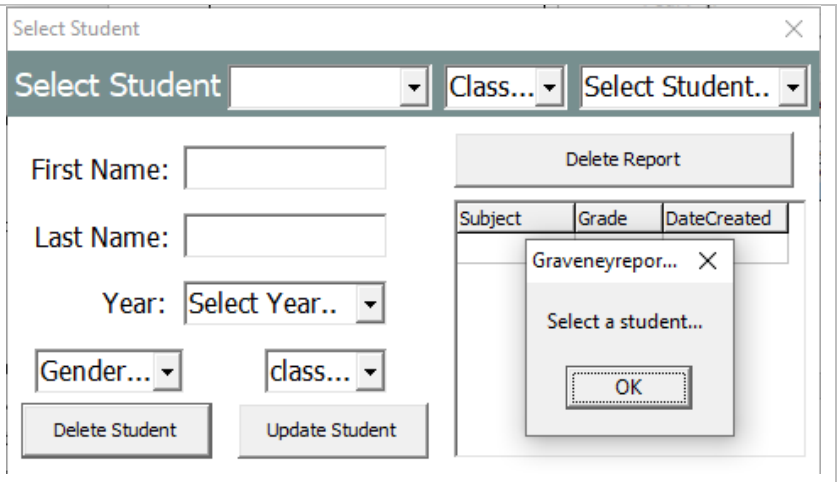
Test No 9.3.2 result: Checking systems output when user attempts to delete a report, when no report is selected…

Press delete report	
Summary: Test Successful – instead of displaying an unreadable error message, the system is able to output a user friendly error message to the user.	

Test No 9.4.1 result: Checking if user can delete student…

Press delete report	
Summary: Test Successful – the system allows users to delete reports for students.	

Test No 9.4.2 result: Checking systems output when user attempts to delete a student, when no student is selected…

Press delete student	
Summary: Test Successful – instead of displaying an unreadable error message, the system is able to output a user friendly error message to the user.	

Stage 10: Hashing and Positioning

Prototype 1

Changes in Implemented Code for Signup Form (unit_Signup)

```
Public
  …
  function hash(salt, mypassword:string):string; //used in login and signup
  function generatesalt(passlength:Integer):string; // used in update and signup
```

```
// Reverses String order
Function ReverseString(AText: string): string;
var
    i,j:integer;
begin
  setlength(result,length(atext));
  i:=1; j:=length(atext);
  while (i<=j) do
    begin
      result[i]:=atext[j-i+1];
      inc(i);
    end;
end;

// Generates salt for hashing
function TfrmSignup.generatesalt(passlength:Integer):string;
var i, gensalt:Integer;
  salt:string;
begin
  for i:=1 to 50-passlength do
  begin
    gensalt:=randomrange(32,127);
    Randomize;
    salt:= salt + chr(gensalt);
  end;
  Result := salt;
end;

// hashes the password using a pregenerated salt
function TfrmSignup.hash(salt, mypassword:string):string;
var i, passlength, myresult, indexval, denhashedinteger:  integer;
  saltedPassword, hashedpassword, exclusivedata, revpassword,denhashedpassword:
string;
  addcharacter:char;
begin
  passlength:= length(mypassword);
  saltedpassword := mypassword+ salt;
  revpassword := reversestring(mypassword);
  exclusivedata :='';
  hashedpassword:='';
  denhashedpassword :='';

  for i:=0 to 49 do
  begin
    indexval:= (i mod passlength) + 1;
    addcharacter := revpassword[indexval];
    exclusivedata := exclusivedata + addcharacter;
  end;

  for i:=1 to 50 do
  begin
    denhashedinteger:= (ord(exclusivedata[i]) XOR ord(saltedpassword[i]));
```

```
    denhashedpassword:= denhashedpassword + chr(denhashedinteger);
  end;

  for i := 1 to 50 do
  begin
    myresult:=  ord(denhashedpassword[i]);
    hashedpassword := hashedpassword + InttoHex(myresult,2);
  end;

  Result := hashedpassword;
end;
```

Justification: Making the algorithm for the system being able to hash a string with a salt given. Also a algorithm to generate salt.

Changes in Implemented Code for "Signup Form" (unit_signup)

Before

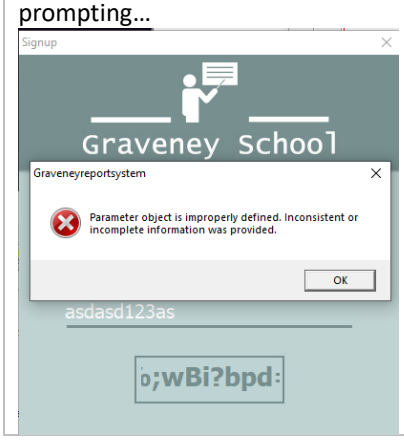
```
function writeAccount(fname,lname,email,password:string):boolean;
var LgnID:Integer;
begin
  frmSignup.qrySignup.SQL.Text := 'Select LgnID from login where
email='+quotedStr(email)+';';
  frmSignup.qrySignup.open;
  if (frmSignup.qrySignup.RecordCount=0)then
  begin
    frmSignup.qrySignup.close;
    frmSignup.qrySignup.SQL.Text := 'insert into login(email,password) values
('+quotedStr(email)+','+ quotedStr(password)+');';
    frmSignup.qrySignup.ExecSQL;
    frmSignup.qrySignup.SQL.Text := 'Select LgnID from login where
email='+quotedStr(email)+';';
    frmSignup.qrySignup.open;
    LgnID := frmSignup.qrySignup.fieldbyname('LgnID').AsInteger;
    frmSignup.qrySignup.close;
    frmSignup.qrySignup.SQL.Text := 'insert into staff(LgnID,Firstname,Lastname)
values ('+inttostr(LgnID)+','+quotedStr(fname)+','+ quotedStr(lname)+');';
    frmSignup.qrySignup.ExecSQL;
    ShowMessage('account created');
    writeAccount:=True;
  end
  else
  begin
    frmSignup.qrySignup.close;
    ShowMessage('account with email address already exists');
    writeAccount:=False;
  end;
end;
```

After

```
function writeAccount(fname,lname,email,password:string):boolean;
var LgnID:Integer;
   salt:string;
begin
  salt:='';
  frmSignup.qrySignup.SQL.Text := 'Select LgnID from login where
email='+quotedStr(email)+';';
  frmSignup.qrySignup.open;
  if (frmSignup.qrySignup.RecordCount=0)then
  begin
    frmSignup.qrySignup.close;
    salt := frmSignup.generatesalt(Length(password));
```

```
    frmSignup.qrySignup.SQL.Text := 'insert into login(email,password,salt) values
('+quotedStr(email)+','+ quotedStr(frmSignup.hash(salt,password))+', :saltpara
);';
    frmSignup.qrySignup.Parameters.ParamByName('saltpara').DataType := ftString;
    frmSignup.qrySignup.Parameters.ParamByName('saltpara').value := salt;
    frmSignup.qrySignup.ExecSQL;
    frmSignup.qrySignup.SQL.Text := 'Select LgnID from login where
email='+quotedStr(email)+';';
    frmSignup.qrySignup.open;
    LgnID := frmSignup.qrySignup.fieldbyname('LgnID').AsInteger;
    frmSignup.qrySignup.close;
    frmSignup.qrySignup.SQL.Text := 'insert into staff(LgnID,Firstname,Lastname)
values ('+inttostr(LgnID)+','+quotedStr(fname)+','+ quotedStr(lname)+');';
    frmSignup.qrySignup.ExecSQL;
    ShowMessage('account created');
    writeAccount:=True;
  end
  else
  begin
    frmSignup.qrySignup.close;
    ShowMessage('account with email address already exists');
    writeAccount:=False;
  end;
end;
```

Justification: Generates a salt and hashes the password. Then stores both in the database.
For storing the salt in the database, I was required to use parameter method, otherwise there was a constant error prompting…



Changes in Implemented Code for "Login Form" (unit_login)

Before

```
procedure databaseLogin;
var min,max,i:Integer;
  temp:string;
  success:Boolean;
begin
  frmLogin.qryLogin.SQL.Text := 'SELECT staff.StfID, Login.Email,  Login.Password,
staff.firstname, staff.lastname FROM Login inner join staff on (login.LgnID =
staff.LgnID) where email = '+quotedstr(frmLogin.edtemail.text)+';';
  frmLogin.qryLogin.Open;
  if (frmLogin.qryLogin.RecordCount=0)then
    frmLogin.lblerror.caption:='Email not found'
  else
  begin
    if (frmLogin.qryLogin.fieldbyname('password').AsString =
frmLogin.edtpassword.Text)then
      begin
```

```
        info.FirstName:=frmLogin.qryLogin.fieldbyname('firstname').AsString;
        info.LastName:=frmLogin.qryLogin.fieldbyname('lastname').AsString;
        info.Email:=frmLogin.qryLogin.fieldbyname('email').AsString;
        info.password:=frmLogin.qryLogin.fieldbyname('password').AsString;
        info.StaffID:=frmLogin.qryLogin.fieldbyname('StfID').AsInteger;
        frmMainMenu.lblProfileName.Caption := info.Firstname+' '+info.lastname;
        frmMainMenu.Top := frmLogin.Top;
        frmMainMenu.left := frmLogin.left;
        frmmainmenu.Show;
        frmlogin.Hide;
      end
      else
      begin
        frmLogin.lblerror.Caption:='incorrect password';
        frmLogin.edtpassword.Text := 'password';
        frmLogin.edtpassword.PasswordChar := #0;
      end;
  end;
  frmLogin.qryLogin.close;
end;
```

After

```
procedure databaseLogin;
var hashedpassword, salt:string;
begin
  frmLogin.qryLogin.SQL.Text := 'SELECT staff.StfID, Login.Email,
Login.Password,login.salt, staff.firstname, staff.lastname FROM Login inner join
staff on (login.LgnID = staff.LgnID) where email =
'+quotedstr(frmLogin.edtemail.text)+';';
  frmLogin.qryLogin.Open;
  if (frmLogin.qryLogin.RecordCount=0)then
    frmLogin.lblerror.caption:='Email not found'
  else
  begin
    salt := frmLogin.qryLogin.fieldbyname('salt').AsString;
    hashedpassword := frmSignup.hash(salt, frmLogin.edtpassword.Text);
    if  (frmLogin.qryLogin.fieldbyname('password').AsString = hashedpassword)then
      begin
        info.FirstName:=frmLogin.qryLogin.fieldbyname('firstname').AsString;
        info.LastName:=frmLogin.qryLogin.fieldbyname('lastname').AsString;
        info.Email:=frmLogin.qryLogin.fieldbyname('email').AsString;
        info.password:=frmLogin.qryLogin.fieldbyname('password').AsString;
        info.StaffID:=frmLogin.qryLogin.fieldbyname('StfID').AsInteger;
        frmMainMenu.lblProfileName.Caption := info.Firstname+' '+info.lastname;
        frmMainMenu.Top := frmLogin.Top;
        frmMainMenu.left := frmLogin.left;
        frmmainmenu.Show;
        frmlogin.Hide;
      end
      else
      begin
        frmLogin.lblerror.Caption:='incorrect password';
        frmLogin.edtpassword.Text := 'password';
        frmLogin.edtpassword.PasswordChar := #0;
      end;
  end;
  frmLogin.qryLogin.close;
end;
```

Justification: Gets the salt corresponding to the email from the database, then it hashes the password entered. It then compairs the hashed password with the hashed password entered. If they equal, login success.

Changes in Implemented Code for "Update setting Form" (unit_UpdateSetting)

Before

```
procedure TfrmUpdateSetting.btnUpdateClick(Sender: TObject);
var loginID:string;
begin
  if not((edtfName.Text = '') or (edtlName.Text = '') or (edtpassword.Text ='') or
(edtEmail.Text = '') or (edtpassword.Text ='new password'))then
  begin
    if (frmSignup.verifyName_and_Password(edtfName.Text, edtlName.Text,
edtpassword.Text))then
    begin
      if (frmSignup.IsValidEmail(edtEmail.Text))then
      begin
        LoginID:=getLgnIDusingStfID(IntToStr(info.staffID));
        qryUpdateSetting.SQL.Text:='select lgnID from login where
email='+quotedStr(edtemail.Text)+';';
        qryUpdateSetting.Open;
        if
not((qryUpdateSetting.RecordCount<>0)and(qryUpdateSetting.FieldByName('lgnID').AsS
tring<>loginID))then
        begin
          qryupdatesetting.SQL.Text:='update Staff set
FirstName='+quotedStr(frmSignup.FirstLetterUppercase(edtfname.Text))+',
LastName='+quotedStr(frmSignup.FirstLetterUppercase(edtlname.Text))+' where
StfID='+inttostr(info.staffID)+';';
          qryUpdateSetting.ExecSQL;
          qryupdatesetting.SQL.Text:='update Login set
Email='+quotedStr(edtemail.Text)+', password='+quotedStr(edtpassword.Text)+' where
LgnID='+loginID+';';
          qryUpdateSetting.ExecSQL;
          showmessage('Account Updated');
        end
        else
          ShowMessage('Account with email address already exists...');
      end
      else
        ShowMessage('invalid email address');
    end;
  end
  else
    ShowMessage('fill in the form...');
end;
```

After

```
procedure TfrmUpdateSetting.btnUpdateClick(Sender: TObject);
var loginID, salt:string;
begin
  if not((edtfName.Text = '') or (edtlName.Text = '') or (edtpassword.Text ='') or
(edtEmail.Text = '') or (edtpassword.Text ='new password'))then
  begin
    if (frmSignup.verifyName_and_Password(edtfName.Text, edtlName.Text,
edtpassword.Text))then
    begin
      if (frmSignup.IsValidEmail(edtEmail.Text))then
      begin
        LoginID:=getLgnIDusingStfID(IntToStr(info.staffID));
        qryUpdateSetting.SQL.Text:='select lgnID from login where
email='+quotedStr(edtemail.Text)+';';
        qryUpdateSetting.Open;
```

```
        if
not((qryUpdateSetting.RecordCount<>0)and(qryUpdateSetting.FieldByName('lgnID').AsS
tring<>loginID))then
        begin
          salt := frmSignup.generatesalt(Length(edtpassword.Text));
          qryupdatesetting.SQL.Text:='update Staff set
FirstName='+quotedStr(frmSignup.FirstLetterUppercase(edtfname.Text))+',
LastName='+quotedStr(frmSignup.FirstLetterUppercase(edtlname.Text))+' where
StfID='+inttostr(info.staffID)+';';
          qryUpdateSetting.ExecSQL;
          qryupdatesetting.SQL.Text:='update Login set
Email='+quotedStr(edtemail.Text)+',
password='+quotedStr(frmSignup.hash(salt,edtpassword.Text))+',
salt='+quotedStr(salt)+' where LgnID='+loginID+';';
          qryUpdateSetting.ExecSQL;
          showmessage('Account Updated');
        end
        else
          ShowMessage('Account with email address already exists...');
      end
      else
        ShowMessage('invalid email address');
    end;
  end
  else
    ShowMessage('fill in the form...');
end;
```

Justification: Generates a new salt for the new password since the length may differ.

Changes in Database (unit_CreateReport)

Before	After
Login (LgnID, Email, Password)	Login (LgnID, Email, Password, Salt)

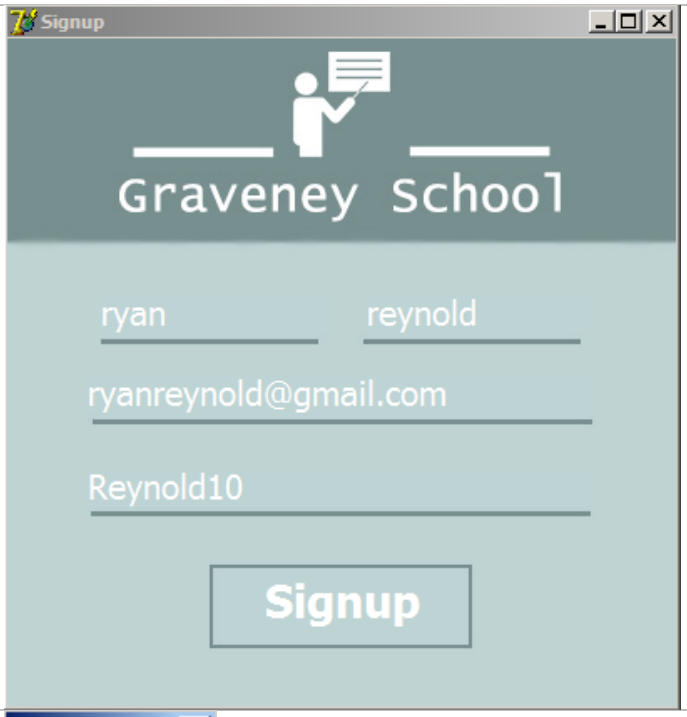
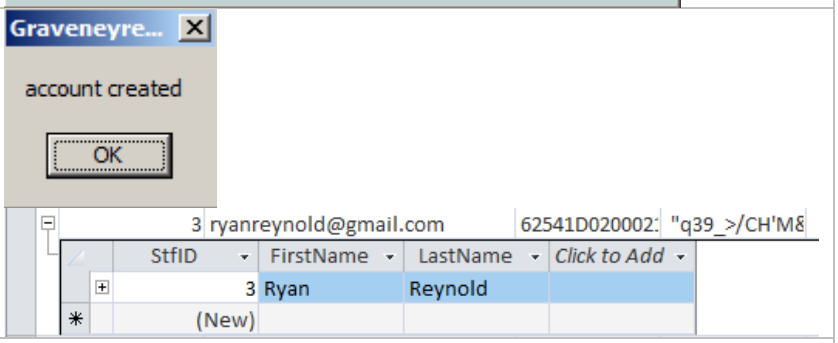
Justification: The system needs to be able to store the salt which is used to hash the password, when account is created, so that when the user logs in, the same salt can be applied to the password entered by the user. This allows the user to login and use the system.
Salt Verification: Length must be less than 50
Password Verification: Length must be less than 50

What has been done

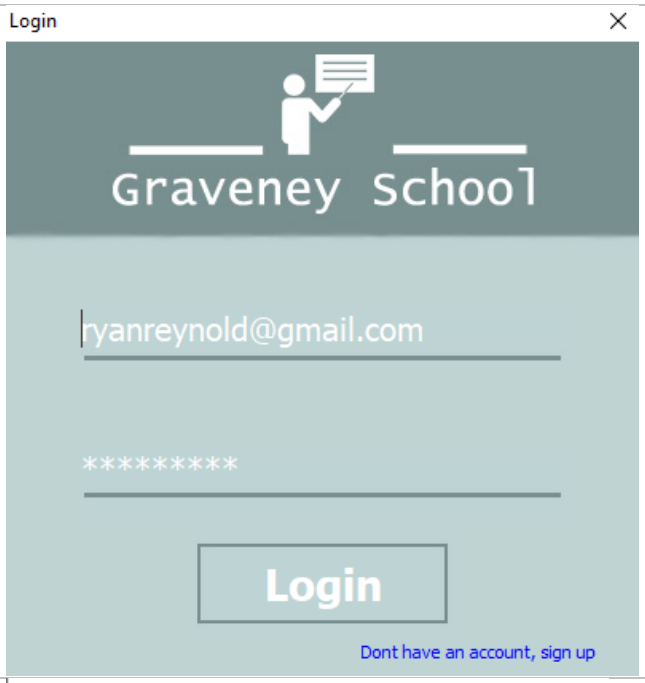
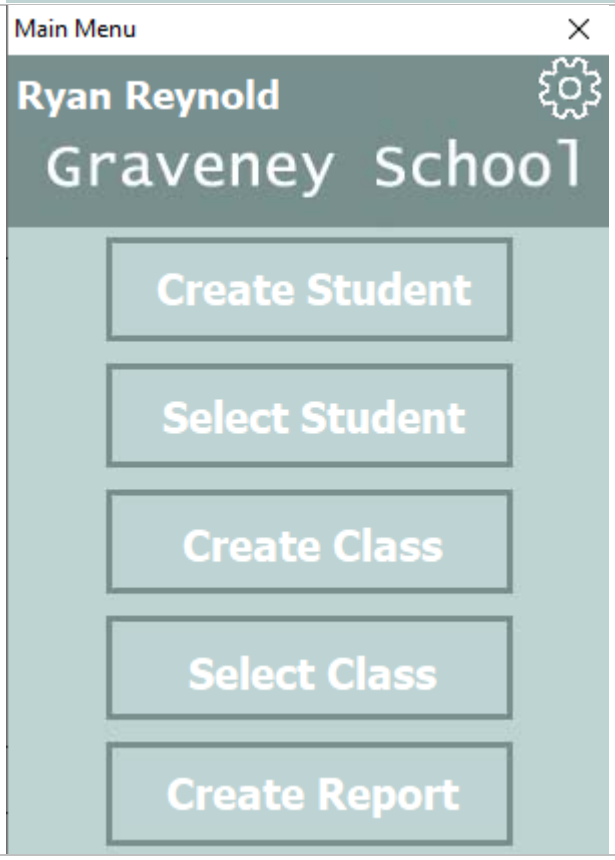
The system now hashes the password stored in the database for security reasons. Now, users are less prone for being hacked. Requirement 17 has been met.

Summary: The user is able to create and sign in using an account. The user is able to update their details. They are also able to increment all years by one and delete unneeded data at the start of a new academic year by pressing a single button, within the update setting form. The main menu form has also been created, which will allow users to open different forms for the solution. The user is able to create classes and students within it. They can view all students within a class, delete the class along with the students and their data inside the class, and update classes. Users are able to create a report of an existing student, and view the report in a separate window. The user is able to export the report as a PDF or/ and copy the comment. The user is able to view and/ or delete student's reports from past, and update and/ or delete students.

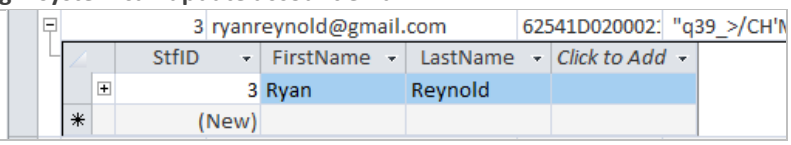
Retest No 3.1 result: checking if users can create an account…

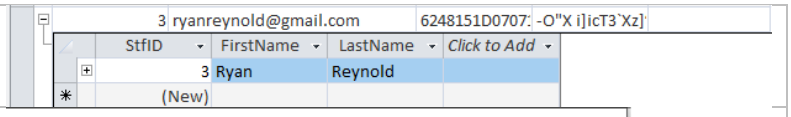
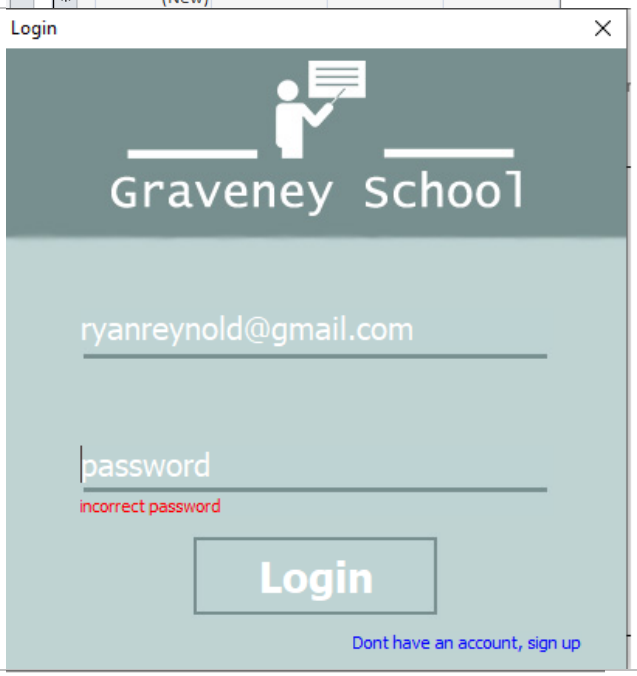
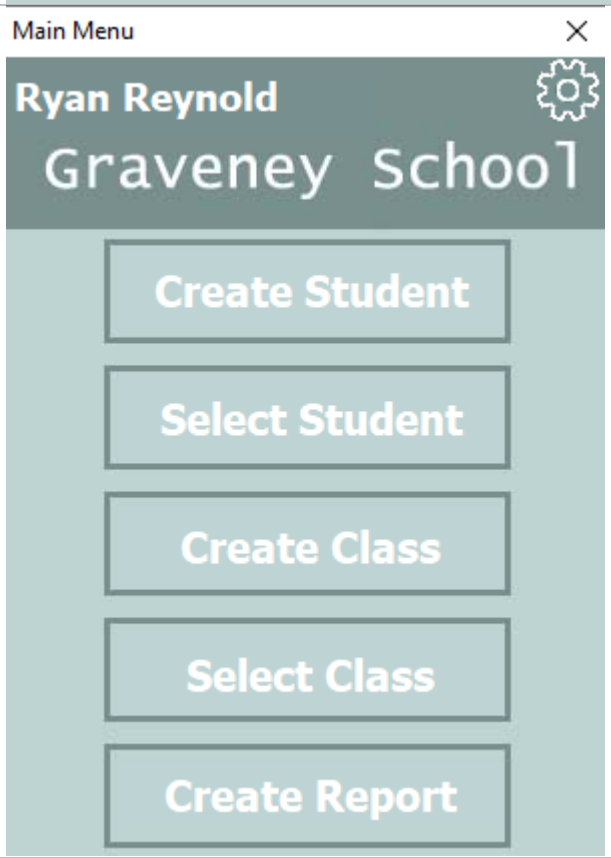
Step 1) Entering the values	
Step 2) pressing signup button	
Summary: Test Successful – The system is able to create an account and have it stored within the database. This allows the user to login. The system also prompts the user a message when the account has been created. Requirement 5.3 and 5.1 has been met. The system hashes the password when storing it in the database, and also stores the salt. Storing the salt is needed to use it to hash the password entered at login by the user to check if the password is valid.	

Retest No 2.1.1 result: checking if users can login with their accounts…

Step 1) Inputting the test data values… Email: ryanreynold@gmail.com Password: Reynold10	
Step 2) Pressing Login Button	
Summary: Test Successful – The system is able to login and displayed the menu form, when the correct values of a registered account are entered. Hashing is a success in login unit.	

Retest No 6.3.1 result: checking if system can update account email…

Before Updating	

After Updating Email: ryanreynold@gmail.com Password: Ryanbest10	
Trying to Login with old password	
Trying to Login with new password	
Summary: Test Successful – The System is able to update an account with valid values. It changed the email within the database. Users is informed when the account was updated. The system is able to generate a new salt to comprehend with the new length of the password. The update setting form is able to hash the password and update salt and password in database.	

Prototype 2

One of the hardware requirements was for the system to work efficiently with different type of resolution monitors, since the school consists of a mixture of monitors of resolution 720p and 1080p, requirement 2.This prototype aims to make the form open in the centre of any monitor whenever the system is ran.

Changes in Implemented Code for Graveney Report System (GraveneyReportSystem)

Before

```
begin
  Application.Initialize;
  Application.CreateForm(TfrmLogin, frmLogin);
  Application.CreateForm(TfrmReportComment, frmReportComment);
  Application.CreateForm(TfrmExportReport, frmExportReport);
  Application.CreateForm(TfrmSelectStudent, frmSelectStudent);
  Application.CreateForm(TfrmSignup, frmSignup);
  Application.CreateForm(TfrmMainMenu, frmMainMenu);
  Application.CreateForm(TfrmCreateClass, frmCreateClass);
  Application.CreateForm(TfrmCreateStudent, frmCreateStudent);
  Application.CreateForm(TfrmSelectClass, frmSelectClass);
  Application.CreateForm(TfrmUpdateSetting, frmUpdateSetting);
  Application.CreateForm(TfrmCreateReport, frmCreateReport);
  Application.Run;
end.
```

After

```
procedure changeWindowPos(sender:Tobject);
begin
  Tform(sender).Left := (screen.Width - Tform(sender).ClientWidth) div 2;
  Tform(sender).Top := (screen.Height - Tform(sender).ClientHeight) div 2;
end;
begin
  Application.Initialize;
  Application.CreateForm(TfrmLogin, frmLogin);
  Application.CreateForm(TfrmReportComment, frmReportComment);
  Application.CreateForm(TfrmExportReport, frmExportReport);
  Application.CreateForm(TfrmSelectStudent, frmSelectStudent);
  Application.CreateForm(TfrmSignup, frmSignup);
  Application.CreateForm(TfrmMainMenu, frmMainMenu);
  Application.CreateForm(TfrmCreateClass, frmCreateClass);
  Application.CreateForm(TfrmCreateStudent, frmCreateStudent);
  Application.CreateForm(TfrmSelectClass, frmSelectClass);
  Application.CreateForm(TfrmUpdateSetting, frmUpdateSetting);
  Application.CreateForm(TfrmCreateReport, frmCreateReport);
  changeWindowPos(frmSelectClass);
  changeWindowpos(frmUpdateSetting);
  changeWindowPos(frmlogin);
  changeWindowpos(frmCreateStudent);
  changeWindowPos(frmCreateClass);
  changeWindowPos(frmCreateReport);
  Application.Run;
end.
```

Changes in Implemented Code for Main Menu (Unit_MainMenu)

```
procedure changeWindowPos(sender:Tobject);
begin
  Tform(sender).Left := (screen.Width - Tform(sender).ClientWidth) div 2;
  Tform(sender).Top := (screen.Height - Tform(sender).ClientHeight) div 2;
end;

…
```

```
procedure TfrmMainMenu.btnSelectStudentClick(Sender: TObject);
var i:integer;
  arr:yearArr;
begin
  FreeAndNil(frmselectstudent);
  frmSelectStudent := TfrmSelectStudent.Create(frmselectstudent);
  arr := selectyear(frmSelectStudent.qrySelectStudent, frmSelectStudent);
  changeWindowPos(frmselectstudent);
…

procedure TfrmMainMenu.btnCreateReportClick(Sender: TObject);
var i:Integer;
  arr:yearArr;
begin
  FreeAndNil(frmCreateReport);
  frmCreateReport := TfrmCreateReport.Create(frmCreateReport);
  changeWindowPos(frmCreateReport);
…
```

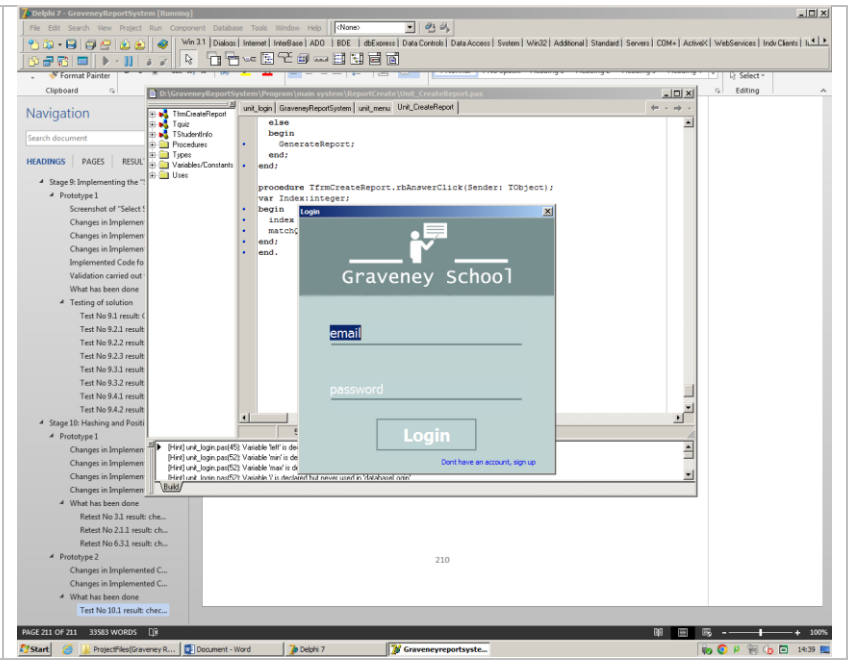
The select student and create report forms are created in the main menu unit hence the position of the two forms are needed to be corrected each time.

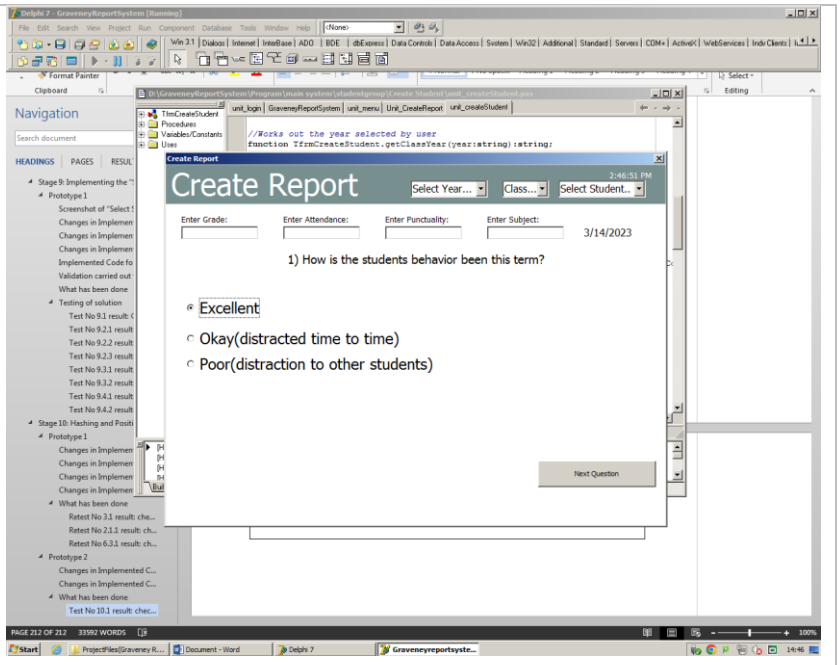
What has been done

The system now makes sure all forms open in the centre of any monitor for the system to be usable by any monitor. Requirement 2 has been met.

Summary: The user is able to create and sign in using an account. The password is hashed for security. The user is able to update their details. They are also able to increment all years by one and delete unneeded data at the start of a new academic year by pressing a single button, within the update setting form. The main menu form has also been created, which will allow users to open different forms for the solution. The user is able to create classes and students within it. They can view all students within a class, delete the class along with the students and their data inside the class, and update classes. Users are able to create a report of an existing student, and view the report in a separate window. The user is able to export the report as a PDF or/ and copy the comment. The user is able to view and/ or delete student's reports from past, and update and/ or delete students.

Test No 10.1 result: checking if form positions in the centre…

Login form, which is positioned in GraveneyReportSystem Unit	

Select student form, which is positioned in Main Menu Unit	
Summary: Test Successful – The system is able to open the forms in the centre on any resolution	

Section 4: Evaluation

4.1 System and End-user testing

Usability Testing

Darker teal colour, to make the heading stand out. A large logo on first form displaying schools name.

Label displaying what to enter on the input section, the line. White font, a standing out colour and not distracting.

Large button easy to read. States the purpose of the button. Changes colour when mouse on top to indicate it's clickable.

When mouse on top…



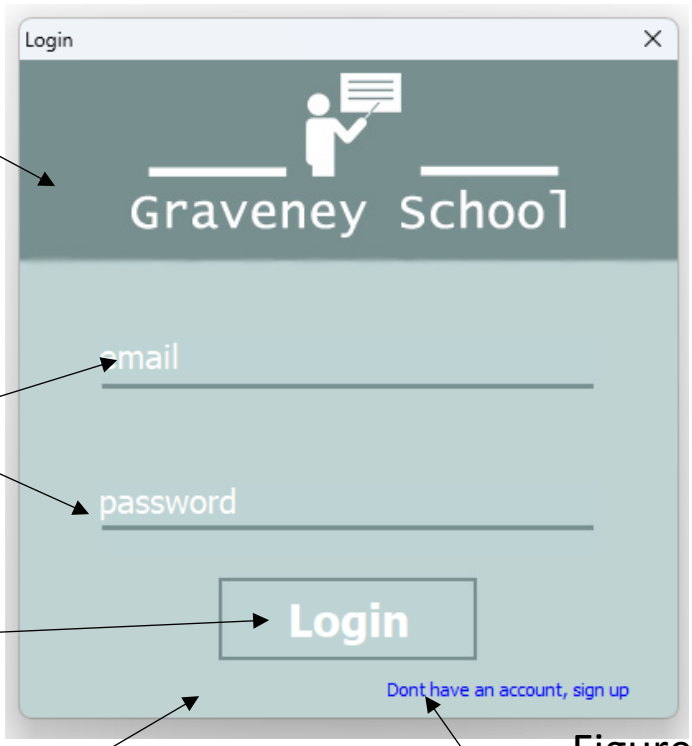
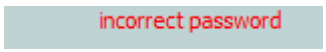


Figure 1

A light teal colour, easy to the eye and formal. Meets requirement 6.

Label with a standing out colour of blue. States it purpose and changes colour when mouse on top to indicate it's clickable.



A red label on login informing if email or password entered is incorrect.

Here is an example of one of the forms in the system. Multiple other forms have the same design and usability features as this form. As you can see, the design is simple, meeting requirement 6, and the school name is on the system to show who this program is made for, and to make sure that people know it is the property of Graveney School.

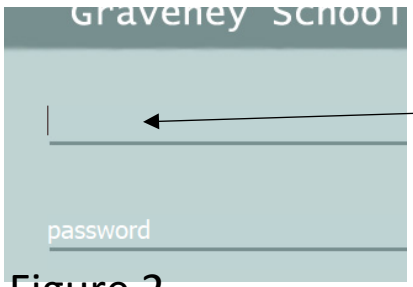


Figure 2

When you press on the text box, the default value, which indicates what it the user needs to type in disappears. The user is not required to manually delete the value using backspace every time.

When the value is left empty, the text box returns to its default value, always guiding the user what needs to be entered



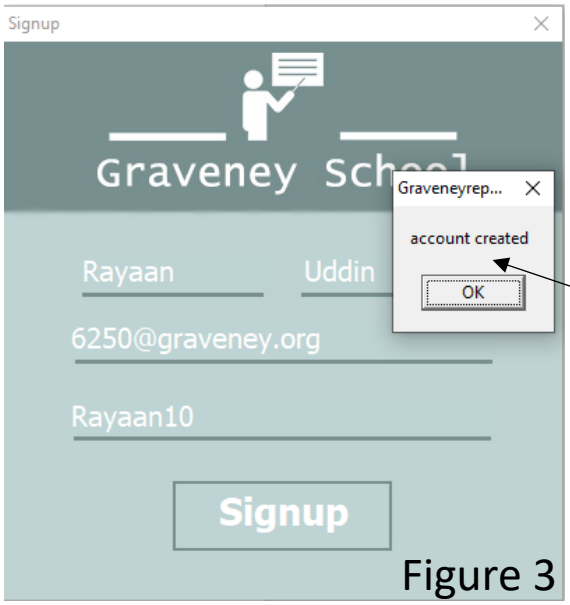


Figure 3

A pop up informing user

The system always informs the user when the users command is completed. In this example the user is creating a class and the system informs the user it has been created.

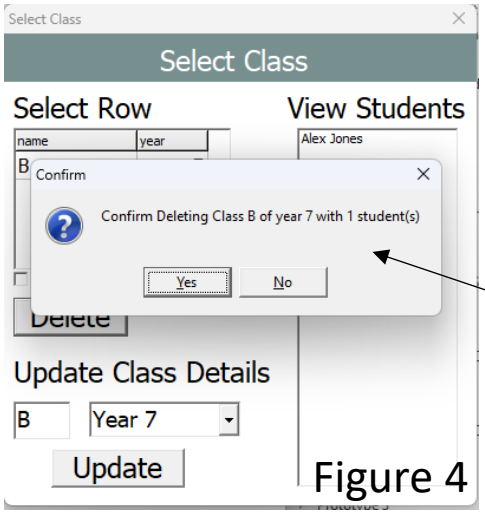


Figure 4

A conformation window confirming with the user if they want to still delete the…

The system always asks the user for confirmation on deletion of an item. This is to prevent accidental clicks. In this example, you can see the user goes to delete a class and the class has a student in it. The system informs the user that in deleting the class, the student will also delete, and confirms with the user if the user still wants to continue.

System testing

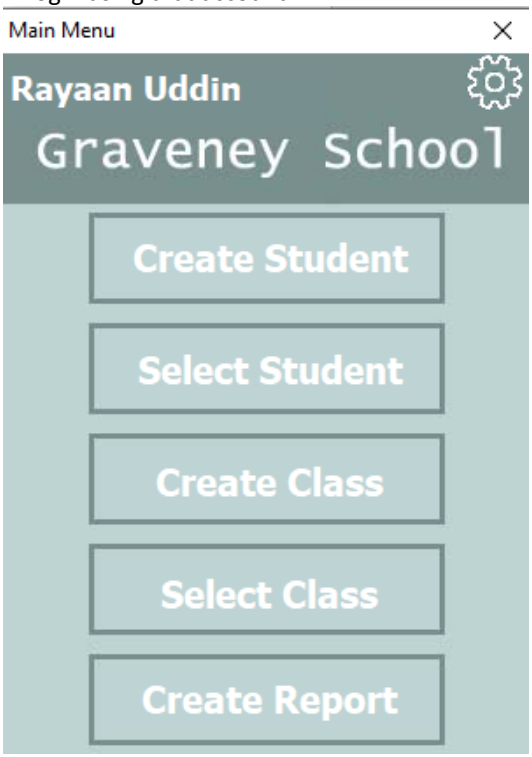
Test S1: Process of creating and logging into an account

Create an account



Adding the data and pressing the signup button to create the class. System displays a message informing them.

… login using that account



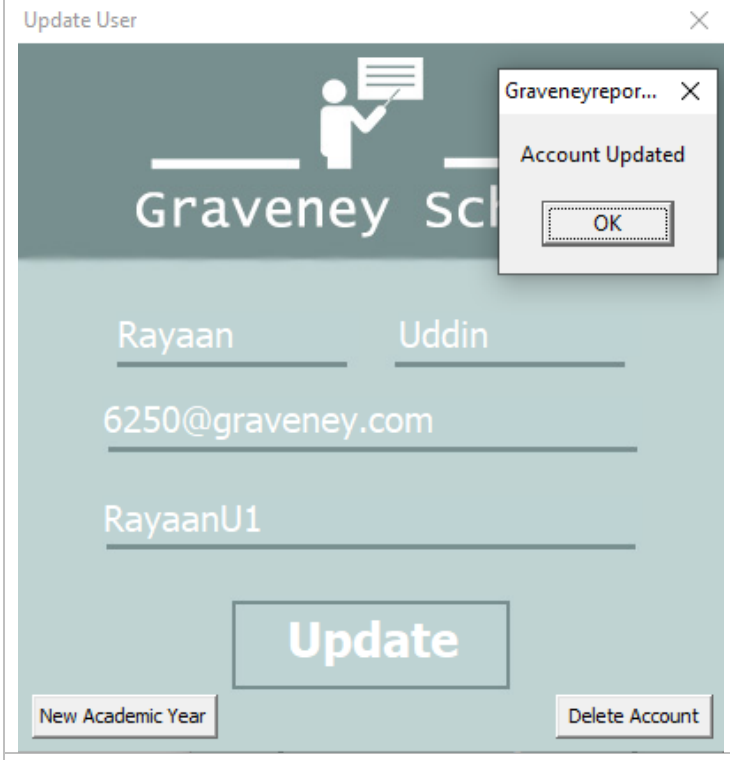
Logging in with the account created above. System Opens the main menu.

Requirements met: 4, 5, 6, 8.1, 8.2

Test Success: We were able to create an account and login into it successfully.

Test S2: Process of updating an account

… update its details

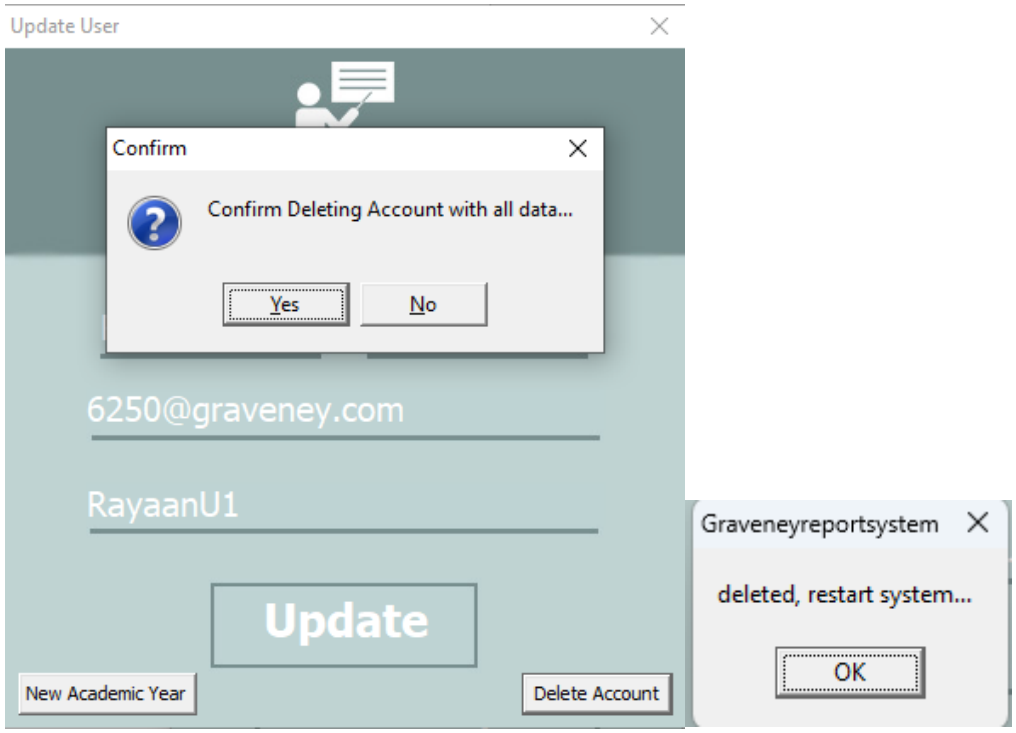


Requirements met: 6, 8.3, 9.1, 9.2

Test Success: We were able to update the account created previously.

Test S3: Process of deleting an account

… delete its account

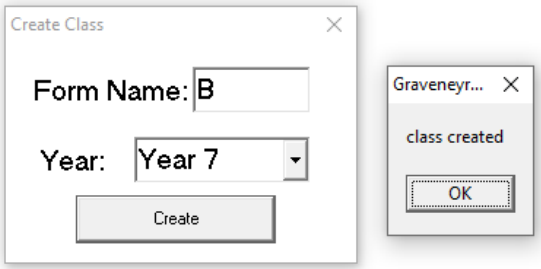


Pressing the delete button brings a conformation screen to delete the users account. It informs the user that all of their data will also be deleted. When pressed yes, the account is deleted and the system informs the user with a pop up message that it is deleted. After pressing okay the system is restarted.

Requirements met: 9.3
Test Success: We were able to delete the account created previously.

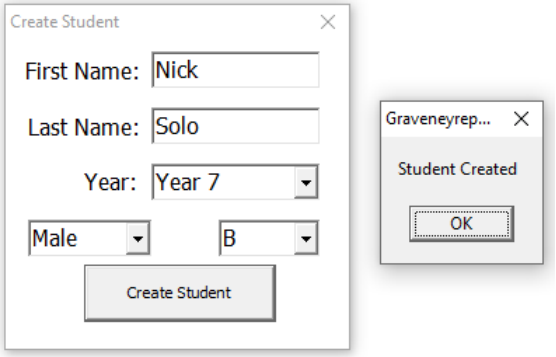
Test S4: Process of setting up students and classes, and sharing students

… create a class



Data was added into the text box, and the year was selected form a combo box. Then the user pressed create button which allowed the user to create a class. It then informed the user that the class has been created.

… create a student

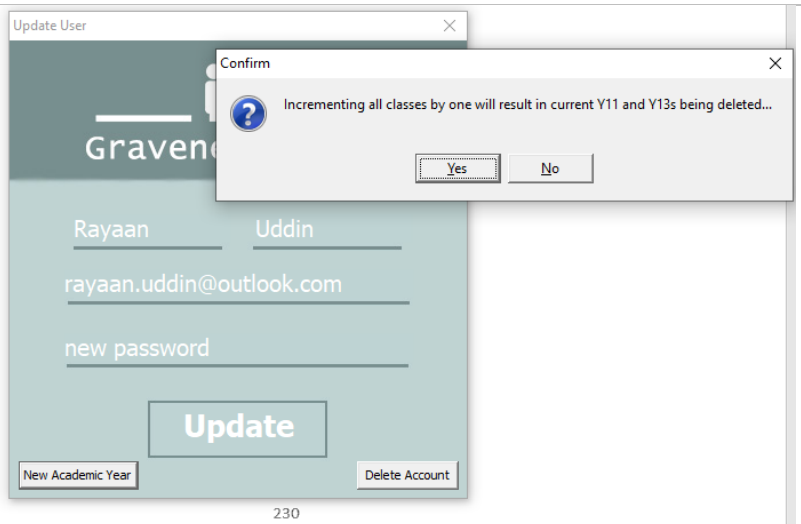


Data was added to the texted boxes to create a student. The existing class created before was selected, to store the student within it. The create student button was pressed to create the student. The system informed the user that the student has been created.

Requirements met: 6, 8.3, 10, 12 Requirements not met: 16
Test Success: We were able to successfully create a class, then use that class to create a student. Test Failed: We were not able to share a student to another user in the system.

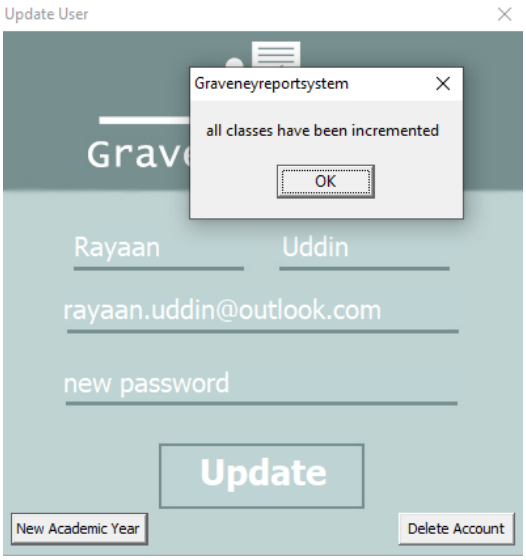
Test S5: Process of updating a class

… increment all class year by one

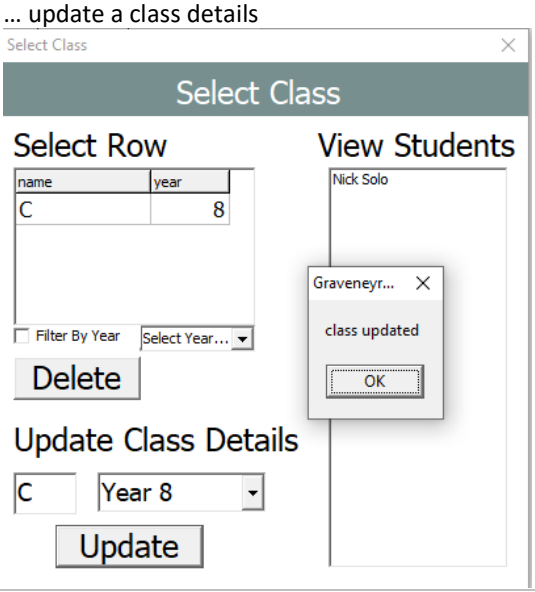


230

Pressing the New Academic Year button on the update setting form allows the user to increment all classes. Before in doing so, it confirms the with user if they are willing to do so as it does delete any previous year 11 and 13 classes.



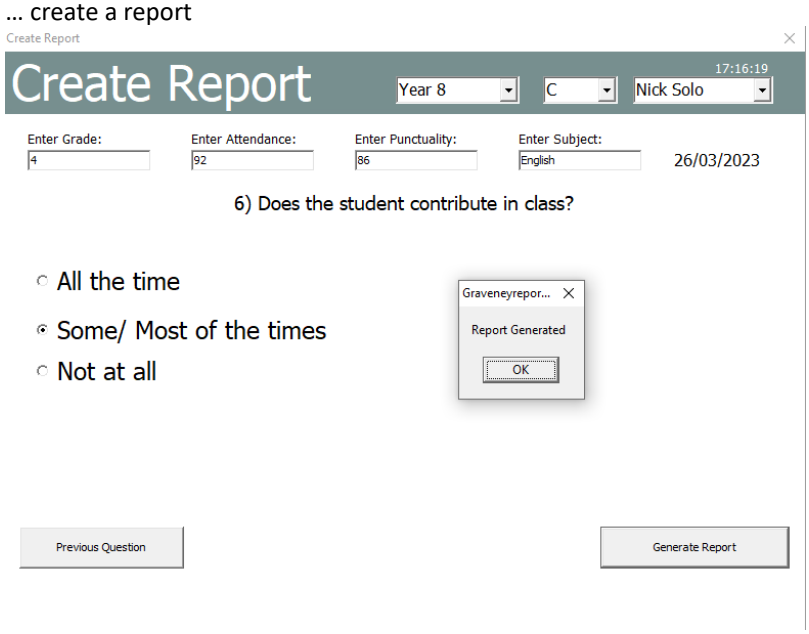
... update a class details



New details were added into the boxes. The update button was pressed and the user was informed that the class has been updated with a pop up message box.
Requirements met: 6, 8.3, 13.1, 13.3
Test Success: We were able to edit the class details and increment all classes by one, successfully.

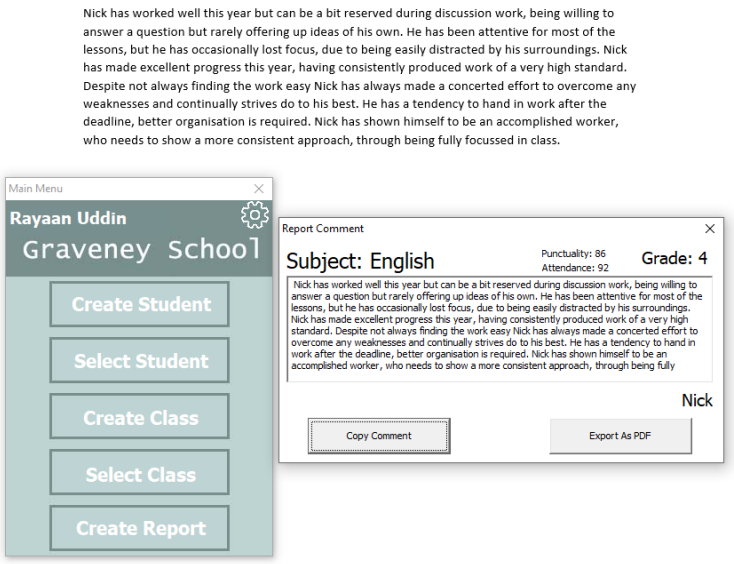
Test S6: Process of creating and copying a report

… create a report



Details of the report were added into the boxes and the student was selected. Then the multiple choice question was completed and generate report was pressed. The system informed that the report has been created and when pressed ok…
The report is viewed the user as shown below

… copy its comment



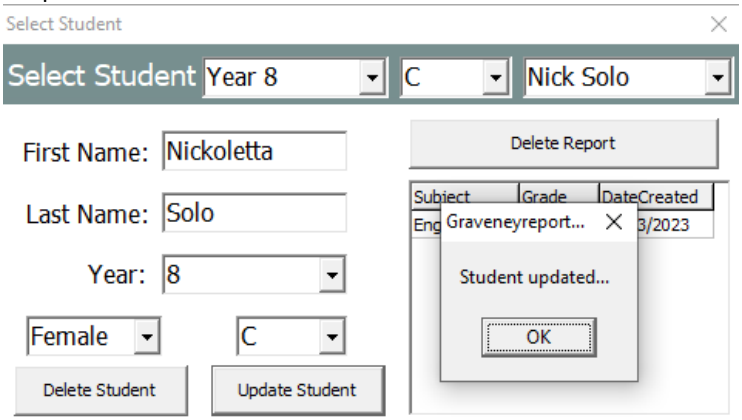
Copy button is pressed and the report is added to the user's clipboard.

Requirements met: 6, 8.3, 14.1, 14.2, 14.4, 15.1

Requirements not met: 14.3
Test Success: We were able to successfully create a report based on the student created above, and copy the comment and paste it in a word document. Test Failed: We were not able to edit the report inside the system.

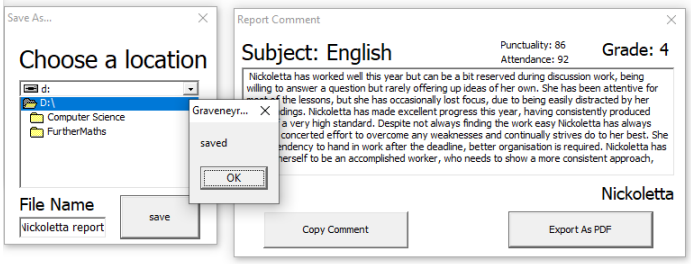
Test S7: Process of updating a student and exporting a report

… update a student

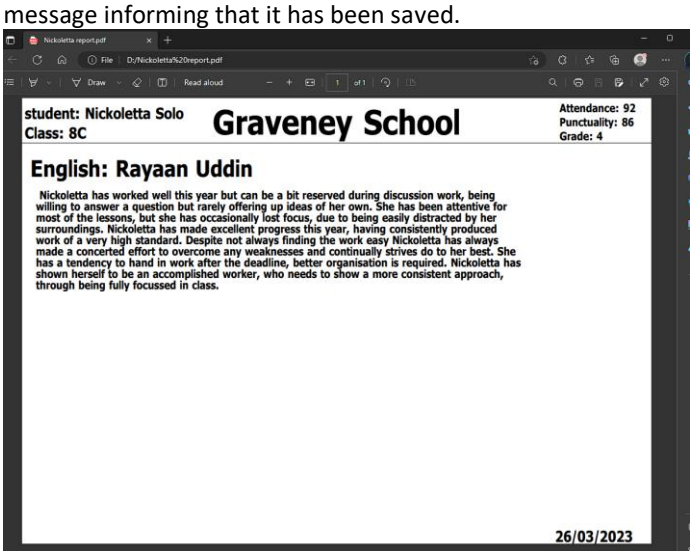


Student is selected and its details are auto filled in the text boxes and its reports are displayed. Data is replaced in the text boxes to update the students name and gender. When the update student button is clicked, the student is updated and the system pops a message up informing it has been updated.

… View and export a report



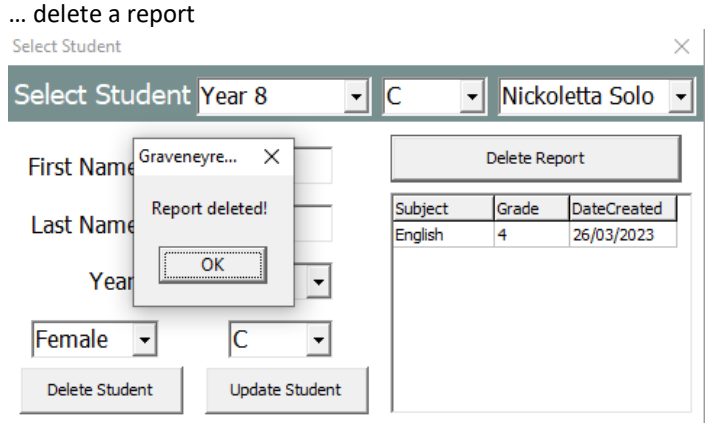
Clicking the report opens the report in the system. Pressing the export button shows a window to select the location to export to. When the save button is pressed the file is saved in the location and the system pops a message informing that it has been saved.



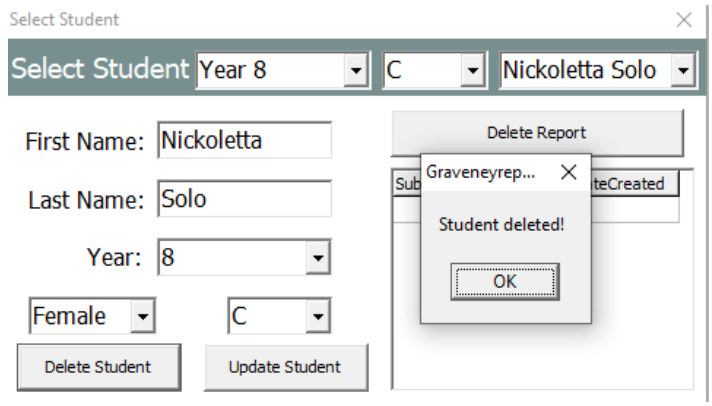
When opening the pdf the report is all shown.

Requirements met: 6, 8.3, 11.1, 15.3
Test Success: We were able to update the student's details and export the report successfully.

Test S8: Process of deleting a report

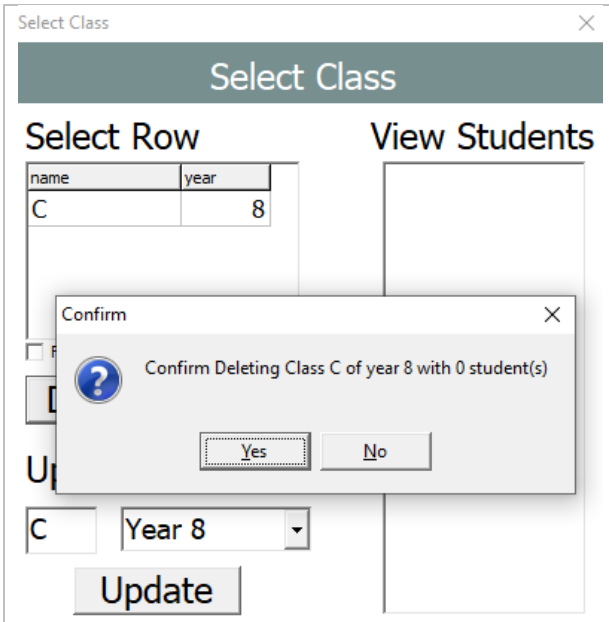
… delete a report  Clicking the delete report button, after selecting the report and student, deletes the report and the system pops a window to inform them.
Requirements met: 6, 8.3, 15.2
Test Success: We were able to delete a report successfully.

Test S9: Process of deleting a student

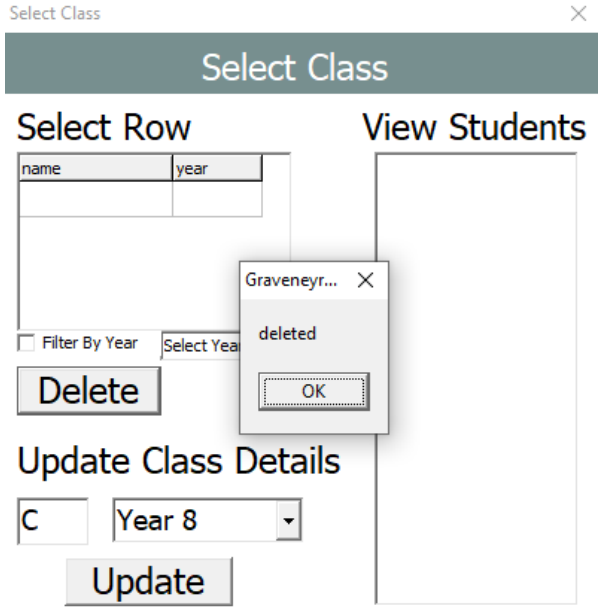
… delete a student  Clicking the delete student button, after selecting the student, deletes the student and the system pops a window to inform them.
Requirements met: 6, 8.3, 11.2
Test Success: We were able to delete a student successfully.

Test S10: Process of deleting a class

… delete a class



Clicking the delete button, after selecting the class, shows a pop up message informing the user how many students are in the class and that the students and its data will also be deleted.



Clicking Yes confirms to delete the class and the system informs them that the class has been deleted.

Requirements met: 6, 8.3, 13.2

Test Success: We were able to delete a class successfully.

End-user testing

I gave a questionnaire for both of my stakeholders to complete together, covering all the major requirements of the system and all of the criteria, to see if the system is complete.

System Questionnaire

Does the login screen show up when the program is ran?
Yes, when we launch the program, the login screen opens and we can interact with the program.

Does the login screen have an option to sign up?
Yes, the login screen has a small writing at the bottom which opens the sign up page.

When entered an invalid email, name, or password, does the system inform why the account is not being created, helping you out?
Yes. The program did validate our account.

Does the program allow you to create your account?
Yes, we were able to create an account.

Does the system allow you to create the same account?
No, when we attempted to enter the same details, the program informed use the account already exists.

Are you able to login with that account?
Yes, indeed we are. The main menu opens and our name shows on the top as intended too.

Does the program inform you if you got the password or email wrong when logging in?
Yes, it displays which part is incorrect in red on the bottom of the form above the login button.

Are you able to edit your details and login with the updated details?
Yes we were able to update our details fully, close the program and login with our account

Does the program allow you to open the create class window from the main menu?
Yes it does.

Are you able to create a class?
Yes we were able to create a class.

Do you get a message when you have entered the wrong class name?
Yes when we enter a class name which is not in the format of Graveney School, we received a pop up informing us.

Does the program allow you to open the create student window from the main menu?
Yes it does.

Are you able to create a student?
Yes we were able to create a student.

Does the program allow you to create the same student or class?
No it does not. It informs us that the student and class already exist.

Does the program allow you to open the create report window from the main menu?
Yes it does.

Does the program allow you to create a report and generate the comment for you?
Yes. The program asks all six multiple choice questions and generates the comment.

Does the program inform you if you have entered any details incorrectly, and tell you which one you have?
Yes it does.

Does the program display the report after it has been generated?

Yes it does, the report window pops up. It displays the reports and has two buttons to export or copy.

Does the program allow you to copy the report, and can you paste it anywhere on your computer?
Yes it does.

Does the program allow you to export the report as a pdf?
Yes it does.

Does the program display what location you want to export the report to?
Yes it displays a window to select a location after we pressed export report

Can you see all details of the report on the pdf file?
Yes it states the attendance, punctuality, grade, subject, teacher, date and the comment of the report.

Does the program allow you to open the select class window from the main menu?
Yes it does.

Does the program allow you to edit the class's details?
Yes it did allow us to edit the class name and year.

Does the program display all the students in the class?
Yes it did display all students in when we selected the class inside a box.

Does the program allow you to open the select student window from the main menu?
Yes it does.

Does the program allow you to edit a student's details?
Yes it allowed us to edit the students name, and gender and allows us to put him in another class.

Does the program allow you to open created reports?
Yes it allows us to open the reports from the select student screen the same way after you create the report.

Does the program allow you to increment all class years by one from the update settings window?
Yes it does.

Do all windows open in the centre of your screen?
Yes they do.

Does the system prevent you deleting any of you data accidentally?
Yes it does. We received a confirmation at each point to make sure we are aware of the action we are about to do.

Is the system designed to your like?
Yes it is. The program is very formal and has a clean user interface, with readable text and buttons, making it so much easier to use it and view it for a long period of time without containing your eyes.

How easy is it to navigate the system on a scale of 1 to 10?
The system is very easy to use. All buttons are named to its function and when using the system, the program lets you know why you can't create a class or account, etc. After 2 to 3 uses, I think I would pretty much get the hand of it.

One thing we think newer users would not know when using the system is the settings window, where we can update our details or increment all classes by one. It is a setting logo, but is much hidden, and does not react at all when we hovered our mouse on top of it, unlike every other button.

We think a manual for the system may be required since a lot of key features, such as viewing created reports by double clicking on it is not readable from the user interface.

Therefor I would give it an 8 out of 10, in terms of how much the program is user friendly.

What would oppose you from using the system as a teacher?

The system is great and serves it purpose, however if the system would allow us to edit the comment, it would be even better. We think the report comment that the report generates is good but generic at the same time, and with some students, we would require to add 2 to 3 extra lines about them personally in the report comment.

Did you experience any faults in the system?

There were no faults in the system, it functions fully.

To conclude

The system does mostly everything it is required to do. The stakeholder is happy with the system and how it operates. The system would be better if it let the users to edit the report comment. To improve the system, we should aim to meet requirement 14.3, where the user will be able to edit the report. Moreover the setting button should change colour when hovered over. Also there should be some sort of guidance to explain to the user how to use the system.

4.2 Evaluate the solution

In the limited development time given, the system has been created, meeting most requirements set by the stakeholders.

Evaluating the solution to the success criteria…

Criteria	Met?
User is able to login with their email and password to access their account	My program has successfully achieved this. Test S1 of system testing clearly shows that the user is able to login. Moreover, end-user also shows that the stakeholders were able to login without any trouble, hence it is very easy to use login. Furthermore, iterative tests 2's show that the user can successfully login with the system guiding the user in the process, where the system informs the user when the user has entered an incorrect email or password. This was very well implemented and is fully functional. It meets all the requirements 4.
User must be able to create an account with name, email and password	My program has successfully achieved this. Test S1 of system testing clearly shows that the user is able to sign up. Moreover, end-user also shows that the stakeholders were able to create an account without any trouble, and were able to see the option to on login form, hence it is very easy to use create an account. Furthermore, iterative tests 3 show the user can create an account, and that the system only accepts valid inputs. It also shows how the system informs the user if any inputted value is invalid. This was very well implemented and is fully functional. It meets all the requirements 5.
User must be able to edit or delete their account details	My program has successfully achieved this. Test S2 and S3 of system testing clearly shows that the user is able to edit or delete their account. Furthermore, iterative tests 6 shows the system allows user to update its account, and guides the user what are the valid values they can for their username, password, and name. It also shows that the system allows users to delete their account, and confirms before deleting their account, just in case of an accidental press. This was implemented and is fully functional. It meets all the requirements 9. One thing I would change about the process of this is the button to open the window, as stated in the end-user testing by the stakeholder. All buttons in the system are intractable except for the settings logo, therefore does not make it clear if it is a button.
User must be able to create a school class with form name and year	My program has successfully achieved this. Test S4 of system testing clearly shows that the user is able to create a class. Moreover, end-user also shows that the stakeholders were able to create a class with ease, hence it is very easy to use this function. Furthermore, iterative tests 4 shows the system allows the user to create a class, with the system guiding the user what they need to input, and informing the user when they have inputted a value incorrectly. This was very well implemented and is fully functional. It meets all the requirements 12.
User must be able to edit or delete any school class	My program has successfully achieved this. Test S5 of system testing clearly shows that the user is able to update a class. Test S10 of system testing clearly shows that the user is able to delete a class. Moreover, end-user also shows that the stakeholders were able to update and delete a class with ease, hence it is very easy to use this function. Furthermore, iterative tests 7.3 to 7.8 shows the system allows the user to update its account or delete it. It also shows that the user is warned before deleting a class, and is told the amount of students which will also be deleted and which students. This was very well implemented and is fully functional. It meets all the requirements 12, 13.1, 13.2.
User must be able to increment the year of all school class by pressing a button	My program has successfully achieved this. Test S4 of system testing clearly shows that the user is able to create a class. Moreover, end-user also shows that the stakeholders were able to create a class with ease, hence it is very easy to use this function. Furthermore, iterative test 7.9 shows the system is able to increment all classes by one. This was very well implemented and is fully functional. It meets the requirement 13.3. One thing I would change about the process of this is the button to open the window, as stated in the end-user testing by the stakeholder. All buttons in the system are intractable except for the settings logo, therefore does not make it clear if it is a button.
User must be able to create a student with	My program has successfully achieved this. Test S4 of system testing clearly shows that the user is able to create a student. Moreover, end-user also shows that the stakeholders were

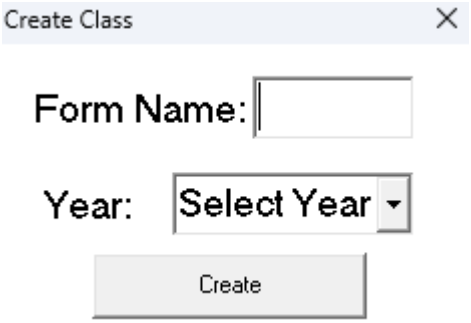
student name and class	able to create a student with ease, hence it is very easy to use this function. Furthermore, iterative tests 5 shows the system allows the user to create a student. This was very well implemented and is fully functional. It meets all the requirements 10.
User must be able to edit, share or delete any student	My program has partially achieved this. Test S7 of system testing clearly shows that the user is able to update a student. Test S9 of system testing clearly shows that the user is able to delete a student. Moreover, end-user also shows that the stakeholders were able to update and delete a student with ease, hence it is very easy to use this function. Furthermore, iterative tests 9.2's show the system is able to allow the user to edit students, and iterative tests 9.4's shows the system allows the user to delete students. This was well implemented and meets all the requirements 11, however the user is not able to share a student with another user, hence system fails to meet requirement 16.
User must be able to create a report of a student by completing a multiple choice quiz	My program has successfully achieved this. Test S6 of system testing clearly shows that the user is able to create a report. Moreover, end-user also shows that the stakeholders were able to create a report with ease, hence it is very easy to use this function. Furthermore, the iterative tests 8.1 to 8.5 shows the system allows the user to create a report by completing a multiple choice quiz. This was very well implemented and is fully functional. It meets all of the requirements 14.
User must get an option to export the report as a PDF or copy the comment	My program has successfully achieved this. Test S7 of system testing clearly shows that the user is able to export a report. Test S6 of system testing clearly shows that the user is able to copy a report comment. Moreover, end-user also shows that the stakeholders were able to copy and export a report with ease, hence it is very easy to use this function. Furthermore, iterative tests 8.7s allow the user to copy or export the report. This was very well implemented and is fully functional. It meets the requirement 15.3.
User must be able to view, edit and delete reports after they have been created	My program has partially achieved this. Test S8 of system testing clearly shows that the user is able to delete a report. Test S6 of system testing clearly shows that the user is able to view a report comment. Moreover, end-user also shows that the stakeholders were able to delete and view a report with ease, hence it is very easy to use this function. Furthermore, the iterative tests 9.1 and 9.3s shows the system is able to allow users to view and delete reports. This meets the requirement 15.1, and 15.2, however the user is limited and is not able to edit the report comment, hence failing to meet requirement 14.3. This was also mentioned in end-user testing emphasising how important this feature is to the stakeholder and will be to a user.

Effectiveness of usability features

As you can see, forms are annotated to its usability features above in usability testing.

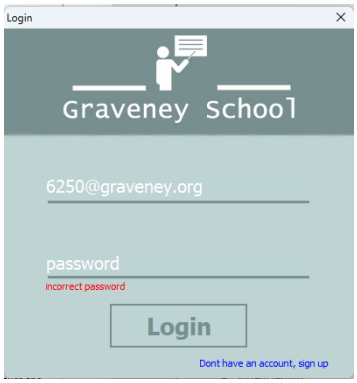
One usability feature the application has successfully met is the design and theme. The forms in the system are either with a teal or white background; a light formal colour, meeting requirement 6. The usability tests in end user testing shows that the stakeholder is satisfied with the aesthetics of the form and so the usability feature is successfully met.





These are the 2 form designs. As you can see, both have a light background. Each font colour is used at places to make it stand out from the background, making it easier for the user to read. Figure 1 also shows size of the font and the window, which meet requirement 6 and is readable by people who are short sighted, etc, as suggested by the stakeholders in the usability tests in end-user testing, hence the usability requirement is met. As stated in section 2.4, usability features, all main forms will have a heading with same background colour, maintaining the theme of the system, seen above in usability testing figure 1, 3 and 4, and further shown in system testing's above.

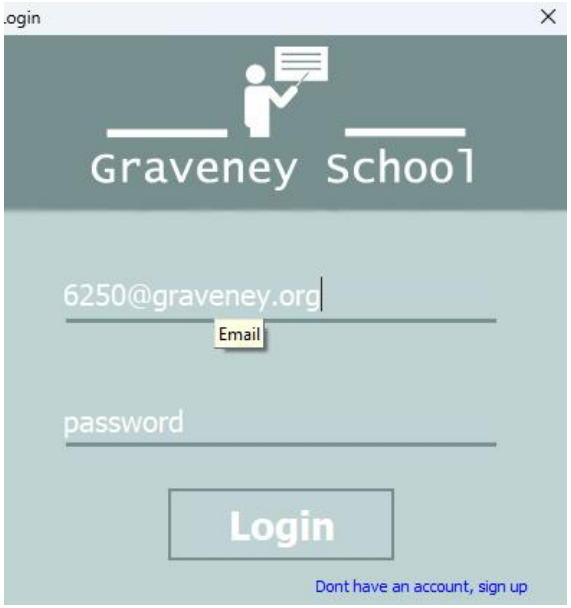
Another usability feature the application has successfully met is informing the user when invalid login details is written by displaying test in red. Informing the user the login details entered are incorrect is needed to meet requirement 4.2. The system, displays the error in red on the form rather than displaying a pop up. All users generally are to make a typo at the login. It is not ideal for the user to keep switching one of their hands from the keyboard to mouse to click ok on a pop up message at login. The usability tests in end user testing shows that the feature is met and the stakeholder is satisfied with the usability feature.



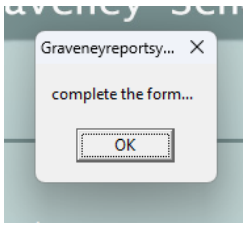
Another usability feature the application has successfully met is having each button in the system intractable as shown in Figure 1 in usability testing. This means the usability requirement form section 2.4 is met. Moreover, there is evidence from the usability tests, in end-user testing, to suggest that the user is satisfied with the idea of intractable buttons, however this requirement was not implemented for the settings icon. The stakeholder in the end-user testing, suggested that not having this feature implemented for the setting icon may confuse users, making them think that it is not a button.

Another usability feature the application has successfully met is having hints on each text box. The system always makes sure the user knows what needs to be entered in a text box, by using a label or text on top of the text box. The

default text automatically disappears when the user clicks on the text box, to avoid having the user manually delete it themselves. Moreover, the system uses Delph's inbuilt hint feature where then the mouse is on top of the text box, it informs the user what needs to be inputted. Figure 2 in usability testing shows how the system automatically removes and adds default text to text boxes, making sure the user knows what needs to be inputted at all times.



Another usability feature the application has successfully met is having a pop up box with message displayed to the user that … has been created, deleted or updated, shown in Figure 3 in usability testing. This feature confirms all actions made by the user, keeping the user updated at all times with that is going on, and insuring they are always aware the system is fully functioning. The usability tests in end user testing shows that the stakeholder is satisfied with the usability feature and so the usability feature is successfully met.



Another usability feature the application has successfully met is having a confirmation box. When the user is deleting any major items the confirmation does appear before deleting, as shown in figure 4 usability testing above, which the stakeholder is pleased with, hence the usability feature is met.

Another usability feature the application has successfully met is having all buttons appropriately named to its functionality. As you can see in the usability testing and system testing, it is quite clear what each button on a form does, making it easier for users to learn how to use the system. Moreover, in the end user testing, the stakeholder was pleased with this usability feature and has said that it makes it much easier to navigate the system, therefor the usability feature has been met.

Another usability feature the application has successfully met is having the system automatically close windows when they are not needed. This makes the user interaction with the system much nicer and navigates the user when using the system to create a report. Moreover, all windows are opened on the middle of a screen on any monitor with any resolution. In the end user testing the stakeholder was satisfied with this usability feature, hence usability feature has been met.

Further Development

Requirement 14.3 was not met due to the restrictions from hardware requirement 0. The edited report comment cannot be stored in the database due to a significant amount of space requirement increase, which would fail to meet hardware requirement 0. However it can be implemented to be stored on client side. The system can create a local folder on client side and store all edited comments in a text file with the name as the report id, which is a unique identifiable value. This will allow the user to edit reports and view them through the system whenever they want, on the same computer.

What would make the user interaction with the system even much better would be having sizable windows. Some users may prefer to full screen the window when creating a report, and would be a feature which would be preferred by a certain group. Moreover, it would allow them use the windows screen portioning features, for example, where they can drag the window and fill it on one side, and have another application on the other side.

In the future, I would like to implement a feature to import students and classes into the system. The application the school uses at the moment can export the class and students. If the system was able to import students and a class using that file, it will save a significant amount of time for users, instead of creating classes and students individually.

Requirement 16 was not met, however can be accomplished with more development time. The systems database is set to allocate students to a single class, which is allocated to a single user. I can implement a solution, to allow the users to share students, by creating a database table, Share Class, which has a relationship between staff and class. This will allow me to code a solution, where the staff are able to share classes with students.

The stakeholder requested the usability feature of intractable buttons to be implemented for the settings icon. This is to avoid confusion of it not being a button. This can be easily be implemented by opening the icon in Photoshop, changing the colour to another colour, and creating a system event procedure, that when the mouse is on top of the icon, the new created icon loads, and when it’s not, the original icon loads back into the object.

The system would be much more reasonable for the solution if it gave the user an option to save its accounts details, and had an option to log out. This way the user will not be asked to login each time. Furthermore, since the password is hashed already, there account details being stored in a text file on their drive will be secure as no one would be able to retrieve their password from it.

Limitations and potential improvements

The user is not able to create a student with the first and last name inside a class. At rare moments a single class may have two students with the same name, along with spelling. This is not a major issue but can be fixed. The user can create a student with same name, with the system making users to add the student's school identification number on create student. This will give the system a differentiating factor for the students. If the user selects a student with the same name of another student in a single class, the system can show a pop up box for the user displaying each student's id number and asking the user to select which student they were wanting to select.

The user email cannot be verified by sending an email to the user. The email system can be accomplished by either, the school giving me permission for using their SMTP server, or more development time to implement a solution where the system will allowing users to store multiple emails on the system. The system can have a primary email requirement of an outlook or Gmail, and have a secondary optional option to add their Graveney email. My database is set in a way where this will be very easy to implement, since the login details is already separate from the staff details, hence there can be a many-to-one relationship between each of them, respectfully.

The user cannot export multiple reports as a pdf. This limitation can be solved by allowing users to select multiple reports and exporting them. A newer pdf layout will be required, where each report is separated in a tabula form. The system will also need to ask the user the date which needs to be placed on the report, since multiple dates cannot be listed on a formal report. More development time would be required to implement the solution for this limitation, since it is a complex problem, and the time constraints met it could not be implemented.

The system does not give an option for the user to save its login details. This limitation can simply be solved using a text file client-side. The system can give an option to the user to opt into this feature, when the users email and password can be saved to a test file located on the user's computer. Fortunately, the system already has a hashing system, therefore it will be safe to use a text file, since no one will be able to read the password of the file. Every time the application is open, it will check if that file exists and if it does, it will auto login for the user, and display the main menu, if not it will display the login menu.

Maintenance Issues

The systems database will be stored on the schools server, allowing access to all users on the school network, and special access to users at home. The system database needs to be regularly backed up in case of data loss or corruption for any unknown reason, such as a security issue.

The system gives a feature to users that allows them to increment all classes by one, which is needed for a new school year in the month of September. Since the data for classes of years 11 and 13 is deleted, the school administrator may be required to take a backup beforehand, just in case any user wants its old data.

Since the program is a single executable file, which only runs on windows, users will be required to delete the previous version of the program and download and use the newer version, if an update occurs, only on their personal devices. For updates on the school computers, the user will not be required to do anything. This will be a single operation that the school administrator will be able to do.

The system is very easy to use, hence no manual support/ training is required for system users, however training will be required for the school administrator to update the system. It will not be too hard or require technical skills. Support will be needed to be given to users when upgrading on their personal devices. This can be done by some sort of manual.

One limitation and maintenance issue the system has is the system does not allow users to back up and restore their own data. This would make the system more user independent and effortless. This would save time for the user to rely on the school administrator, if they have backed the data up, when they have deleted something accidentally. Moreover, there are a limited amount of school administrators, hence this would be needed.